

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Система автоматизованого аналізу студентських
робіт»**

Виконав студент групи ПЗ-21-3
ОСИПЧУК Антон Олексійович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

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

«ЗАТВЕРДЖУЮ»

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

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОСИПЧУК Антон Олексійович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Система автоматизованого аналізу студентських робіт»**,
затверджена наказом закладу вищої освіти від «14» лютого 2025 р., №61/с

Вихідні дані для роботи: об'єктом дослідження є використання сучасних технологій для проектування, розробки та реалізації системи автоматизованого аналізу студентських письмових робіт, яка охоплює етапи моделювання реляційної БД, побудови REST-API, інтеграції AI-модуля Semantic Kernel та створення адаптивного веб-інтерфейсу. Предметом дослідження є методи та інструменти розробки, що забезпечують продуктивність, безпеку, масштабованість і зручність експлуатації платформи.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	17.02.2025р.	17.02.2025р.
2	Морозов А.В.	17.02.2025р.	17.02.2025р.
3	Морозов А.В.	17.02.2025р.	17.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з опису системи автоматизованого аналізу студентських письмових робіт та пояснювальної записки. Пояснювальна записка до випускної роботи містить 80 сторінок, 26 ілюстрацій та 22 таблиць.

Метою роботи є розробка інтегрованого рішення, яке забезпечує користувачам зручну платформу для завантаження та перегляду письмових робіт у форматі PDF, автоматичного аналізу їхнього змісту з використанням методів AI, генерування короткого резюме та попереднього виставлення оцінок за заданими критеріями, із подальшою можливістю ручного коригування.

Основна ідея полягає у створенні простого та інтуїтивно зрозумілого інтерфейсу, що підтримує рольову модель користувачів (студент/викладач), дає змогу студентам безпечно завантажувати роботи, а викладачам — переглядати результати аналізу, виставляти оцінки та стежити за успішністю студентів.

У рамках проєкту було визначено ключові завдання розробки, проведено аналіз функціональних вимог до системи, а також розглянуто подібні сервіси, що вже існують на ринку. Було здійснено розробку моделі даних для зберігання студентських робіт, результатів аналізу та історії оцінювання із застосуванням PostgreSQL. Була спроектована багаторівнева архітектура з інтеграцією з AI-модулем на базі Semantic Kernel для виклику LLM-моделі та обробки тексту. Для зручного користування системою було розроблено зручний та технологічний фронтенд на базі бібліотеки React.js.

КЛЮЧОВІ СЛОВА: СИСТЕМА АВТОМАТИЗОВАНОГО АНАЛІЗУ, СТУДЕНТСЬКІ ПИСЬМОВІ РОБОТИ, AI, REACT, ASP.NET CORE, SEMANTIC KERNEL, POSTGRESQL, N-TIER ARCHITECTURE.

					ІПЗ.КР.Б-121-25-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Осипчук А.О.			Система автоматизованого аналізу студентських робіт	Літ.	Арк.	Аркушів
Керівник		Морозов А.В.					3	80
						Житомирська політехніка, група ІПЗ-21-3		
Н. контр.		Тичина О.П.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a description of the system for automated analysis of student written works and an explanatory note. The explanatory note to the final thesis contains 80 pages, 26 illustrations and 22 tables.

The aim of the work is to develop an integrated solution that provides users with a convenient platform for uploading and viewing written papers in PDF format, automatically analyzing their content using AI methods, generating a short summary and preliminary grading according to specified criteria, with the subsequent possibility of manual correction.

The main idea is to create a simple and intuitive interface that supports the role model of users (student/teacher), allows students to securely upload work, and allows teachers to view the results of the analysis, assign grades, and monitor student progress.

The project identified key development tasks, analyzed the functional requirements for the system, and reviewed similar services already on the market. A data model was developed to store student work, analysis results, and assessment history using PostgreSQL. A multi-level architecture was designed with integration with an AI module based on Semantic Kernel to call the LLM model and process text. A user-friendly and technologically advanced frontend based on the React.js library was developed for easy use of the system.

KEYWORDS: AUTOMATED ANALYSIS SYSTEM, STUDENT WRITING, AI, REACT, ASP.NET CORE, SEMANTIC KERNEL, POSTGRESQL, N-TIER ARCHITECTURE.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного забезпечення	10
1.3 Вибір архітектури	15
1.4 Обґрунтування вибору інструментальних засобів	17
Висновки до першого розділу	18
РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО АНАЛІЗУ СТУДЕНТСЬКИХ РОБІТ	19
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	19
2.2 Розробка бази даних системи	23
2.3 Проектування та реалізація алгоритмів роботи системи	29
2.4 Реалізація функціоналу системи	31
2.5 Реалізація аналітичного модулю системи	39
Висновки до другого розділу.....	42
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ АВТОМАТИЗОВАНОГО АНАЛІЗУ СТУДЕНТСЬКИХ РОБІТ	43
3.1 Порядок встановлення та налаштування параметрів системи	43
3.2 Структура веб-інтерфейсу системи автоматизованого аналізу студентських робіт.....	46
3.3 Тестування роботи системи	56
Висновки до третього розділу	59
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (Application Programming Interface)

BLL – рівень бізнес-логіки (Business Logic Layer)

DAL – рівень доступу до даних (Data Access Layer)

ПЗ – програмне забезпечення

БД – база даних

СУБД – система управління базами даних

UI – користувацький інтерфейс (User Interface)

UX – користувацький досвід (User Experience)

AI – штучний інтелект (Artificial Intelligence)

ML – машинне навчання (Machine Learning)

LLM – велика мовна модель (Large Language Model)

DTO – об'єкт передачі даних (Data Transfer Object)

UoW – патерн «одиниця виміру» (Unit of Work)

DI – впровадження залежностей (Dependency Injection)

e2e – від початку до кінця (End-to-End)

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. У сучасній системі вищої освіти дедалі гостріше стоїть питання оптимізації процесу оцінювання письмових робіт студентів. З введенням дистанційних та змішаних форматів навчання навантаження на викладачів зростає, адже кожен курс може містити десятки, а то й сотні студентських робіт, які потребують детальної перевірки, аналізу та надання індивідуального зворотного зв'язку. Традиційні методи ручної перевірки займають значний час і ресурси, при цьому не завжди забезпечують сталість оцінювання: ступінь об'єктивності може змінюватися від настрою викладача, втоми або суб'єктивного сприйняття тексту.

Автоматизація аналізу письмових робіт із застосуванням сучасних технологій штучного інтелекту та обробки природної мови (NLP), що дозволить значно зменшити час, необхідний на первинну перевірку, стандартизувати критерії оцінювання та сконцентрувати увагу викладача на наданні глибшого аналітичного зворотного зв'язку. Водночас така система стає потужним інструментом для студентів, вони можуть отримувати миттєві рекомендації щодо покращення структури викладених думок, стилістики та оформлення, що сприяє підвищенню якості навчальних робіт.

Метою випускної кваліфікаційної роботи є розробка застосунку, який забезпечить автоматизований аналіз студентських письмових робіт формату PDF, генерацію резюме, попереднє виставлення оцінок за заданими критеріями та надання ролей “Студент” і “Викладач” з відповідними можливостями взаємодії.

Для досягнення поставленої мети слід виконати такі основні **завдання**:

Спочатку провести огляд і порівняльний аналіз існуючих підходів до автоматизованого оцінювання студентських робіт, визначити їхні переваги й обмеження та сформулювати ключові вимоги до нової системи.

Далі на основі отриманих результатів розробити загальну архітектуру платформи, яка включатиме окремі рівні для обробки запитів, реалізації бізнес-

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

логіки й зберігання даних, а також передбачатиме місце для підключення модуля аналізу тексту.

Наступним кроком стане поетапна реалізація серверної частини: створення API для обробки запитів користувачів, налаштування зберігання файлів і організація доступу до бази даних. Паралельно необхідно розробити клієнтський інтерфейс, який забезпечить зручну навігацію, подачу робіт та перегляд результатів аналізу.

Після цього слід інтегрувати компонент семантичного аналізу тексту – модуль, що автоматично оброблятиме завантажені документи та формуватиме попередні оцінки за заданими критеріями.

Нарешті, завершальним етапом робіт стане комплексне тестування всієї системи. Перевірка окремих сервісів, коректність взаємодії між компонентами та зручність роботи інтерфейсу в реальних сценаріях, щоб упевнитися в надійності та відповідності рішення поставленим завданням.

Об’єктом дослідження є процес автоматизованого аналізу та оцінювання студентських письмових робіт із використанням методів Обробки природної мови (Natural Language Processing, NLP) та прикладних веб-технологій.

Предметом дослідження є методи, алгоритми та інструменти розробки, що використовуються для побудови платформи, яка повинна бути не лише ефективною з технічної точки зору, але й зручною, безпечною та гнучкою у використанні.

Розроблена система може бути впроваджена в університетах та освітніх платформах для скорочення часу перевірки робіт, підвищення об’єктивності оцінювання та надання студентам корисного зворотного зв’язку. У майбутньому можливе розширення функціоналу через інтеграцію з системами антиплагіату, адаптивною системою рекомендацій та аналітикою успішності на основі машинного навчання.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези [21].

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

У сучасних умовах цифровізації освіти зростає потреба у створенні ефективних інструментів для автоматизації перевірки письмових робіт студентів. Традиційний процес оцінювання текстових завдань потребує значних витрат часу викладачів, оскільки включає аналіз структури тексту, змісту, орфографічних та граматичних помилок, а також загальної відповідності критеріям оцінювання.

Розробка веб-системи для автоматизованої перевірки студентських письмових робіт є актуальним завданням, що спрямоване на спрощення процесу аналізу текстових матеріалів, підвищення об'єктивності оцінювання та поліпшення зворотного зв'язку між викладачами та студентами.

Створений програмний комплекс дозволить автоматизувати перевірку письмових робіт на основі заданих критеріїв, а також використовувати алгоритми обробки природної мови для аналізу змісту. Окрім того, система дозволить користувачам формувати реферативний виклад роботи та надавати зворотний зв'язок користувачам.

Система буде реалізована у вигляді веб-додатку, який дозволить викладачам та студентам працювати в єдиному середовищі. Даний підхід спростить процес оцінювання та зробить навчальний процес більш ефективним.

Результатом роботи є програмний інструмент для автоматизованого аналізу студентських письмових робіт. Основною метою є полегшення перевірки документів для викладачів шляхом аналізу змісту, оцінювання певних критеріїв та автоматизованого виставлення оцінок.

Основні завдання, які система забезпечуватиме:

1. Завантаження та обробку PDF-документів із текстовим вмістом.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

2. Аналіз письмових робіт за заданими критеріями (структура, зміст, оригінальність тощо).
3. Автоматичний підсумковий виклад змісту.
4. Використання методів обробки природної мови (NLP) для семантичного аналізу.
5. Виставлення попередньої оцінки на основі аналізу.
6. Надання можливості викладачу переглянути результати та скоригувати оцінку.

Із впровадженням даної системи навчальний процес стане більш ефективним, оскільки зменшить навантаження на викладачів, дозволить швидше надавати зворотний зв'язок студентам та забезпечить високу якість автоматичного оцінювання.

1.2 Аналіз аналогів програмного забезпечення

Для ефективного вирішення задачі автоматизованого аналізу письмових робіт важливо розглянути існуючі рішення, які вже використовуються в освіті для перевірки текстових матеріалів. Нижче проаналізовано кілька категорій програмного забезпечення, що можуть бути корисними для розробки подібної системи.

Антиплагіатні сервіси

Системи для перевірки унікальності тексту, такі як Turnitin, Unicheck, Copyscape, Plagscan, дозволяють знаходити запозичення та плагіат у студентських роботах. Вони працюють на основі текстового аналізу та порівняння з базами наукових статей, книг і веб-сайтів [1].

Системи автоматизованого виявлення плагіату вирізняються високою точністю виявлення збігів завдяки доступу до великих баз даних, а також легко інтегруються з різноманітними освітніми платформами — це дозволяє вбудувати їх у вже наявні навчальні процеси без додаткових зусиль. Крім того, такі рішення автоматично генерують детальні звіти для викладачів, що значно спрощує аналіз результатів перевірки та економить час на підготовку звітності.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

AI Detector by JustDone

Maintain the authenticity of your writing by identifying AI-generated content.

✓ 79% AI content

The seaside town was a picturesque blend of old-world charm and modern amenities. Waves crashed gently against the shore, their rhythmic sound providing a soothing backdrop to the bustling boardwalk. Colorful fishing boats bobbed in the harbor, their nets filled with the day's catch. Tourists strolled along the promenade, enjoying the salty sea breeze and the vibrant atmosphere. The scent of fresh seafood filled the air, tempting passersby to stop at the various restaurants and cafes lining the streets. As the sun began to set, casting a golden glow over the water, the town came alive with music and laughter. Street performers entertained crowds, and families gathered to watch the brilliant colors of the sunset fade into twilight. It was a place where memories were made, and the worries of everyday life

79%
AI Generated Text

Double checked by

Originality.ai
 Scribbr
 GPTZero
 Copyleaks

[See Detailed Report](#)

Try our AI Text Humanizer

- ✓ Get 100% unique text with 0% AI content
- ✓ Maintain the original meaning
- ✓ Bypass all detectors
- ✓ We don't share your texts with anyone

Remove AI Content

Рис. 1.1 – Головна сторінка Turnitin

Водночас варто врахувати, що вони фокусуються переважно на пошуку запозичень і не здійснюють глибинного аналізу логіки викладення думок, а більшість комерційних продуктів пропонують обмежений безкоштовний доступ, через що повний функціонал зазвичай вимагає придбання ліцензії.

Граматичні та стилістичні аналізатори

Системи такі як Grammarly, LanguageTool, ProWritingAid допомагають перевіряти граматику, пунктуацію та стилістичні помилки. Вони працюють на основі правил та штучного інтелекту [2].

Інструменти автоматичної перевірки тексту забезпечують швидке й точне виявлення граматичних помилок, при цьому кожне виправлення супроводжується докладним поясненням, що допомагає користувачеві краще зрозуміти сутність помилки. Окрім цього, такі сервіси пропонують поради щодо вдосконалення стилю письма — від підбору більш влучних конструкцій до оптимізації ритму речень — і легко інтегруються з популярними текстовими редакторами та браузерними, що робить їх використання максимально зручним у повсякденній роботі.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

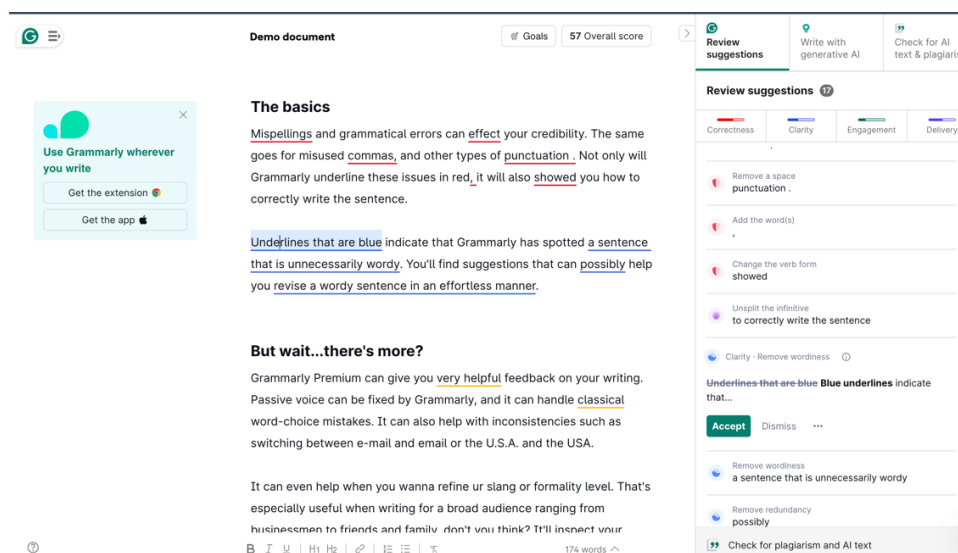


Рис. 1.2 – Головна сторінка Grammarly

Проте слід пам'ятати, що фокус цих систем обмежується формальними аспектами тексту: вони не оцінюють логіку викладення думок та глибину змісту, а деякі платформи взагалі не підтримують українську мову або мають у ній лише базовий набір функцій, тож для комплексного аналізу варто звертатися до спеціалізованих рішень або комбінувати кілька інструментів.

Засоби автоматизованого оцінювання текстів

Певні освітні платформи, наприклад EdX, Coursera, OpenAI API, використовують AI-моделі для оцінювання студентських робіт. Наприклад, GPT-4 або BERT можуть бути застосовані для аналізу логічності, послідовності викладення та відповідності темі [3].

Сучасні системи автоматизованого аналізу студентських робіт вирізняються високою гнучкістю: завдяки модульному підходу та можливості налаштування параметрів вони можуть працювати з різними дисциплінами й тематиками, від гуманітарних досліджень до технічних звітів. Використання алгоритмів штучного інтелекту дозволяє не тільки знаходити запозичення чи стилістичні помилки, але й оцінювати глибинний зміст роботи — від структурування аргументації до відповідності висловлених думок навчальним цілям. Крім того, масштабовані хмарні рішення здатні обробляти тисячі документів за короткий час, що значно скорочує навантаження на викладачів та прискорює зворотний зв'язок зі студентами.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

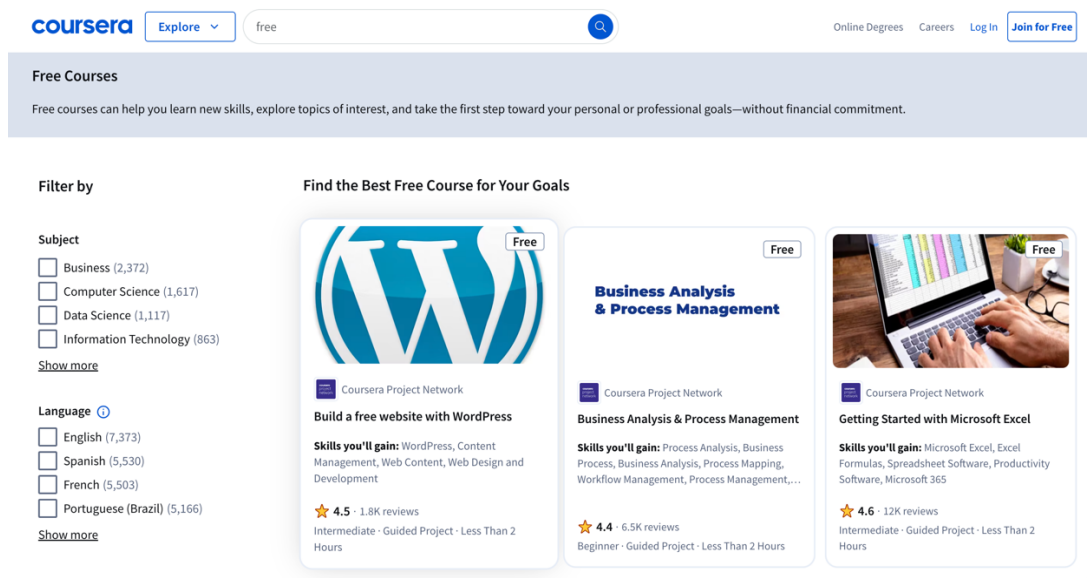


Рис. 1.3 – Головна сторінка Coursera

Водночас такі платформи мають і свої обмеження: обробка великих обсягів даних часто передбачає значні витрати на використання API та обчислювальних ресурсів, особливо якщо перевірка проводиться через зовнішні сервіси. Додатково для досягнення належної якості оцінки може знадобитися донавчання моделей під конкретні типи завдань або предметні області, що вимагає залучення фахівців з машинного навчання та додаткового часу на підготовку й тестування.

Більшість інструментів або перевіряють плагіат, або граматику, але не поєднують їх із глибоким змістовним аналізом. Деякі комплексні системи наявні на ринку надають можливість аналізу із врахуванням логічної структури та загальної якості роботи. Запропонована система інтегруватиме аналіз змісту, оцінку логічності та якості тексту за заданими критеріями, використовуючи сучасні AI-рішення на основі обробки природної мови (NLP). Це дозволить викладачам швидко й об'єктивно оцінювати якість письмових робіт, підвищуючи ефективність навчального процесу.

Для порівняння вище описаних застосунків було створено порівняльну таблицю (табл. 1.1).

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Таблиця 1.1

Порівняння застосунків аналогів

Характеристика	Антиплагіатні сервіси	Граматичні та стилістичні аналізатори	Автоматизоване оцінювання текстів
Точність виявлення	Висока точність пошуку запозичень завдяки великим БД	Точне виявлення граматичних і стилістичних помилок	Залежить від якості та тренуваності моделей
Інтеграція	Інтеграція з освітніми платформами	Інтеграція з текстовими редакторами та браузерами	Через API, вимагає початкового налаштування
Звітування	Детальні звіти для користувачів	Вбудовані пояснення помилок та рекомендації	Підсумкові оцінки та аналітичні дані
Аналіз логіки й змісту	Фокус лише на запозиченнях, без оцінки аргументації	Не аналізують логіку викладення та структуру думок	Оцінюють аргументацію, послідовність і відповідність темі
Вартість	Платні продукти з обмеженою безкоштовною версією	Є безкоштовні тарифи з обмеженнями, повний функціонал платний	Висока вартість API для великих обсягів перевірок

Аналізуючи табл. 1.1, можна констатувати, що кожна з трьох груп інструментів має свої яскраво виражені сильні та слабкі сторони. Антиплагіатні сервіси гарантовано знаходять запозичення та генерують детальні звіти, проте їхній функціонал обмежується тільки виявленням схожостей і часто вимагає придбання ліцензії для повноцінної роботи. Граматичні й стилістичні аналізатори миттєво коректують помилки та пояснюють користувачеві причину виправлень, однак не заглиблюються в логіку чи структуру тексту. Автоматизовані системи оцінювання, навпаки, здатні аналізувати аргументацію, послідовність викладу і відповідність темі в масовому режимі, але їхня реалізація може бути досить дорогою й вимагати донавчання моделей під конкретні предметні області.

З огляду на ці висновки, для розробки комплексної платформи варто поєднати найкраще з кожного підходу. З одного боку, інтегрувати потужний модуль виявлення запозичень з доступом до широких баз даних, з іншого – вбудувати сервіс граматичної та стилістичної перевірки з детальними поясненнями. Така система повинна мати відкритий API для простої інтеграції

з іншими сервісами, забезпечувати підтримку української мови на рівні з англійською і водночас залишатися економічно ефективною завдяки гібридній архітектурі. Це дозволить користувачам оперативно отримувати повноцінний аналіз письмових робіт.

1.3 Вибір архітектури

Для створення ефективної, продуктивної та надійної системи автоматизованого аналізу студентських письмових робіт було прийнято рішення побудувати її за класичною клієнт-серверною моделлю, де серверна частина реалізована у вигляді багаторівневої архітектури.

Спочатку було розглянуто два принципово різні підходи: багаторівневу (N-tier) архітектуру та мікросервісну модель. N-tier підхід запропонував чіткий поділ відповідальності між окремими шарами – від представлення даних в інтерфейсі до бізнес-логіки й доступу до сховища. Це суттєво спрощує як процес розробки, так і тестування, адже модифікації в одному шарі не породжували непередбачуваних наслідків на інших [10].

У свою чергу мікросервісна модель уявляла великий потенціал для незалежного масштабування та розгортання окремих компонентів, але її впровадження на старті могло призвести до складнощів у координації сервісів і вимагало налаштування додаткової інфраструктури для балансування навантаження та моніторингу.

Зваживши на переваги простоти й стабільності, вирішили зробити N-tier архітектуру основою серверної частини. Водночас важливі функціональні блоки (наприклад, служба авторизації чи модуль обробки текстів) за необхідності можна винести у вигляді окремих сервісів із власними інтерфейсами взаємодії.

Нижче в табл. 1.2 наведено опис рівнів багаторівневої архітектури, реалізованої в межах цього проєкту.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

Рівні архітектури

Рівень	Функціонал
Рівень представлення (Frontend/UI Layer)	Відповідає за взаємодію з користувачем: авторизація, завантаження документів, відображення результатів аналізу.
Рівень бізнес-логіки (Business Logic Layer, BLL)	Обробляє запити клієнтів, координує виклики AI-модулів для перевірки плагіату, граматики та змісту тексту.
Додатковий модуль для аналізу тексту (Semantic Analysis Layer, SAL)	Приймає в запиті список критеріїв та текст, який обробляє за допомогою AI-модуля, та видає результат аналізу
Рівень доступу до даних (Data Access Layer, DAL)	Забезпечує збереження й отримання інформації з реляційної бази даних.

На наступному рис. 1.4 зображено основні модулі та їхні взаємозв'язки: від обробки HTTP-запитів у контролерах до виконання бізнес-логіки у сервісах і взаємодії з базою даних, а також інтеграцію AI-модулів і системи зберігання файлів. Все це буде детальніше розібрано у наступних розділах [4].

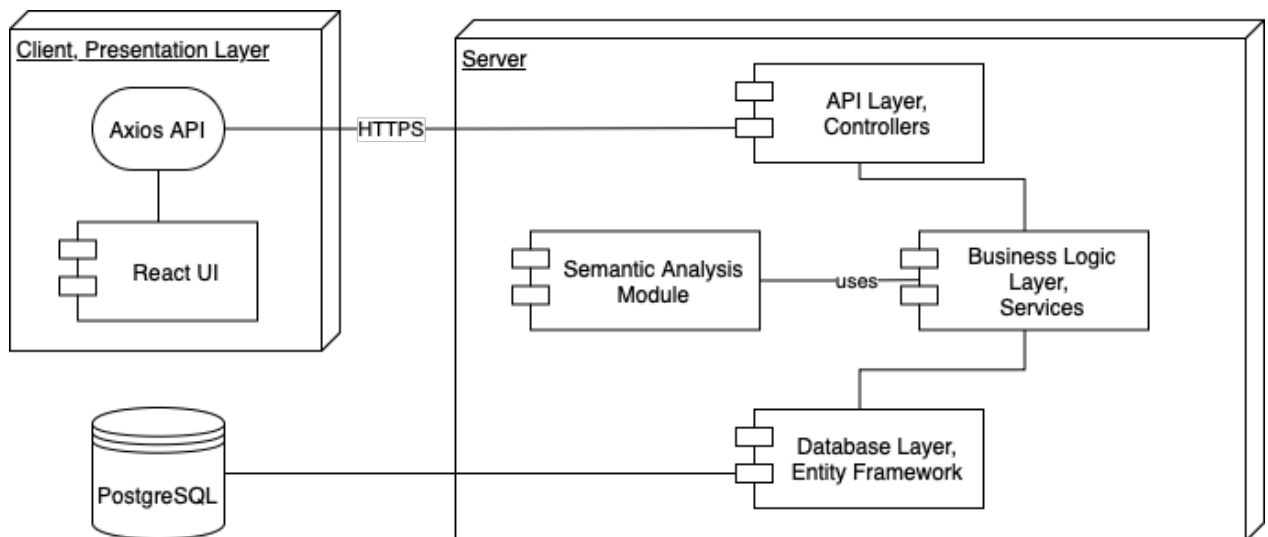


Рис. 1.4 Діаграма компонентів сервера

Завдяки такій організації серверної частини можна гнучко додавати нові сервіси, масштабувати окремі компоненти та забезпечувати стабільну роботу під високим навантаженням.

Такий розподіл шарів створює надійну основу для подальшої еволюції платформи: рівні легко тестувати та замінювати, а критичні сервіси за потреби можуть бути переведені в автономні мікросервіси без перебудови всієї системи. Це дозволяє гнучко масштабувати окремі компоненти й інтегрувати сучасні AI-та NLP-механізми для глибокого аналізу текстів [6].

1.4 Обґрунтування вибору інструментальних засобів

Для реалізації системи автоматизованого аналізу студентських письмових робіт необхідно визначити оптимальний набір інструментів та технологій, які забезпечать ефективність, продуктивність, зручність розробки та подальшу підтримку.

Вибрані технології повинні безперешкодно взаємодіяти одна з одною, підтримуючи загальноприйняті протоколи та відкриті API, щоб інтеграція завершених модулів відбувалася органічно. Водночас архітектура повинна передбачати можливість поступового розширення її компонентів та ресурсів – без серйозних заміन ключових модулів, що дозволить адаптувати платформу під нові вимоги в довгостроковій перспективі. І нарешті, обрана стекова комбінація технологій мусить відповідати сучасним стандартам інформаційної безпеки, захищати конфіденційні дані викладачів і студентів та враховувати найновіші регуляторні вимоги й рекомендації щодо шифрування, автентифікації й аудиту доступу.

Для виконання поставленого завдання вирішено використовувати програмне забезпечення Microsoft, а саме ASP.NET Core. Даний інструмент надає високу продуктивність завдяки компіляції в машинний код, гнучкість у розробці API – підтримка RESTful API [24] для інтеграції з іншими сервісами. Окрім зазначених переваг, .NET Core має вбудовану підтримку JWT, OAuth 2.0, ASP.NET Identity [25], для реалізації безпечного потоку даних на сервері.

PostgreSQL [26] надає відкритий код та стабільність – одна з найбільш надійних SQL-баз. Наведена БД має вбудовану підтримку JSONB – можливість зберігання напівструктурованих даних, та надає підтримку векторних індексів

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

(PgVector) для складніших NLP-завдань. Інтеграція з .NET через Entity Framework Core [27] – зручна ORM для роботи з базою.

Гнучкість та швидкість – динамічний інтерфейс без перезавантаження сторінки. Компонентний підхід – простота повторного використання компонентів. Строгість типізації – TypeScript допомагає уникати помилок при розробці продукту [28].

Для аналізу тексту було обрано технологію Microsoft Semantic Kernel. Глибока інтеграція з Azure AI – доступ до потужних NLP-моделей, проте даний інструмент можна використовувати і з локально розгорнутими моделями, щоб уникнути непередбачених витрат. Semantic Kernel також надає можливість використання кастомних промптів та агентів. Оптимізація для .NET – проста інтеграція через SDK.

Висновки до першого розділу

У першому розділі закладено основу для створення платформи, що має автоматизувати та спростити аналіз студентських письмових робіт за допомогою AI. У ході дослідження ми визначили ключові потреби освітнього середовища, оцінили існуючі рішення та виокремили вимоги до сучасного інтелектуального інструмента, точніше кажучи, до AI-моделі для оцінки логіки й змісту тексту. Це дозволило чітко сформулювати функціональні та нефункціональні вимоги, а також технічні умови для впровадження і масштабування системи. На основі цих вимог був вибраний набір технологій і засобів реалізації, необхідних для побудови та інтеграції різних модулів у єдину зручну платформу.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО АНАЛІЗУ СТУДЕНТСЬКИХ РОБІТ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому підрозділі розглядаються ключові сценарії взаємодії користувачів із системою та її основні компоненти. Визначені варіанти використання лягають в основу UML-діаграми варіантів (Use Case) і відображають бізнес-процеси автоматизованого аналізу студентських робіт.

Таблиця 2.1

Актори системи	
Актор	Роль у процесі
Студент	Завантажує PDF-роботу, переглядає власні подані завдання й результати аналізу та оцінювання.
Викладач	Переглядає автоматично згенерований аналіз, коригує оцінки та залишає текстові або числові коментарі.

У межах кожного з цих акторів реалізовано низку типових варіантів використання. Кожен з цих сценаріїв моделюється на діаграмі Use Case (рис. 2.1) для подальшого формального опису послідовностей [23].

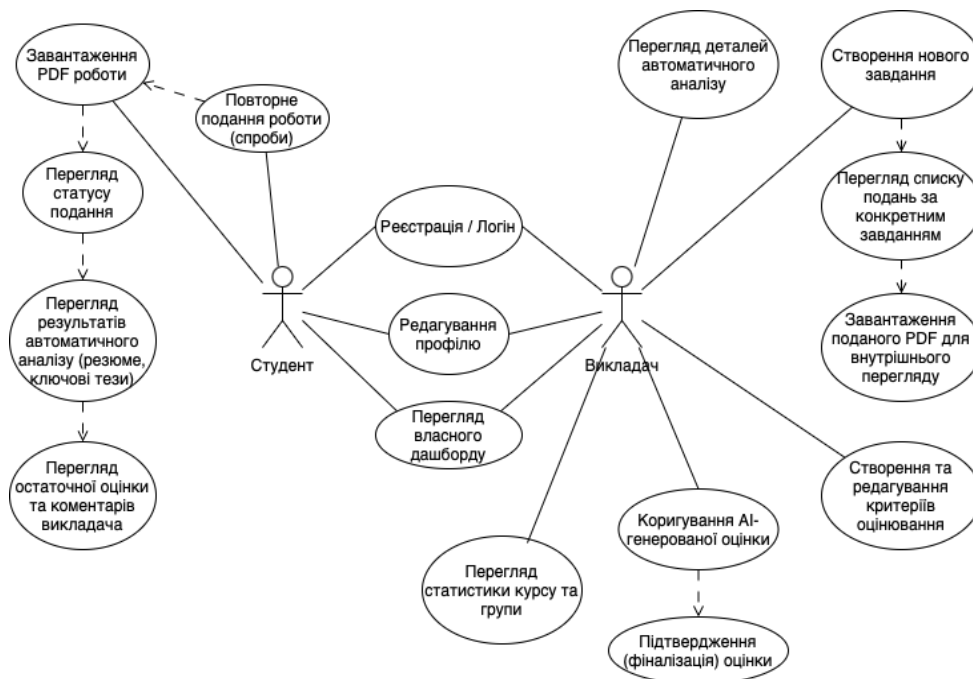


Рис. 2.1 – Діаграма варіантів використання додатку

Кожен користувач системи починає роботу з етапу реєстрації або входу. Для цього необхідно ввести існуючі облікові дані або зареєструвати новий обліковий запис, вказавши логін та пароль. Після успішної валідації даних система генерує JSON Web Token (JWT), який дозволяє надалі автентифікуватися в усіх запитах до API. У разі коректного введення інформації користувач отримує повідомлення про вдало виконаний вхід, а токен зберігається в браузері для подальшої роботи.

Студент, після входу до системи, має можливість подати свою письмову роботу у форматі PDF. Для цього він обирає потрібний файл у файловій системі, після чого фронтенд надсилає його на сервер за допомогою захищеного запиту. Сервер приймає документ, перевіряє формат і розмір файлу, створює запис у таблиці Documents із вказанням ідентифікатора користувача, часу завантаження та шляху до файлу в обраному сховищі (локально чи в хмарі). Після успішного завантаження студент бачить підтвердження прийому роботи на обробку.

В свою чергу, викладач, після того як бачить що студент завантажив роботу на оцінку, може вибрати дану роботу із списку та запустити її аналіз. Після того як запит на аналіз потрапив на сервер, запускається послідовність модулів для обробки документу: спочатку виконується витяг тексту та його нормалізація, потім текст передається до Semantic Kernel модулю для отримання і побудови стислого резюме. Результатом роботи AI-модуля стає об'єкт AnalysisResult, який зберігається в базі даних і містить згенероване резюме, набір сильних сторін роботи та попередній бал за заданими критеріями.

Після завершення автоматичного аналізу викладач може перейти до інтерфейсу перегляду результатів. Сторінка аналізу відображає короткий виклад основних ідей, список ключових тез та графічні індикатори відповідності роботи кожному з критеріїв. Завдяки чіткому і зрозумілому представленню даних викладач швидко оцінює якість робіт, не витрачаючи часу на ручний пошук ключових моментів у тексті.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Отримавши автоматично запропонований бал, викладач може змінити його відповідно до власного бачення та додати текстовий коментар. Інтерфейс дає можливість вказати остаточну оцінку, після чого зміни фіксуються у записі Grade. Після збереження викладач бачить оновлені дані в історії завдання, а студент отримує сповіщення про остаточний результат перевірки.

Бізнес вимоги

Платформа має суттєво знизити трудовитрати викладачів на первинну перевірку письмових робіт, скоротивши середній час обробки одного завдання порівняно з ручним методом. Система повинна забезпечувати єдину точку входу для всіх етапів оцінювання – від завантаження документа до фіналізації оцінки.

Вимоги користувачів

Студенти очікують простий та інтуїтивний інтерфейс завантаження PDF-файлів із чітким індикатором статусу подання і можливістю перегляду історії оцінок у вигляді списку з датами та коментарями викладача. Викладачі потребують швидкого доступу до результатів автоматичного аналізу з можливістю миттєвого коригування оцінок і коментарів, а також фільтрації подань за курсом чи групою.

Для забезпечення виконання наведеного вище функціоналу, система повинна відповідати наступним вимогам. **Функціональні вимоги** представлені у вигляді таблиці 1.1, де кожна функція представлена з її описом відповідно.

Таблиця 2.2

Функціональні вимоги

Функція	Опис
Інтерфейс користувача	Простий веб-додаток для завантаження файлів Відображення аналізу, підсумку та оцінки. Інтерактивні елементи для перегляду результатів та коригування оцінки.
Обробка документів	Видобування тексту з документів. Попередня перевірка якості тексту (розпізнавання нерозбірливих фрагментів).

Аналіз письмової роботи	Визначення ключових слів і основних тез документа. Перевірка структури (вступ, основна частина, висновки). Аналіз оригінальності за допомогою порівняння з базою попередніх робіт (векторна база даних). Визначення граматичних і стилістичних помилок.
Оцінювання	Розрахунок оцінки на основі аналізу по кожному критерію, зведення оцінки у загальну. Можливість зміни оцінки викладачем.

Нефункціональні вимоги визначають рівень якості роботи системи за показниками продуктивності, безпеки та надійності зберігання даних. Зокрема, час повної обробки одного документу від моменту отримання PDF до запису результатів аналізу не повинен перевищувати 10 секунд, хоча фактична затримка залежить від обсягу файлу та обраної AI-моделі. Доступ до платформи реалізовано через захищені механізми автентифікації користувачів із використанням JWT-токенів, що гарантує, що лише уповноважені особи можуть виконувати операції завантаження, аналізу чи адміністрування. Всі оригінальні файли та результати їхнього аналізу зберігаються в реляційній базі даних із можливістю вибіркового або масового доступу, що забезпечує швидке відновлення інформації та довготривалу архівацію.

Вимоги до апаратного забезпечення

Нижче в табл. 1.3-1.4 наведено мінімальні та рекомендовані вимоги для апаратного забезпечення серверів та клієнтського обладнання.

Таблиця 2.3

Серверні вимоги

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	4 ядра (2.4 GHz)	8+ ядер (3.0 GHz)
ОЗП	8 GB	16+ GB
Сховище	50 GB SSD	200+ GB NVMe SSD
База даних	PostgreSQL 14+	PostgreSQL 15+
Мережа	100 Mbps	1 Gbps

Таблиця 2.4

Клієнтські вимоги (для користувачів)

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Пристрій	ПК, ноутбук	ПК, ноутбук
Процесор	Intel i3 / AMD Ryzen 3	Intel i5+ / AMD Ryzen 5+
ОЗП	4 GB	8+ GB
Браузер	Chrome, Edge	Chrome, Edge

2.2 Розробка бази даних системи

Для збереження метаданих студентських робіт, результатів їхнього аналізу та користувацьких профілів обрана реляційна СУБД PostgreSQL, що гарантує надійність, транзакційну цілісність та можливість масштабування. PDF-файли робіт зберігаються або ж локально на сервері (в цілях тестування), або у хмарному сховищі, у базі ж зберігаються лише шляхи до файлів і пов'язані з ними дані.

Таблиця 2.5

Короткий опис сутностей бази даних

Таблиця	Короткий опис
User	Облікові записи користувачів: id, логін, email, повне ім'я, хеш пароля, посилання на роль
Student	Профіль студента: id, FK на Users, FK на StudentsGroup
Teacher	Профіль викладача: id, FK на Users
StudentsGroup	Групи студентів: id, назва
Course	Курси: id, заголовок, опис
Assignment	Завдання: id, FK на StudentsGroup, FK на Course, заголовок, опис, max_attempts, due_date
Document	Подання студентом роботи: id, FK на Users, FK на Assignment, номер спроби, шлях до файлу, час завантаження
Grade	Оцінювання: id, FK на Document, згенерована оцінка, змінена викладачем, статус фіналізації
GradingCriteria	Критерії оцінювання: id, FK на Users (творець), опис критерію, час створення
AnalysisResults	Результати аналізу: id, FK на Document, FK на GradingCriteria, FK на Grade, згенероване резюме, підтверджений

Для реалізації правильних зв'язків між сутностями використовувались всі три підходи для побудови взаємозв'язків у реляційних базах даних – один-до-одного, один-до-багатьох та багато-до-багатьох. Для правильної реалізації останнього було використано додаткові таблиці [29]:

1. CourseTeacher – між курсами та викладачами (course_id, teacher_id);
2. GroupCourse – між групами і курсами (group_id, course_id).

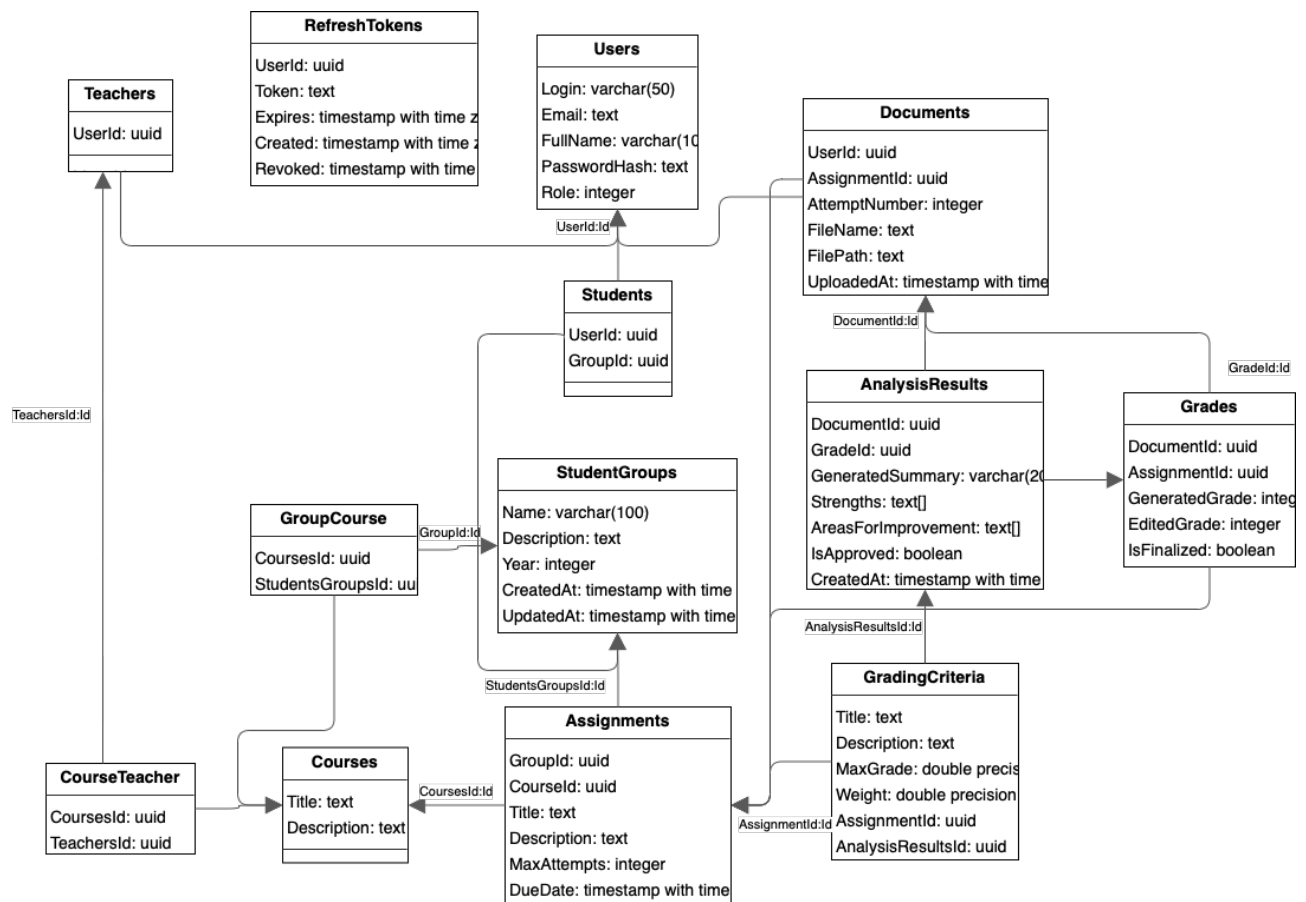


Рис. 2.2 – Діаграма бази даних

Нижче наведено детальну інформацію про кожну із створених таблиць, їх поля, типи та призначення.

Таблиця Users зберігає базову інформацію про кожного зареєстрованого користувача. Для розрізнення профілів студентів і викладачів застосовуються таблиці Student і Teacher, у яких зберігаються додаткові поля (посилання на обліковий запис у Users та налаштування групи для студента). Таблиця StudentsGroup фіксує групи студентів, а Course – перелік навчальних курсів, доступних у системі.

Таблиця 2.6

Сутність User

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор користувача
login	varchar(100)	–	–	Логін для входу в систему
email	varchar(255)	–	–	Електронна адреса користувача
fullname	varchar(255)	–	–	Повне ім'я користувача
password_hash	varchar(255)	–	–	Хеш пароля
role	integer	–	–	Номер ролі користувача (0 – Студент, 1 – Викладач)

Таблиця 2.7

Сутність Student

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор студента
user_id	uuid	–	User.id	Посилання на обліковий запис користувача
group_id	uuid	–	StudentsGroup.id	Посилання на групу студента

Таблиця 2.8

Сутність Teacher

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор викладача
user_id	uuid	–	User.id	Посилання на обліковий запис викладача

Таблиця 2.9

Сутність StudentGroup

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор
name	varchar(100)	–	–	Назва (шифр) групи студентів
description	text	–	–	Опис групи
year	integer	–	–	Рік запису групи
created_at	timestamp	–	–	Час створення групи
updated_at	timestamp	–	–	Час оновлення групи

Таблиця 2.10

Сутність Course

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор курсу
title	varchar(100)	–	–	Заголовок курсу
description	text	–	–	Опис курсу

Окремо виділена таблиця RefreshTokens, яка керує механізмом поновлення сесій. Кожен запис містить ідентифікатор користувача (UserId), сам токен, часові позначки його створення, закінчення дії та відкликання. Ця модель гарантує безпечне зберігання й перевірку довготривалих маркерів доступу [30].

Таблиця 2.11

Сутність RefreshTokens

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор
user_id	uuid	+	Users.id	Посилання на користувача
token	text	–	–	Згенерований JWT токен користувача
expires	timestamp	–	–	Час стгоряння токєну
created	timestamp	–	–	Час створення токєну
revoked	timestamp	–	–	Час оновлення токєну

Сутності Assignment, Document, Grade, GradingCriteria та AnalysisResults відповідають за весь цикл аналізу студентської роботи. Таблиця Assignment задає умови й дедлайни завдання, Document фіксує саме подання (розташування PDF-файлу і метадані завантаження). GradingCriteria містить описи правил оцінювання, а AnalysisResults зберігає результати AI-аналізу: згенероване резюме, відповідність критеріям і статус підтвердження. Таблиця Grade накопичує попередню оцінку від модуля й остаточну оцінку, відкориговану викладачем.

Таблиця 2.12

Сутність Assignment

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор завдання
group_id	uuid	–	StudentsGroup.id	Зв'язок із групою студентів
course_id	uuid	–	Course.id	Зв'язок із курсом
title	text	–	–	Назва завдання
description	text	–	–	Детальний опис завдання
max_attempts	integer	–	–	Максимальна кількість спроб подання
due_date	timestamp	–	–	Кінцева дата подання роботи

Таблиця 2.13

Сутність Document

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор подання
user_id	uuid	–	Users.id	Посилання на студента, який подав роботу
assignment_id	uuid	–	Assignment.id	Посилання на відповідне завдання
attempt_number	integer	–	–	Номер спроби подання
file_name	text	–	–	Назва документу
file_path	text	–	–	Шлях до PDF-файлу
uploaded_at	timestamp	–	–	Дата і час завантаження

Таблиця 2.14

Сутність Grade

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор запису оцінювання
document_id	uuid	–	Document.id	Посилання на подання, яке оцінюється
assignment_id	uuid	–	Assignment.id	Посилання на завдання
generated_grade	numeric(5,2)	–	–	Оцінка, згенерована AI
edited_grade	numeric(5,2)	–	–	Оцінка, відкоригована викладачем
is_finalized	boolean	–	–	Позначка остаточного затвердження оцінки

Таблиця 2.15

Сутність GradingCriteria

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор критеріїв оцінювання
assignment_id	uuid	–	Assignment.id	Посилання на завдання
analysis_results_id	uuid	–	AnalysisResults.id	Логін для входу в систему
title	text	–	–	Електронна адреса користувача
description	text	–	–	Повне ім'я користувача
max_grade	double	–	–	Хеш пароля
weight	double	–	–	Хеш пароля

Таблиця 2.16

Сутність AnalysisResults

Поле	Тип	ПК	ФК	Призначення
id	uuid	+	–	Унікальний ідентифікатор результату аналізу
document_id	uuid	–	Document.id	Посилання на подання для аналізу
grade_id	uuid	–	Grade.id	Посилання на запис оцінки
generated_summary	text	–	–	Стислий виклад змісту роботи, згенерований AI
strengths	text[]	–	–	Масив сильних сторін роботи
areas_for_improvement	text[]	–	–	Масив рекомендацій
is_approved	boolean	–	–	Позначка, що результат аналізу підтверджено викладачем
created_at	timestamp	–	–	Дата та час створення запису результату аналізу

Для реалізації складних зв'язків (багато-до-багатьох) використовуються таблиці CourseTeacher та GroupCourse. CourseTeacher пов'язує сутності курсів з викладачами, а GroupCourse – групи з курсами. Завдяки цим таблицям система може гнучко розподіляти права доступу та відображати навчальні плани, не дублюючи даних.

Таблиця 2.17

Сутність CourseTeacher

Поле	Тип	ПК	ФК	Призначення
course_id	uuid	+	Course.id	Посилання на курс
teacher_id	uuid	+	Teacher.id	Посилання на викладача

Таблиця 2.18

Сутність GroupCourse

Поле	Тип	ПК	ФК	Призначення
group_id	uuid	+	StudentsGroup.id	Посилання на групу студентів
course_id	uuid	+	Course.id	Посилання на курс

Для роботи з базою даних використано Entity Framework Core – ORM від .NET, що значно спрощує налаштування доступу до PostgreSQL та управління міграціями схеми. У проєкті визначено клас ApplicationDbContext, в якому через властивості DbSet<T> описано всі сутності (Users, Courses, AnalysisResult тощо), а підключення рядка з'єднання винесено у appsettings.json і підключається в Program.cs.

Щоб ізолювати запити до даних від бізнес-логіки, у шарі Data Access Layer реалізовано патерн Repository: для кожної сутності створено інтерфейс IRepository<T> із загальними методами CRUD, а конкретні клас GenericRepository<T> із реалізацією цих методів за допомогою EF Core (DbContext.Set<T>()). Використання патерну Unit of Work забезпечує групування декількох операцій у єдину транзакцію: через інтерфейс IUnitOfWork, який інкапсулює екземпляр ApplicationDbContext, ми контролюємо момент збереження змін (SaveChangesAsync) та гарантуємо атомарність набору дій.

Реєстрація контексту, репозиторіїв і UoW у DI-контейнері дозволяє в сервісному шарі просто використовувати необхідні залежності та працювати з даними через високорівневі сервіси, не використовуючи сирі запити SQL. Це рішення спрощує тестування (можливість мокати репозиторії та UoW) і робить код більш підтримуваним і зрозумілим.

2.3 Проєктування та реалізація алгоритмів роботи системи

Передача даних від клієнта до користувача реалізована за класичною багаторівневою схемою. HTTP-запит надходить на API-контролер, який через вбудовану систему маршрутизації ASP.NET Core визначає потрібний метод контролера. Далі контролер викликає відповідний сервіс бізнес-логіки (наприклад, AnalysisService), в якому застосовуються правила валідації та перетворення вхідних даних. Для взаємодії із сховищем даних сервіс звертається до репозиторію, реалізованого через EF Core і патерн Repository/Unit Of Work, що інкапсулює ApplicationDbContext. Репозиторій формує

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

LINQ-запит, EF Core перетворює його на SQL і виконує над PostgreSQL, після чого результат повертається знизу-вверх до сервісу й надсилається назад контролеру. Контролер пакує дані у відповідь у форматі JSON і повертає клієнту [7].

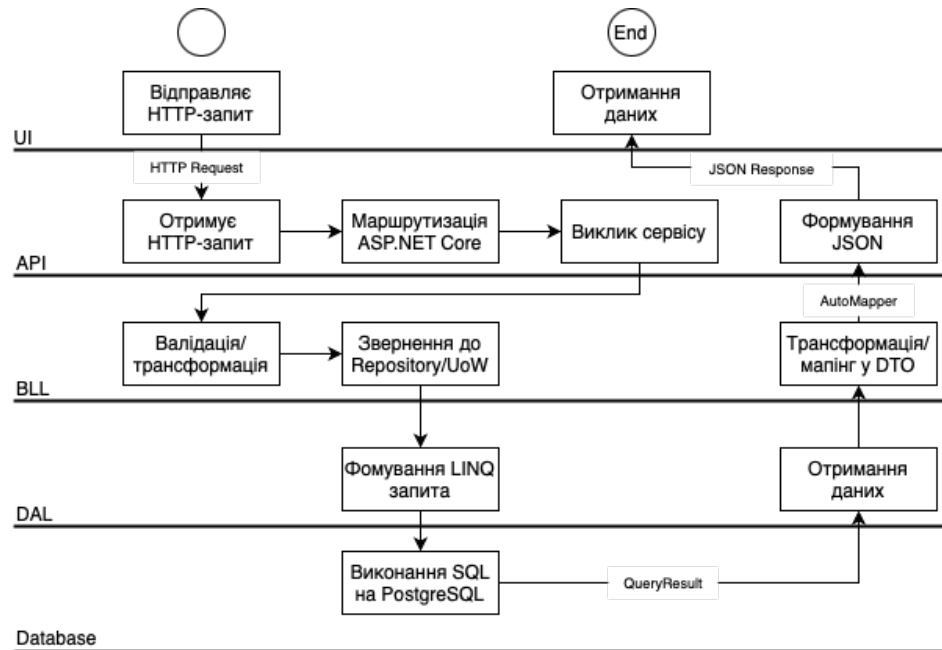


Рис. 2.3 – Діаграма активності завантаження та обробки документів

Завдяки такій архітектурі додавання нових сценаріїв відбувається швидко та безболісно: у разі потреби в новому API-ендпоінті достатньо зареєструвати маршрут у контролері, реалізувати бізнес-логіку в новому методі сервісу, за необхідності доповнити інтерфейс репозиторію новим методом доступу до даних і переходити до тестування нового функціоналу. Це гарантує гнучкість системи та простоту розширення її функціоналу без втручання в існуючий код.

Основний алгоритм обробки студентських робіт починається з отримання PDF-файлу як єдиного вхідного параметра та закінчується набором вихідних даних: стислим резюме, сильними сторонами викладення та списком на покращення та попереднім балом за критеріями. Спершу виконується витяг тексту з документа за допомогою PDF-парсера, що повертає один суцільний рядок тексту. Отриманий текст передається у модуль Semantic Kernel, який генерує стислий виклад основних тез.

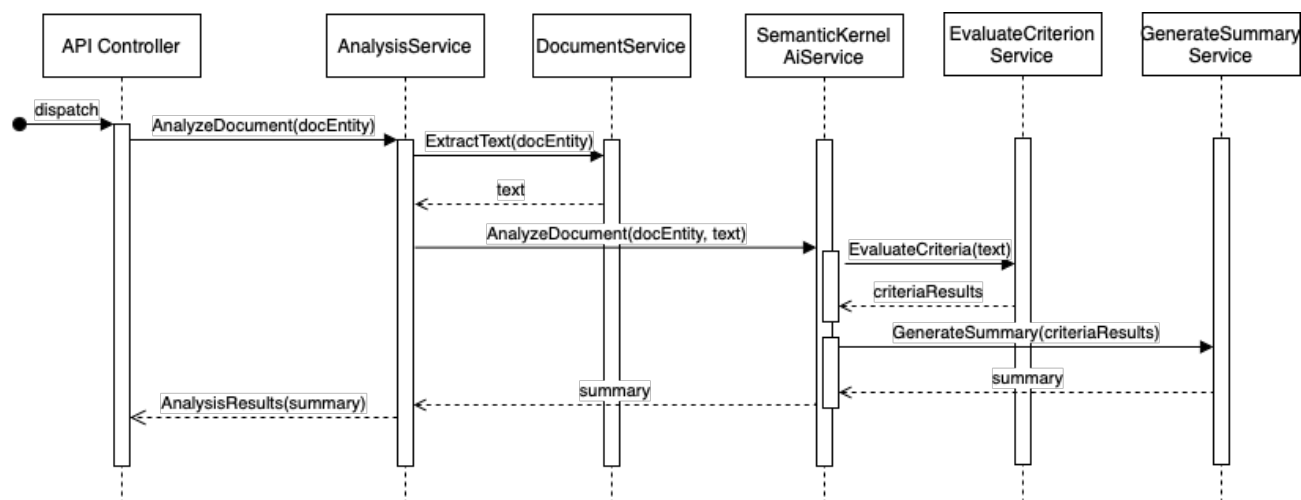


Рис. 2.4 – Діаграма послідовності завантаження та обробки документів

У майбутньому алгоритмічну лінію можна доповнити підтримкою нових мов через додаткові модулі обробки тексту (наприклад, OCR для аналізу картинок), або додати критерії, які аналізують окремі спеціальні блоки тексту (наприклад, результат виконання це блок коду наведений в документі, який можна запустити та перевірити на помилки). Ці точки розширення закладені в архітектурному дизайні, що дозволяє впроваджувати нові компоненти без руйнівних змін існуючої архітектури.

2.4 Реалізація функціоналу системи

Серверну частину системи реалізовано за принципом багаторівневої (N-tier) архітектури, що дозволяє чітко відокремити зони відповідальності й полегшити підтримку та тестування кожного рівня. Угорі знаходяться контролери ASP.NET Core Web API, які приймають HTTP-запити, проводять базову валідацію через атрибути та передають обробку далі. У шарі бізнес-логіки (BLL) згруповано всі сервіси: AuthService для реєстрації, входу й оновлення токенів, AssignmentService для CRUD-операцій із завданнями, DocumentService для завантаження PDF-файлів і FileStorageService, який працює із локальним збереженням файлів, а також AnalysisService, що інкапсулює виклики Semantic Kernel і формування попереднього результату аналізу [5].

Доступ до бази даних виконано через Entity Framework Core з реалізацією патерну Repository: кожна сутність має окремий репозиторій, що інкапсулює стандартні операції Create/Read/Update/Delete, а також спеціалізовані методи для складних запитів. Це дозволяє відокремити написання LINQ-виразів від бізнес-логіки та легко замінити сховище даних без зміни шарів вище.

Нижче наведено розгорнутий приклад алгоритму для роботи з користувачами, який демонструє послідовність обробки запиту “GetUserById” через усі рівні архітектури: від контролера до репозиторію, із застосуванням патернів Result, Repository/UoW та DI.

Перший блок коду демонструє контролер UserController, який виступає точкою входу в N-tier архітектуру для HTTP-запитів, пов’язаних із користувачами. Атрибут [ApiController] забезпечує автоматичну валідацію моделі та формування коректних HTTP-відповідей, а [Route("api/[controller]")] встановлює маршрут до контролера. У методі GetById застосовано атрибут [Authorize], що вимагає дійсний JWT для доступу, після чого з токена витягується ідентифікатор поточного користувача. Контролер делегує обробку запиту сервісу _userService, отримує об’єкт Result<UserDto> і, залежно від властивості IsSuccess, повертає або помилку BadRequest, або успішний відповідь Ok з даними користувача:

```
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    private readonly IUserService _userService;
    // DI через конструктор
    public UserController(IUserService userService)
    {
        _userService = userService;
    }

    // Ендпоінт із захистом JWT
    [Authorize]
    [HttpGet("{id:guid}")]
    public async Task<ActionResult> GetById(Guid id)
    {
        var requesterId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var result = await _userService.GetUserByIdAsync(id, requesterId);
```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

```

        if (!result.IsSuccess)
            return BadRequest(new { error = result.Error });

        return Ok(result.Value);
    }
}

```

Наступний фрагмент коду – це реалізація `UserService`, яка містить бізнес-логіку для прецеденту «отримати користувача за ідентифікатором». Сервіс використовує `IUnitOfWork` для доступу до репозиторіїв і `IMapper` для перетворення сутності в DTO. У методі `GetUserByIdAsync` відбувається запит до репозиторію `Users` через `UoW`, перевірка наявності користувача, а також порівняння запитуваного й поточного ідентифікаторів для контролю доступу. У разі успіху сутність мапиться в DTO і повертається через фабрику `Result.Ok`, а у випадку помилки формується єдинообразний `Result.Fail`:

```

public class UserService : IUserService
{
    private readonly IUnitOfWork _uow;
    private readonly IMapper _mapper;

    public UserService(IUnitOfWork uow, IMapper mapper)
    {
        _uow = uow;
        _mapper = mapper;
    }

    public async Task<Result<UserDto>> GetUserByIdAsync(
        Guid userId, string? requesterId)
    {
        var user = await _uow.Users.GetByIdAsync(userId);
        if (user == null)
            return Result.Fail<UserDto>("Користувача не знайдено");

        if (requesterId != user.Id.ToString())
            return Result.Fail<UserDto>("Доступ заборонено");

        var dto = _mapper.Map<UserDto>(user);
        return Result.Ok(dto);
    }
}

```

Далі наведено клас `UnitOfWork`, що реалізує патерн Unit of Work [11]. Він інкапсулює один екземпляр `AppDbContext` і надає властивість `Users`, яка є екземпляром `IUserRepository`. Метод `SaveChangesAsync` дозволяє комітити усі накопичені через репозиторії зміни одним викликом, забезпечуючи атомарність операцій і єдину точку збереження в базу:

```

public class UnitOfWork : IUnitOfWork
{
    private readonly AppDbContext _context;
    public IUserRepository Users { get; }

    public UnitOfWork(AppDbContext context, IUserRepository userRepo)
    {
        _context = context;
        Users = userRepo;
    }

    public async Task<int> SaveChangesAsync() =>
        await _context.SaveChangesAsync();
}

```

Наступний фрагмент – це репозиторій UserRepository [12], який реалізує інтерфейс IUserRepository і відповідає за роботу з сутностями User у базі через EF Core. Метод GetByIdAsync використовує FindAsync на DbSet<User>, що автоматично генерує SQL-запит до таблиці Users і повертає сутність або null. Репозиторії інкапсують усі операції CRUD і дозволяють відокремити логіку доступу до даних від бізнес-шару:

```

public class UserRepository : IUserRepository
{
    private readonly AppDbContext _context;

    public UserRepository(AppDbContext context)
    {
        _context = context;
    }

    public async Task<User?> GetByIdAsync(Guid id) =>
        await _context.Users.FindAsync(id);
}

```

Далі блок коду описує шаблон Result<T> [13], який уніфікує повернення даних із методів сервісів. Клас містить булеве поле IsSuccess, значення поля Value для даних у разі успіху та рядок Error для повідомлення про помилку. Методи Ok і Fail виступають у ролі фабрик для швидкого створення екземплярів результату. Завдяки цьому шаблону контролери легко обробляють різні сценарії і повертають клієнту єдиний формат відповіді:

```

public class Result<T>
{
    public bool IsSuccess { get; }
}

```

```

public T? Value { get; }
public string? Error { get; }

private Result(bool isSuccess, T? value, string? error)
{
    IsSuccess = isSuccess;
    Value = value;
    Error = error;
}

public static Result<T> Ok(T value) =>
    new Result<T>(true, value, null);

public static Result<T> Fail(string error) =>
    new Result<T>(false, default, error);
}

```

Наступний блок демонструє налаштування Dependency Injection [8] у Program.cs. Сервісний контейнер реєструє ApplicationDbContext з рядком підключення до PostgreSQL, потім прив'язує інтерфейси IUserRepository, IUnitOfWork та IUserService до конкретних реалізацій, а також додає AutoMapper. Це дозволяє використовувати DI-конструктори у контролерах і сервісах, спрощує підключення мок-об'єктів для тестування та забезпечує високу модульність коду:

```

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseNpgsql(builder.Configuration.GetConnectionString("Default")));

builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddAutoMapper(typeof(Program));

```

Таким чином кожний функціонал у системі реалізовано за єдиним шаблоном: для кожного прецеденту визначається контролер, що відповідає за прийом HTTP-запиту та авторизацію, далі запит делегується відповідному BLL-сервісу, який через Unit of Work і репозиторії звертається до EF Core для читання чи запису даних, застосовує бізнес-логіку та повертає результат у єдиному форматі через патерн Result. Dependency Injection (DI) гарантує, що на кожному рівні можна легко замінити реалізацію (наприклад підмінити репозиторій або мок-сервіс у тестах), а шаблон «контролер – сервіс –

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

репозиторій» дозволяє безболісно розширювати функціонал: додається новий ендпоінт у контролері, новий метод у сервісі, та при необхідності – відповідний метод у репозиторії. За таким же принципом побудовані й решта компонентів системи – AssignmentService, DocumentService, AnalysisService та ін., що забезпечує загальну гнучкість, зрозумілість та простоту підтримки коду.

Укладення та тестування контрактів API здійснено за допомогою OpenAPI (Swagger). У ній зібрані всі запити до кінцевих точок, від реєстрації й аутентифікації до завантаження документів, запуску аналізу та коригування оцінок.

Далі наведено розподіл усіх кінцевих точок (ендпоінтів) за функціональними групами. Кожна група відповідає за певний набір операцій у системі.

Таблиця 2.19

Групи API-ендпоінтів

Методи	Префікс маршруту	Опис групи
POST	/api/auth	Реєстрація користувача, вхід у систему та оновлення JWT-токена через refresh-токен
GET	/api/users	Отримання даних поточного користувача, списків викладачів, студентів та учасників груп
POST, GET	/api/courses	Створення нових курсів; отримання інформації про курс, курси викладача або курси студента
POST, GET	/api/assignments	Створення завдань для групи; отримання деталей конкретного завдання та списку завдань груп
POST, GET	/api/criteria	Створення критеріїв оцінювання; запит інформації про один критерій або всі критерії
POST, GET	/api/document	Завантаження PDF-файлів у систему; отримання метаданих одного подання або всіх подань користувача
POST, GET, PUT	/api/grading	Запуск автоматичного аналізу документу; запит результатів аналізу; редагування та фіналізація оцінок
POST, GET, PUT	/api/groups	Створення, редагування, отримання груп; запис студентів в групи, чи групи на курси

При роботі з API в клієнтській частині важливо налаштувати єдиний HTTP-клієнт, який буде автоматично додавати необхідні заголовки та обробляти помилки, а також створити набір повторно використовуваних хуків для авторизації та отримання поточного користувача. Нижче наведено декілька блоків коду які демонструють роботу із API на клієнті.

У першому блоці коду ми створюємо єдиний інстанс Axios, налаштований на роботу з API. Виклик `axios.create` задає базовий URL `"/api"`, стандартний заголовок `"Content-Type": "application/json"` та опцію `withCredentials: true`, щоб браузер автоматично передавав дані авторизації з кожним запитом. Далі додано перехоплювач запитів (`interceptors.request`), який перед відправленням звернення перевіряє наявність `accessToken` у `localStorage` і, якщо він є, додає його до заголовку `Authorization: Bearer <token>`. У відповідному перехоплювачі відповідей (`interceptors.response`) відстежується помилка `401 Unauthorized` – якщо таке трапляється вперше, скрипт самостійно робить POST на `/auth/refresh` із `refreshToken`, зберігає нові `accessToken` і `refreshToken` та повторює початковий запит із оновленими заголовками. Це гарантує, що клієнт ніколи не опиниться у стані “відключеної” сесії без явного `logout`:

```
import axios from "axios";

export const api = axios.create({
  baseURL: "/api",
  headers: { "Content-Type": "application/json" },
  withCredentials: true,
});

api.interceptors.request.use(config => {
  const token = localStorage.getItem("accessToken");
  if (token && config.headers) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Наступний фрагмент містить прості оболонки для викликів API, в яких ми прагнемо мінімізувати повторення коду й обробку помилок. Функція `getCurrentUser` робить GET-запит на `/users/me` і повертає результату власне `response.data`, тобто дані користувача. Аналогічно, `login` відправляє об’єкт

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

LoginDto на /auth/login, чекає відповіді з токенами та об'єктом користувача, а logout робить POST на /auth/logout, після чого викликає localStorage.clear(), щоб видалити всі токени й інформацію про сесію:

```
export const getCurrentUser = () =>
  api.get<UserDto>("/users/me").then(res => res.data);

export const login = (dto: LoginDto) =>
  api.post<AuthResponse>("/auth/login", dto).then(res => res.data);

export const logout = () =>
  api.post("/auth/logout").then(() => {
    localStorage.clear();
  });
```

Далі ми збираємо ці API-функції у єдиний хук useAuth, який зручний для будь-якого компонента React. Спочатку отримуємо екземпляр queryClient з React Query, щоб працювати з кешем. Мутація loginMutation викликає login й у своєму колбеці onSuccess зберігає отримані accessToken та refreshToken у localStorage і оновлює кеш ключем "currentUser". logoutMutation виконує logout та у своєму onSuccess очищує увесь кеш, щоб при наступних запитах по currentUser була реактивна зміна. Нарешті, currentUserQuery робить запит getCurrentUser, але лише якщо в localStorage вже є accessToken, щоб не робити зайвих запитів після logout. У результаті інтерфейс отримує об'єкт user зі свіжими даними, а всі мутації й запити централізовано керуються через React Query:

```
export const useAuth = () => {
  const qc = useQueryClient();

  const loginMutation = useMutation(login, {
    onSuccess: ({ token, refreshToken, user }) => {
      localStorage.setItem("accessToken", token);
      localStorage.setItem("refreshToken", refreshToken);
      qc.setQueryData("currentUser", user);
    },
  });

  const logoutMutation = useMutation(logout, {
    onSuccess: () => qc.clear(),
  });

  const currentUserQuery = useQuery("currentUser", getCurrentUser, {
    enabled: !!localStorage.getItem("accessToken"),
  });
```

```

    });

    return {
        user: currentUserQuery.data,
        loginMutation,
        logoutMutation,
        currentUserQuery,
    };
};

```

2.5 Реалізація аналітичного модулю системи

Вище ми детально розглянули ключові етапи створення серверного функціоналу та продемонстрували, як клієнтські компоненти взаємодіють із розробленим API. Наступним етапом перейдемо до опису семантичного модуля – компонента SemanticAnalysis, який реалізовано на базі передової платформи Microsoft Semantic Kernel [16].

Модуль SemanticAnalysis відповідає за весь семантичний аналіз текстових документів у системі: від оцінки відповідності заданим критеріям до генерації короткого резюме змісту. Реалізований за шаблоном service-oriented, він інкапсулює усі нюанси взаємодії з LLM через Microsoft Semantic Kernel, роблячи виклики до AI максимально прозорими для решти бізнес-логіки.

У центрі кожного сервісу лежить фабрика SemanticKernelFactory, яка на основі конфігурацій створює екземпляр ядра Semantic Kernel із усіма потрібними розширеннями (логування, OpenAI-агенти тощо). Кожний конкретний сервіс, наприклад EvaluateCriterionService або GenerateSummaryService, успадковує спільний базовий клас BasePromptService і реалізує інтерфейс IPromptService<TRequest, TResult>. У конструктор сервісу інжектяться SemanticKernelFactory та IPromptBuilder, що дозволяє абстрагуватися від деталей побудови підказок і зосередитися на логіці перетворення вхідних даних у ключ–значення для LLM. Зразок коду для відповідного сервісу наведено нижче:

```

public class EvaluateCriterionService(
    SemanticKernelFactory semanticKernelFactory,
    IPromptBuilder promptBuilder
) : BasePromptService, IPromptService<EvaluateCriteriaRequest, EvaluateCriteriaResponse>

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

```

{
    protected override string GetYamlFileName() => "EvaluateCriterion.prompt.en.yaml"

    public async Task<EvaluateCriteriaResponse?> ExecuteAsync(
        EvaluateCriteriaRequest request,
        CancellationToken cancellationToken = default
    )
    {
        var kernel = semanticKernelFactory.CreateBasic();
        var kernelFunction = kernel.CreateFunctionFromPromptYaml(
            PromptYaml,
            promptTemplateFactory: new HandlebarsPromptTemplateFactory());

        var input = promptBuilder.BuildCriterionInputs(request);

        var functionResult = await kernelFunction.InvokeAsync(
            kernel, input, cancellationToken);
        var result = functionResult.GetValue<string>();

        return DeserializeJsonFromResponse<EvaluateCriteriaResponse?>(result);
    }
}

```

Для кожної задачі визначено окремий YAML-шаблон підказки, ім'я якого повертає метод `GetYamlFileName()`. Далі алгоритм сервісу такий:

1. Фабрика створює базове ядро `Semantic Kernel`.
2. За допомогою `CreateFunctionFromPromptYaml` із `Handlebars`-фабрикою формується функція для виклику LLM.
3. `IPromptBuilder` перетворює вхідний запит у словник параметрів.
4. Ця функція викликається асинхронно, і з рядка відповіді JSON десеріалізується у відповідний об'єкт.

Такі чіткі контракти й єдиний шаблон дозволяють додавати нові сервіси, достатньо створити новий клас-сервіс із власним YAML-шаблоном і `builder`-ом.

Нижче наведено приклад організації YAML-шаблону [20]:

```

name: EvaluateCriteria
description: |
    Evaluates a student document against multiple grading criteria and assigns a score to each.
template_format: handlebars

template: |
    You are an academic evaluator assessing a student submission.

# Student Submission
{{DocumentText}}

```

```
# Grading Criteria
{{#each Criteria}}
## Criterion {{this.Title}} (weight {{this.Weight}})
Description: {{this.Description}}
{{/each}}
```

```
# Instructions
For each criterion aboveЖ
```

```
# Output Format:
{
  "criteriaResults": [
    {
      "score": number,
      "weight": double,
    },
    ...
  ]
}
```

Output =>

```
input_variables:
- name: DocumentText
  description: The student's document text to evaluate
  is_required: true
- name: Criteria
  description: Array of criteria objects with id, title, description, and weight
  is_required: true
```

У цьому шаблоні у блоці `template` задається повний контекст: спочатку „Student Submission“ із вставленим текстом документа (`{{DocumentText}}`), потім перераховуються усі критерії оцінювання через `Handlebars`-цикл `{{#each Criteria}}`. Далі в секції «Instructions» чітко вказано, як LLM має згенерувати JSON-об’єкт з полем `criteriaResults`, в якому для кожного критерію вказані його `id`, `title`, `weight`, `score`, обґрунтування, сильні сторони та пропозиції з покращення.

Під час виконання `EvaluateCriterionService` цей YAML-шаблон підвантажується методом `GetYamlFileName()`, а фабрика `SemanticKernelFactory.CreateBasic()` будує ядро, яке через `CreateFunctionFromPromptYaml` перетворює шаблон і підстановки на функцію LLM. Значення полів `DocumentText` і `Criteria` формуються класом-будівником `IPromptBuilder` у методі `BuildCriterionInputs`. Після асинхронного виклику

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

kernelFunction.InvokeAsync() отриманий рядок JSON автоматично десеріалізується у EvaluateCriteriaResponse завдяки спільному методу DeserializeJsonFromResponse.

Таким чином, завдяки чіткому поділу на шаблон (YAML), фабрику ядра і сервіс, ми досягаємо високої гнучкості: змінюючи лише текст шаблону чи додаючи нові властивості в секцію input_variables, можна коригувати поведінку LLM без жодних змін у коді сервісів.

Висновки до другого розділу

У цьому розділі ми послідовно відпрацювали всі ключові етапи створення платформи автоматизованого аналізу студентських робіт. Спершу чітко окреслили бізнес- і користувацькі вимоги і задокументували їх у UML-діаграмах прецедентів. Далі спроектували реляційну базу даних на PostgreSQL та налаштували доступ до неї через EF Core із патернами Repository і Unit of Work, що гарантує чистий шар бізнес-логіки й легкість модульного тестування.

Серверна частина, побудована за N-tier архітектурою, розділила обов'язки між контролерами ASP.NET Core (обробка HTTP-запитів і авторизація), сервісами BLL (Auth, Assignment, Document, Analysis) та DAL із LINQ-запитами до БД. Клієнт на React із React Query та налаштованими Axios-перехоплювачами автоматично вставляє токени й оновлює їх при 401-й відповіді, а кешування знижує число повторних звернень і покращує відчуття швидкості.

Особливу увагу приділено модулю SemanticAnalysis, де кожен сервіс (оцінка критеріїв, генерація резюме тощо) живить ядро Semantic Kernel готовими інструкціями з YAML-шаблонів, а результат LLM десеріалізується у строго типізовані відповіді. Така архітектура гарантує високу гнучкість – для додавання нового AI-функціоналу достатньо створити новий YAML-шаблон і сервіс, без додаткових змін.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ АВТОМАТИЗОВАНОГО АНАЛІЗУ СТУДЕНТСЬКИХ РОБІТ

3.1 Порядок встановлення та налаштування параметрів системи

Перед запуском усіх компонентів платформи необхідно підготувати середовище, яке складається зі структури бази даних, опціонального сховища файлів (Blob Storage), конфігураційних файлів для серверної й клієнтської частин та налаштування AI-модулів.

Спочатку потрібно встановити PostgreSQL та створити порожню базу (наприклад, sgs_db). У корені серверного проєкту маємо переконатись, що у файлі appsettings.json в секції ConnectionStrings прописаний рядок підключення:

```
"ConnectionStrings": {  
  "DefaultConnection":  
    "Host=localhost;Port=5432;Database=sgs_db;Username=sgs_user;Password=YourPassword"  
}
```

Після цього з терміналу маємо виконати міграції EF Core:

```
cd SGS.Infrastructure // коренева папка проєкту із DbContext  
dotnet ef database update
```

За замовчуванням використовується дисковий провайдер (Provider: "Disk", використано для тестування), який зберігає PDF у папці, вказаній у параметрі BasePath. Якщо наявне підключення Azure Blob Storage, потрібно змінити Provider на "Blob" та заповнити секцію AzureStorage:

```
"Storage": {  
  "Provider": "Azure"  
},  
"AzureStorage": {  
  "ConnectionString": "<your-azure-connection-string>",  
  "ContainerName": "student-submissions"  
}
```

У файлі appsettings.json (для продакшену) і appsettings.Development.json (для локальної розробки) задаються: JwtSettings (секретний ключ, терміни життя токенів), SemanticKernelSettings (Endpoint, ApiKey, DeploymentName, Language), Logging (рівні логування):

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

```
"JwtSettings": {
  "Secret": "YourVeryStrongSecretKey",
  "Issuer": "SGSServer",
  "Audience": "SGSClient",
  "ExpirationMinutes": 15,
  "RefreshTokenExpirationDays": 7
},
"SemanticKernelSettings": {
  "Endpoint": "https://your-azure-openai-endpoint.openai.azure.com/",
  "ApiKey": "YOUR_AZURE_OPENAI_KEY",
  "DeploymentName": "gpt-4",
  "Language": "en"
}
```

Для правильного запуску серверу системи, у папці з проектом сервера потрібно виконати:

```
export ASPNETCORE_ENVIRONMENT=Development
dotnet run
```

API буде доступне за адресою `https://localhost:7248/api`, де значення порту «7248» буде висвітлено у відкритій консолі після правильного запуску проекту.

Наступним кроком буде налаштування запуску проекту клієнта. У проекті клієнта (`vite.config.ts`) маємо задати проксі для API, щоб браузер без CORS-помилки переадресовував запити:

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    host: true,
    proxy: {
      "/api": {
        target: "https://localhost:7248",
        secure: false,
        changeOrigin: true
      }
    }
  }
});
```

Далі для запуску фронкенду потрібно лише виконати наступні команди в терміналі:

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

```
cd SGS.WebApp
npm install
npm run dev
```

Щоб завершити налаштування проекту, залишилось лише обрати хостинг для AI-моделі: локально (наприклад, за допомогою Ollama), або сторонні хмарні сервіси (Azure OpenAI, OpenAPI, тощо). Для тестування можна налаштувати Semantic Kernel на роботу з локальними моделями через Ollama. Наприклад, у SemanticKernelSettings задати:

```
"SemanticKernelSettings": {
  "Endpoint": "http://localhost:11434",
  "DeploymentName": "qwen2.5:1.5b",
  "ApiKey": "",
  "Language": "en"
}
```

Ми протестували моделі qwen2.5:1.5b (986 MB) та deepseek-r1:1.5b (1.1 GB), але в продакшені зупинилися на Azure OpenAI GPT-4, так як більші потужності пропонують стабільність і якість результатів [18].

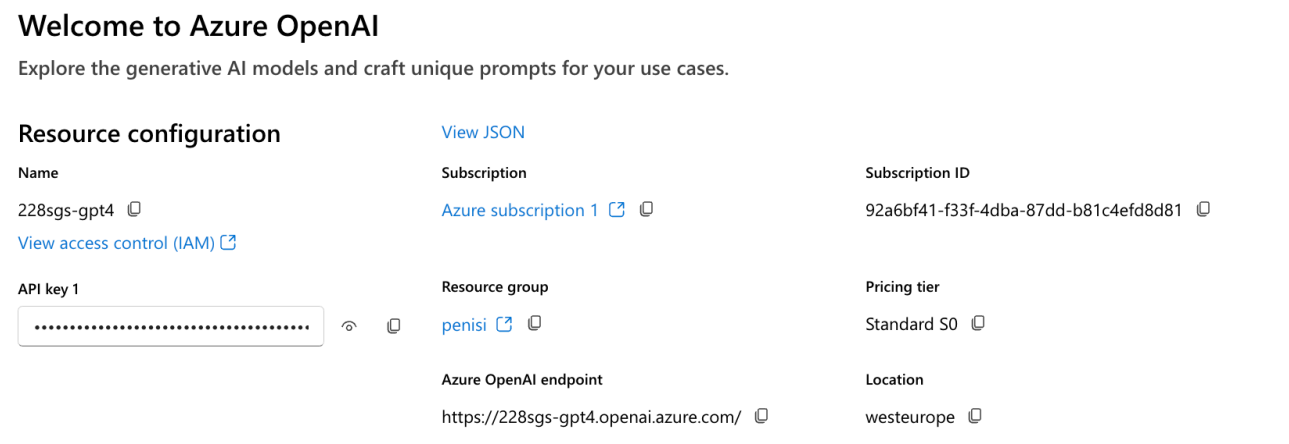


Рис. 3.1 – Сторінка конфігурації AI-моделі на сервісі Azure OpenAI

Після виконання цих кроків сервер і клієнт будуть готові до роботи, а всі сервіси — авторизаційні, файлові, AI-модуль — коректно з’єднуються між собою згідно з N-tier архітектурою.

3.2 Структура веб-інтерфейсу системи автоматизованого аналізу студентських робіт

Інтерфейс реалізовано як односторінковий React-додаток із лівою бічною навігацією, яка адаптується під роль користувача. Після авторизації кожен користувач потрапляє на свою головну панель – для студента вона відображає перелік курсів із мінімалістичними картками, а для викладача – загальний огляд активних груп і курсів [31].

При першому зверненні користувач бачить чистий інтерфейс авторизації – просту форму в центрі екрану з полями для email та пароля й кнопкою (рис. 3.2). Введені дані одразу надсилаються на ендпоінт `/api/auth/login`, а при успішній автентифікації клієнт отримує JWT і переходить на дашборд. Під формою розміщене посилання на реєстрацію, яке веде на окрему сторінку, щоб нові користувачі могли створити обліковий запис.

Рис. 3.2 – Сторінка авторизації

Після входу відкривається головний екран з бічною панеллю навігації (рис. 3.3), де перелічені всі курси користувача. У центральному контенті розміщено календар із підсвіченими датами дедлайнів, а поруч таблиця «Всі завдання» із переліком завдань, їхніми датами й статусами (Завершено, Оцінювання, Строк

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

сплив). Нижче відображаються найближчі дедлайни у вигляді карток. Кнопка Logout у правому верхньому куті дозволяє вийти з системи.

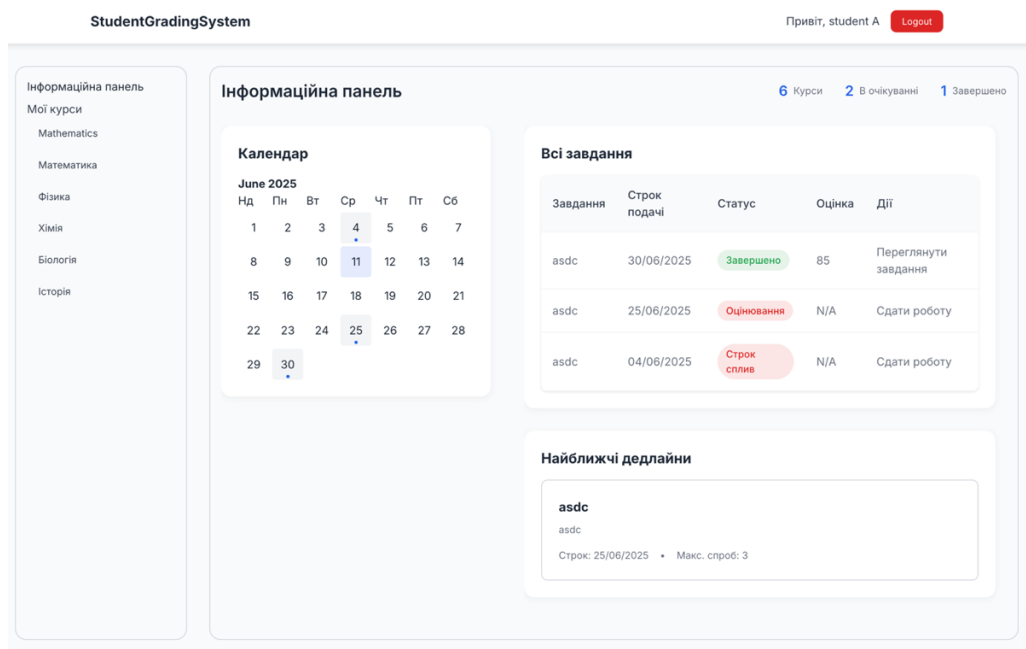


Рис. 3.3 – Інформаційна панель студента

Після вибору курсу студент потрапляє на вкладку «Курс» (рис. 3.4), де міститься його назва та короткий опис. Ліворуч все та ж навігація по курсам, праворуч зверху показано кількість груп та завдань. Списком груп можна перейти безпосередньо до потрібної секції, а картки з назвами завдань дозволяють швидко оцінити, скільки завдань у цьому курсі та їхні дедлайни.

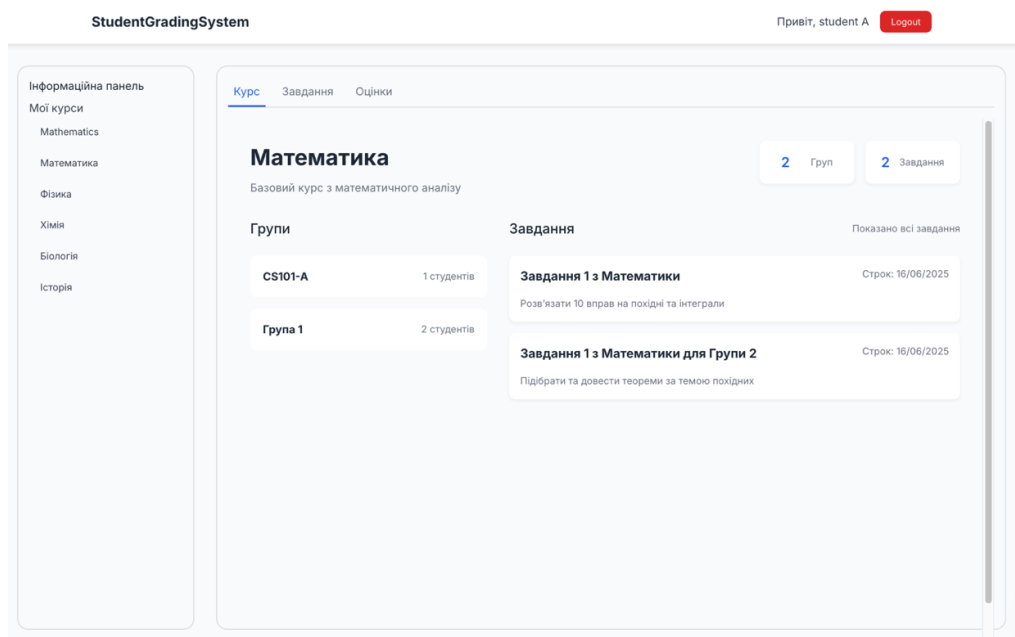


Рис. 3.4 – Сторінка курсу

Вкладка «Завдання» (рис. 3.5) виводить перелік завдань для обраної групи. Кожна картка містить назву завдання, його опис, дату дедлайну та максимальну кількість спроб. За кліком на картку відбувається перехід на деталі конкретного завдання.

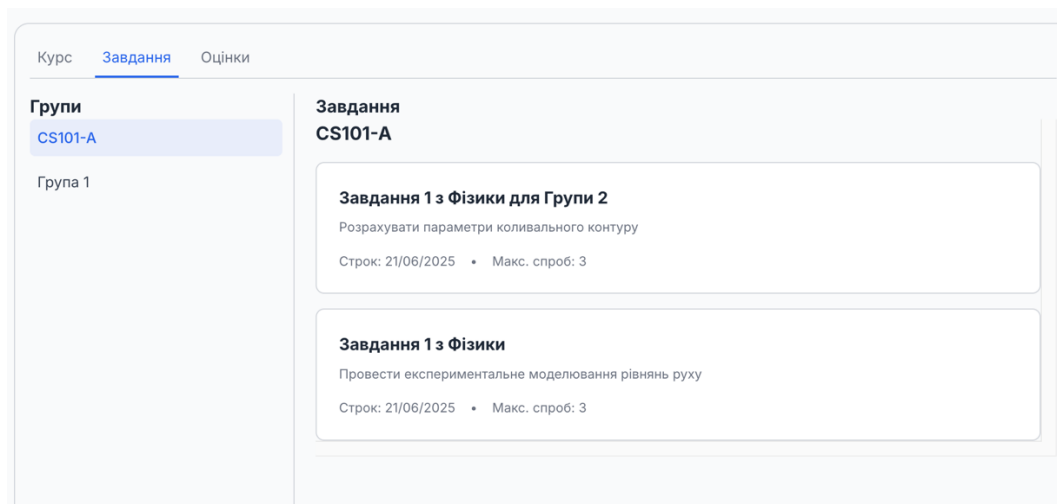


Рис. 3.5 – Перелік завдань з курсу

Вкладка «Оцінки» (рис. 3.6) показує таблицю зі списком студентів обраної групи (в лівій колонці) та їхніми оцінками за кожне завдання у курсі. Поруч виводиться середній бал. Студент може бачити оцінки кожного із студентів, які записані в групи на цей курс і середнє значення.

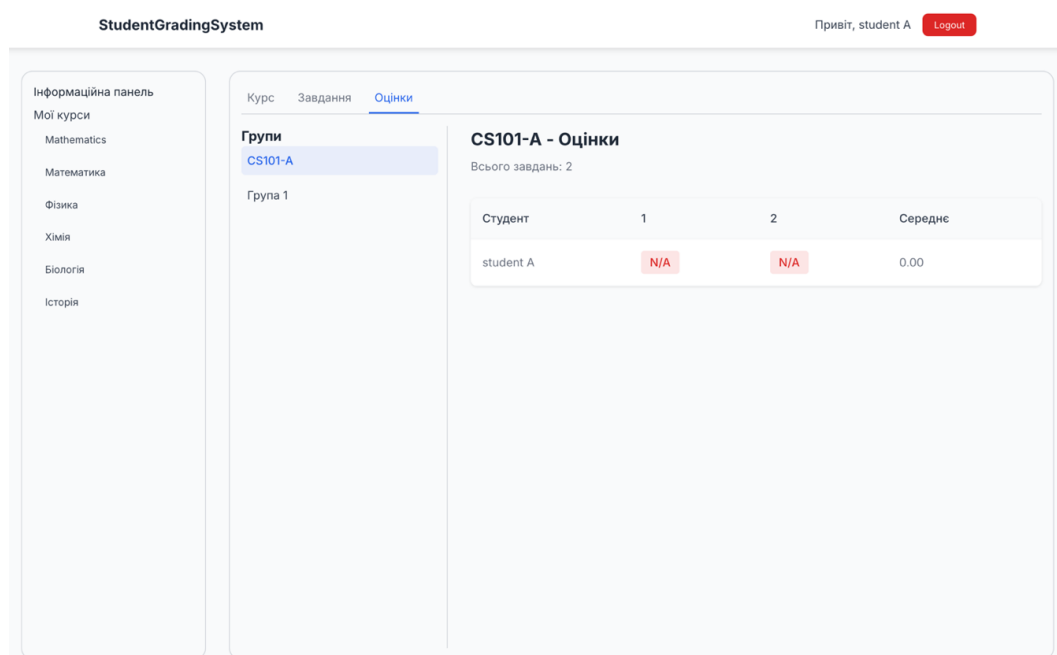


Рис. 3.6 – Таблиця оцінок

На наступному рис. 3.7 зображена панель «Завдання», де студент бачить перелік критеріїв оцінювання: назва критерію, короткий опис і кількість балів. Знизу сторінки на виділеному полі «Здати роботу» є форма, яка дозволяє завантажити PDF-файл роботи через drag&drop чи стандартний файл-інпут. Після натискання відповідної кнопки відбувається POST /api/document/upload, а студент бачить індикатор завантаження.

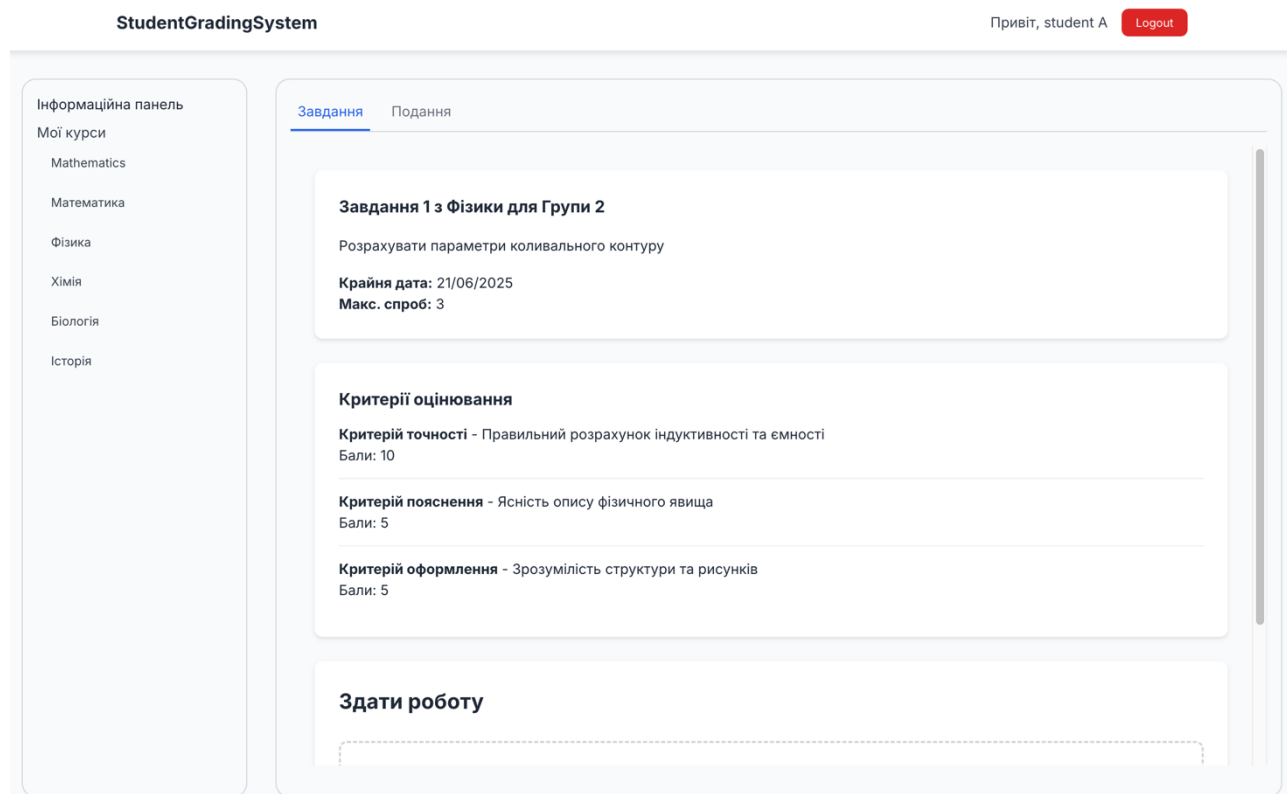


Рис. 3.7 – Сторінка із завданням

Вкладка «Подання», рис. 3.8, на цій сторінці демонструє попередній перегляд PDF-файлу за допомогою React PDF Viewer [22]. Якщо аналіз уже виконано, нижче з'явиться резюме та індикатори критеріїв; якщо ні – повідомлення, що документ ще не оброблено.

Наступним кроком розглянемо вигляд інтерфейсу з боку викладача. Після авторизації викладач бачить головний екран із двома панелями (рис. 3.9). Ліва бокова панель відображає перелік курсів, за якими він закріплений, а права частина – картки курсів із кількістю груп і завдань, а також блок «Нещодавні завдання» для швидкого доступу до останніх завдань. У правому верхньому куті показано загальну статистику (кількість курсів, груп, активних завдань).

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

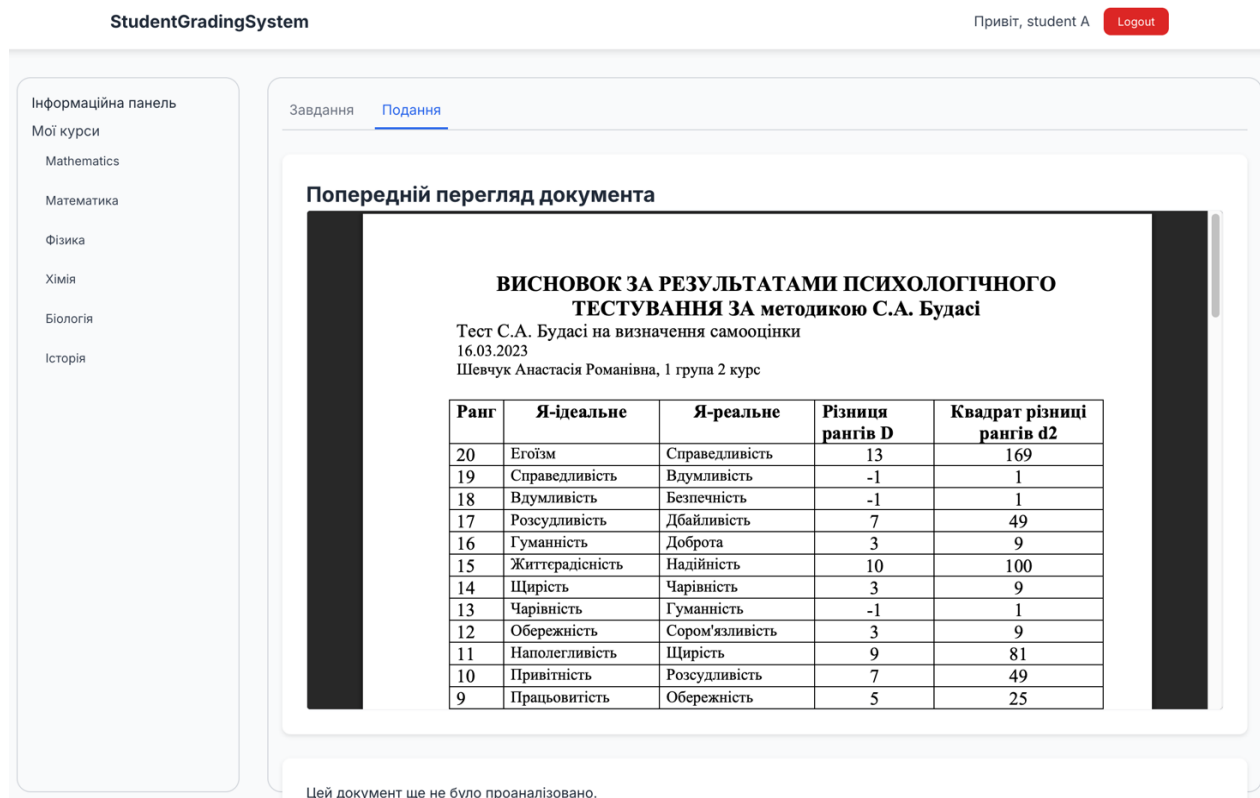


Рис. 3.8 – Перегляд завантаженої роботи

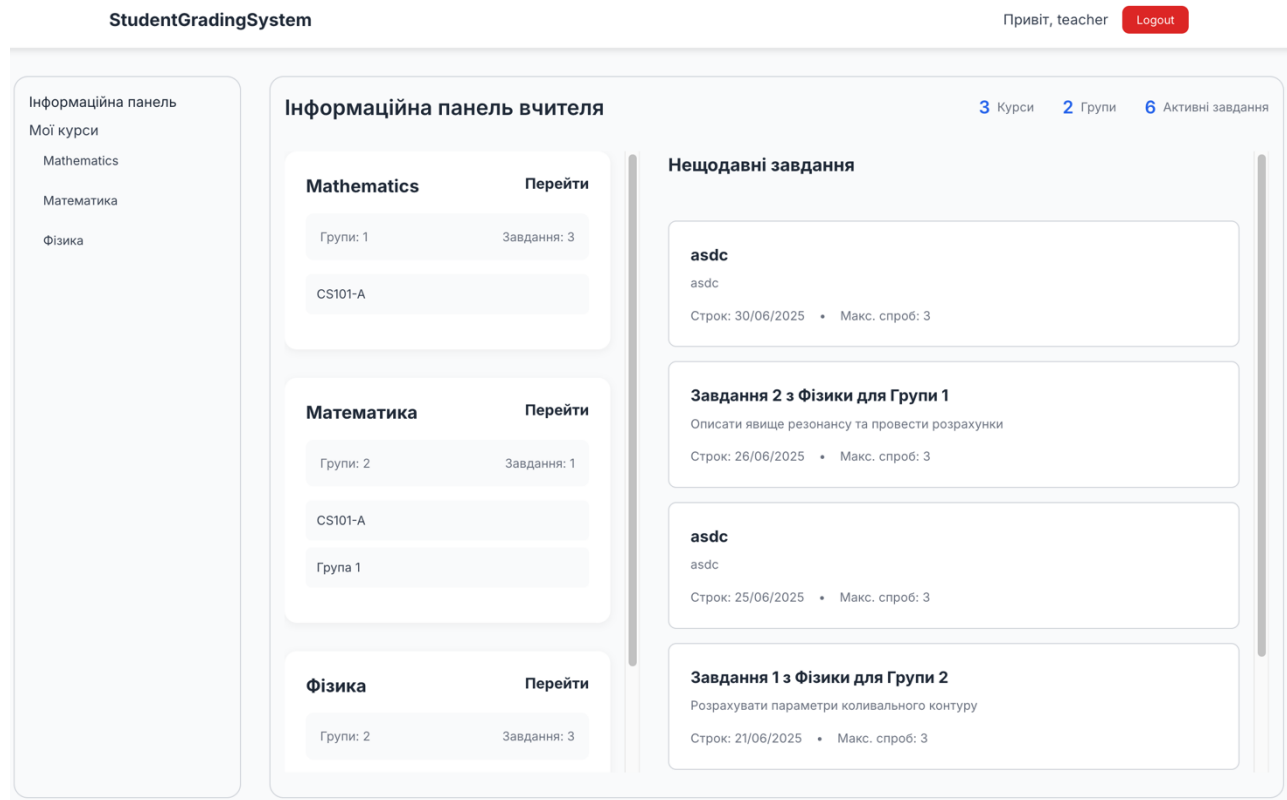


Рис. 3.9 – Інформаційна панель викладача

Перейшовши до курсу (рис. 3.10), викладач потрапляє на вкладку «Курс», де розміщено назву та опис курсу. Біля заголовка виведено загальну кількість груп і завдань. Нижче картками перераховані групи з числом студентів, а поруч – список завдань з їхніми дедлайнами.

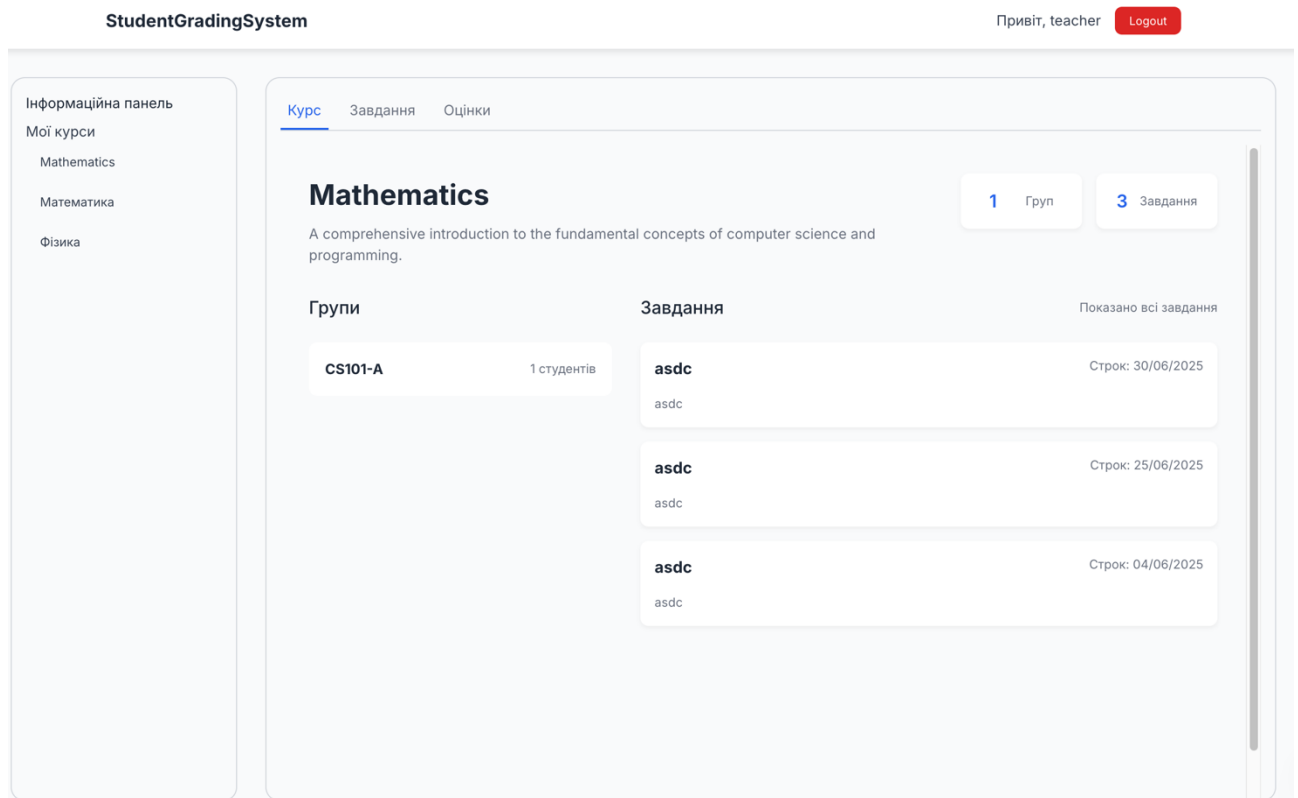


Рис. 3.10 – Сторінка курсу

Вкладка «Завдання» (рис. 3.11) дозволяє викладачу вибрати групу в лівій колонці: якщо група не обрана, відображається анімоване повідомлення-запрошення. Після вибору з'являється перелік завдань для цієї групи у вигляді вертикальних карток із назвою, описом, датою дедлайну й максимальною кількістю спроб. Кнопка «Створити завдання» у нижньому куті відкриває форму створення нового завдання.

На рис. 3.12 зображена форма, що дозволяє створення завдання. Викладач спочатку має задати параметри нового завдання: назву, опис, дату дедлайну, ліміт спроб і групу. Динамічний блок для критеріїв оцінювання дозволяє додавати будь-яку кількість критеріїв: вводити їхні назву, опис, вагу та максимальні бали. Після заповнення натискання «Створити завдання» надсилає дані на `/api/assignments/create` і повертає до списку.

StudentGradingSystem
Привіт, teacher
Logout

Інформаційна панель
Мої курси
Mathematics
Математика
Фізика

Курс
Завдання
Оцінки

Групи
CS101-A

Завдання

Виберіть групу, щоб переглянути її завдання

Створити завдання

Рис. 3.11 – Завдання курсу

Курс
Завдання
Оцінки

Назва завдання

Опис завдання

Строк подання

dd.mm.yyyy

Макс. спроб

Макс. спроби

Група

Виберіть варіант

Grading Criteria

Назва критерію

Опис критерію

0

0

X

Додати критерій

Створити завдання

Рис. 3.12 – Форма для створення нових завдань

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

Вкладка «Оцінки», рис. 3.13, відображає таблицю для вибраної групи: у лівій частині список груп, а в правій – рядки з іменами студентів і їхніми оцінками за кожне завдання курсу. Якщо робіт ще немає або оцінки не виставлені, вони позначені як «В очікуванні» або «N/A».

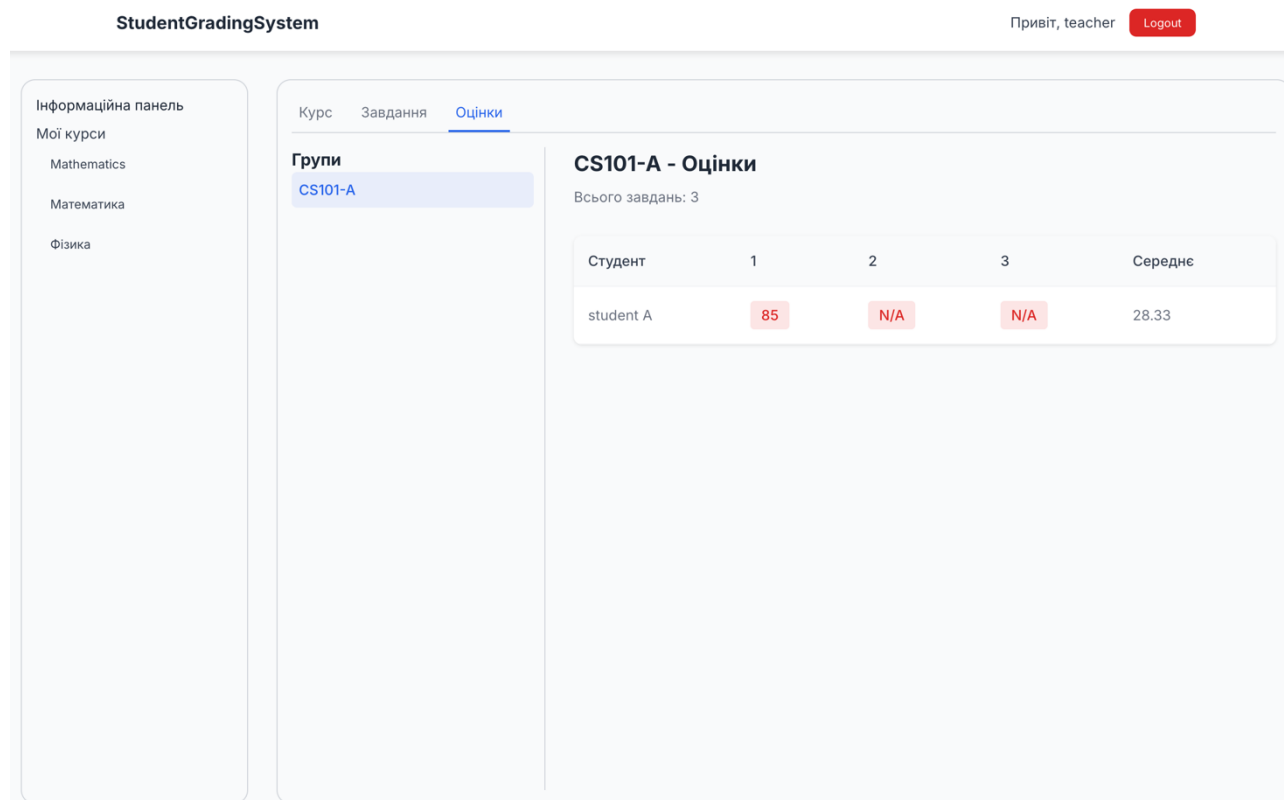


Рис. 3.13 – Оцінки студентів по групам

На вкладці «Завдання», рис. 3.14, викладач бачить поля з описом завдання, датою дедлайну й макс. спробами, а також перелік критеріїв із назвами та максимальною кількістю балів. Окрім того, викладач може змінити опис завдання, вирівняти дедлайн, чи навіть редагувати кожен із окремих критеріїв оцінювання натиснувши відповідну кнопку та перейшовши до форми редагування (рис. 3.15).

Вкладка «Студенти», зображена на рис. 3.17, показує список студентів обраної групи у вигляді вертикального списку. При виборі імені напроти відображаються кнопки «Переглянути» і «Завантажити», а також кнопка «Розпочати аналіз», яка тригерить виклик `/api/grading/analyze/{docId}`.

StudentGradingSystem

Привіт, teacher Logout

Інформаційна панель

Мої курси

Mathematics

Математика

Фізика

Завдання

Студенти

Оцінки

Завдання 1 з Математики

Розв'язати 10 вправ на похідні та інтеграли

Крайня дата: 16/06/2025

Макс. спроб: 3

Критерії оцінювання

Критерій коректності - Перевірка правильності розв'язків вправ

Бали: 10

Критерій оформлення - Відповідність оформлення академічним стандартам

Бали: 5

Критерій вчасності - Здача роботи до встановленого дедлайну

Бали: 5

Редагувати завдання

Рис. 3.14 – Огляд завдання

Завдання

Студенти

Оцінки

Назва завдання

Завдання 1 з Математики

Опис завдання

Розв'язати 10 вправ на похідні та інтеграли

Строк подання

15.06.2025

Макс. спроб

3

Група

Група 1

Критерії оцінювання

Критерій коректності	Перевірка правильності	10	0,5	X
Критерій оформлення	Відповідність оформле	5	0,3	X
Критерій вчасності	Здача роботи до встан	5	0,2	X

Додати критерій

Оновити завдання

Рис. 3.15 – Форма редагування завдання

Після запуску аналізу внизу тієї ж вкладки з'являється блок «Результати аналізу» (рис. 3.17), що містить резюме, індикатори критеріїв і поле для введення фінальної оцінки. Після редагування оцінки, за потреби, та натискання «Виставити» відправляє PUT-запит до /api/grading/edit.

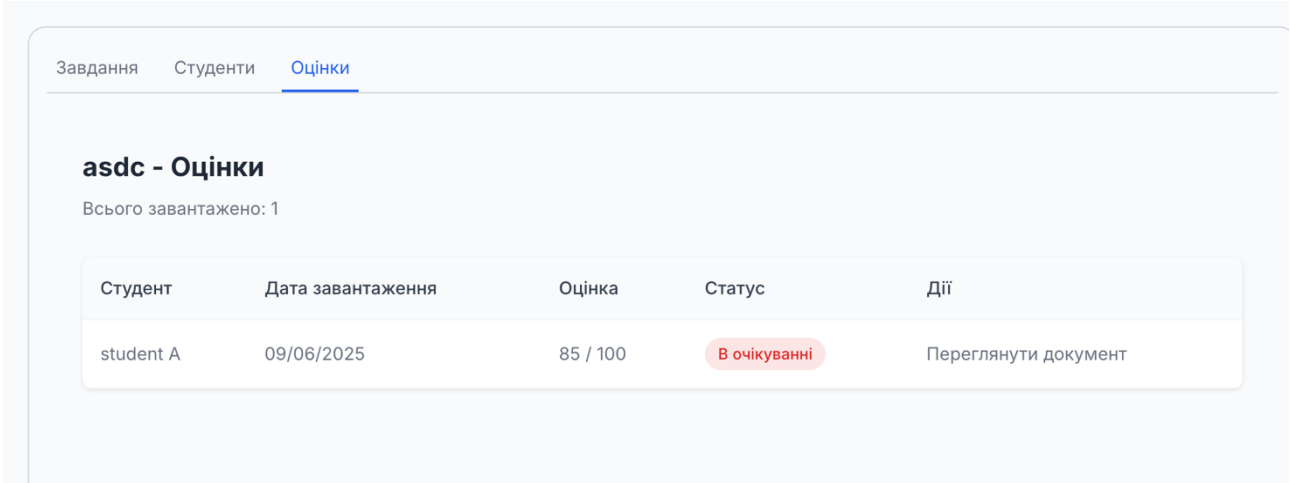


Рис. 3.16 – Перегляд завантажених робіт

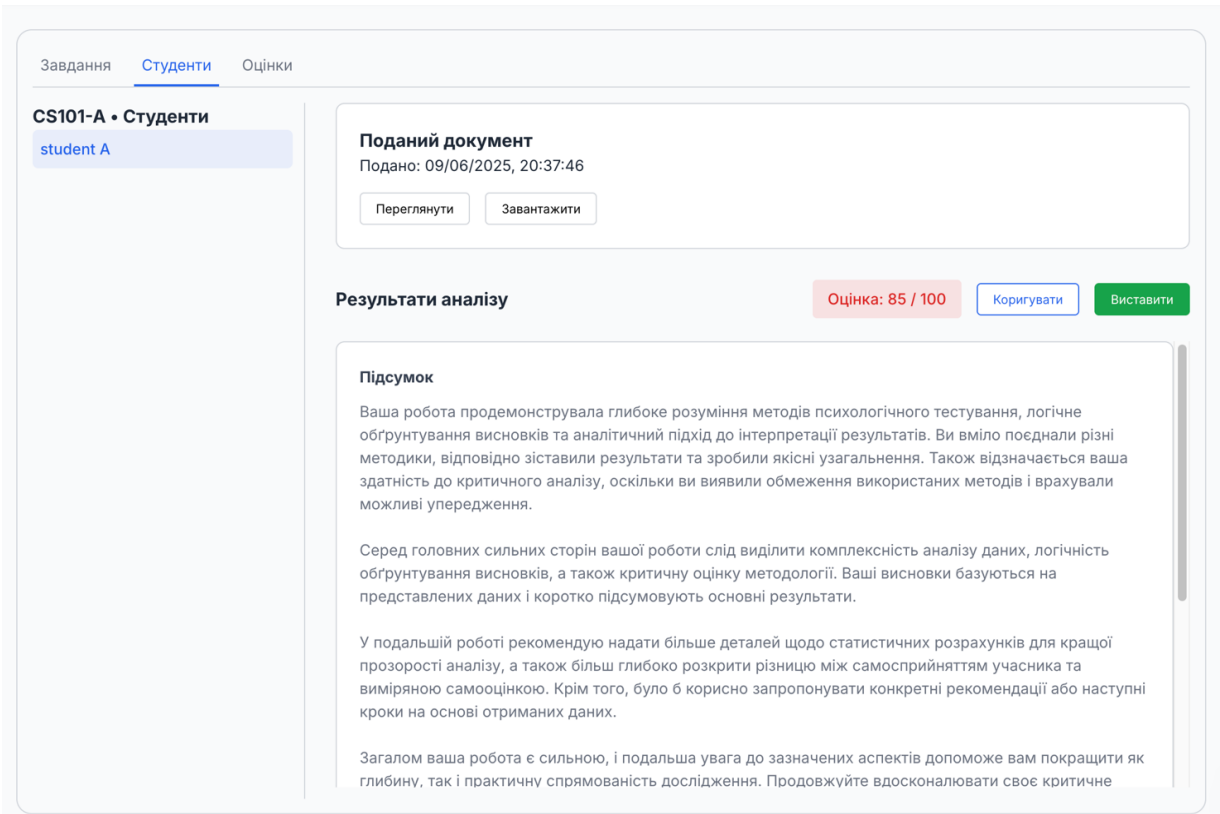


Рис. 3.17 – Перегляд результатів аналізу

Кожен із наведених вище екранних форм-факторів реалізовано як окремий React-компонент, що отримує дані через React Query та оновлюється автоматично при зміні стану. Ліва навігація підтримує контекстне підсвічування поточної секції, а кожна вкладка використовує React Router для відображення відповідного контенту без перезавантаження сторінки.

3.3 Тестування роботи системи

Для гарантування стабільності та відповідності заявленим вимогам у “Student Grading System” було побудовано багаторівневу стратегію тестування, що охоплює автоматизовані та ручні методи.

Насамперед проведено модульні тести бізнес-сервісів, зокрема компонентів, які реалізують обробку документів (DocumentService) та взаємодію з SemanticAnalysis. Використовуючи xUnit і Moq [14], перевірялися окремі методи наповнення репозиторію, нормалізації тексту та виклику LLM-шаблонів. Усі ці тести успішно пройшли, що забезпечило впевненість у працездатності основних алгоритмів без залучення зовнішніх залежностей.

Далі модульні тести Semantic Kernel модуля перевіряли правильність зчитування YAML-шаблонів, формування підказок через IPromptBuilder та десеріалізацію JSON-відповідей у DTO. Для цього застосовувалися як unit-тести з мокованим ядром, так і інтеграційні тести [15], що піднімали тестове API, або реальне Semantic Kernel із тестовими налаштуваннями (наприклад, локальні Ollama-моделі).

На рівні контролерів було виконано інтеграційні тести з WebApplicationFactory, які піднімали мінімальний екземпляр API з in-memory базою даних та перевіряли HTTP-контракти всіх основних endpoint-ів (/api/auth/login, /api/document/upload, /api/grading/analyze/{id} тощо). Ці тести підтвердили, що запит – сервіс – репозиторій – БД – відповідь працює без помилок, а всі сервіси коректно ініціалізуються через DI.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

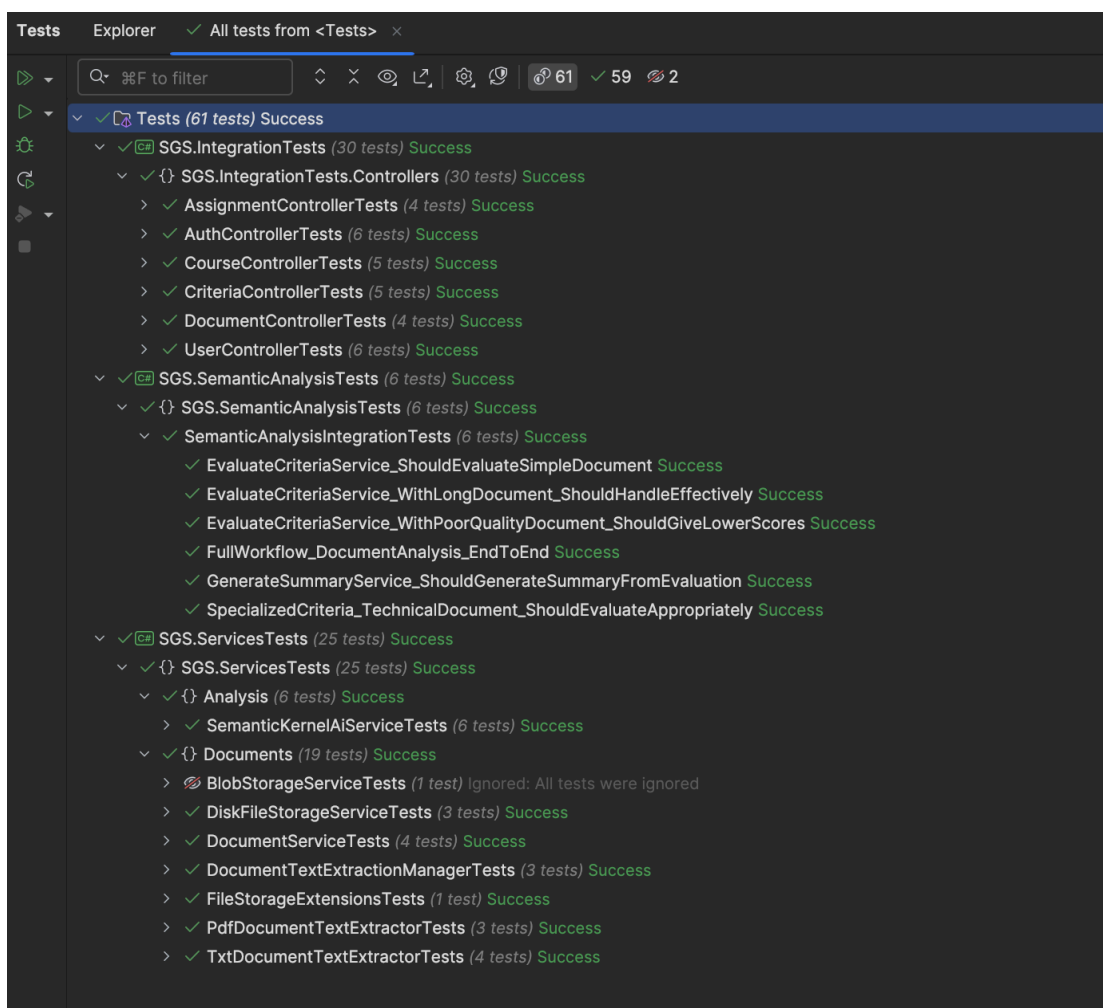


Рис. 3.18 – Тестування системи

Ручне тестування веб-інтерфейсу проводилося на основі end-to-end сценаріїв: від логіну студента до успішного завантаження роботи й перегляду результатів аналізу, а також створення нового завдання викладачем та коригування оцінки. Перевірялися обробка помилок на клієнті та оновлення кешу React Query.

Для порівняння якості семантичного аналізу під різними моделями було зібрано вибірку зі 100 документів та проаналізовано їх з використанням локальних Ollama-моделей (qwen2.5:1.5b, deepseek-r1:1.5b) і хмарного GPT-4 від Azure. Метрики – час виконання та підтримка української мови – підкреслили перевагу GPT-4 у точності викладу та стабільності часу відповіді. Всі моделі показали себе добре у точності викладу результату на англійській мові, проте локально розгорнуті моделі не змогли показати влучний результат аналізуючи текст на українській. Результати наведено в табл. 3.1. [9]

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Таблиця 3.1

Аналіз працездатності AI-моделей

Модель	Середній час аналізу (с)	Підтримка мови	Примітки
deepseek-r1:1.5b	>20	ні	локальний запуск
qwen2.5:1.5b	>35	ні	локальний запуск
Azure OpenAI GPT-4	~6-10	так	хмарний сервіс Azure

З отриманих даних видно, що хмарна модель GPT-4 від Azure забезпечує найшвидший час відповіді (6–10 с) при обсязі документа близько однієї сторінки А4, проте локальні моделі поступаються своїми позиціями в рази. Що стосується якості резюме, GPT-4 продемонструвала найкращі показники когерентності та насиченості фактами, генеруючи стислий виклад, який водночас містить усі ключові тези.

Щодо аналізу тексту використовуючи різні мови, GPT-4 знов випереджає інші аналізуючи текст однаково на будь-якій мові. Насправді такі результати досить очікувані від Ollama моделей Deepseek-r1:1.5b і qwen2.5:1.5b [19] – обидві ці локальні моделі містять близько 1.5 мільярда параметрів і зазвичай обробляють контекст обсягом до 2 048 токенів за раз. Обмежена кількість параметрів і відносно малий контекст-вікно роблять їх порівняно повільними на CPU/GPU середнього класу та змушують розбивати документ на фрагменти. Натомість Azure OpenAI GPT-4, за експертними оцінками, має щонайменше 175 мільярдів параметрів (хоч точна цифра офіційно не розголошується) і підтримує контекст довжиною 8 192 токенів (або до 32 768 у розширеному режимі). Завдяки великій кількості параметрів, оптимізованим кластерам та розширеному контекстному вікну GPT-4 демонструє найкоротший час відповіді та кращу когерентність резюме, оскільки може «бачити» більшу частину документа й одночасно обробляти складніші семантичні зв'язки.

На підставі цих результатів для продакшену обрано Azure OpenAI GPT-4, оскільки вона поєднує високу швидкодію з відмінною якістю семантичного аналізу при роботі з типовими документами обсягом однієї сторінки.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Завдяки такому комплексному підходу система пройшла через всі рівні перевірок: юніт-тести, інтеграційні тести та e2e-сценарії показали високу стабільність, а оцінювання AI-модуля підтвердило якість семантичного аналізу. Платформа готова до промислового запуску та відповідає встановленим технічним і бізнес-вимогам.

Висновки до третього розділу

У цьому розділі було послідовно показано, як розгорнути платформу. Було розглянуто створення бази даних PostgreSQL, запуск ASP.NET Core API та Vite-клієнта з проксі. Відповідні конфігураційні файли appsettings.json і vite.config.ts дозволяють швидко запустити сервер і фронтенд у локальному середовищі або продакшені.

Далі було описано структуру веб-інтерфейсу для студентів і викладачів. Завдяки React Router, React Query і Styled Components інтерфейс вийшов зрозумілим і послідовним для обох ролей.

Останнім етапом було розглянуто тестування: модульні тести бізнес-сервісів і SemanticAnalysis, інтеграційні тести API-контролерів з in-memory БД і порівняння продуктивності AI-моделей (Deepseek, Qwen, Azure GPT-4). Усі тести пройшли успішно, а метрики часу відповіді та якості резюме підтвердили вибір GPT-4 як оптимального рішення.

Таким чином, встановлення, конфігурація, UI-дизайн і тестування платформи відповідають сучасним вимогам до безпеки, продуктивності та зручності, що робить систему готовою до реального використання в освітньому процесі.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було спроектовано та реалізовано комплексну систему автоматизованого аналізу студентських робіт, що поєднує зручний інтерфейс із потужним AI-модулем і комплексною архітектурою сервера. Від первинного аналізу ринку та існуючих рішень до фінального тестування платформи увага була сконцентрована на тому, щоб зробити продукт одночасно простим для користувачів і технічно стійким.

У першому розділі ми дослідили наявні сервіси для перевірки плагіату, граматики та автоматичного оцінювання, визначили їхні сильні й слабкі сторони та сформулювали бізнес- і користувацькі вимоги. Було обрано стек технологій, які найкраще відповідали завданню проєкту.

Другий розділ присвячений архітектурному проєктуванню. Тут описано багаторівневу N-tier архітектуру з Presentation Layer, BLL-сервісами, окремим Semantic Analysis модулем і DAL із патернами Repository/UoW. Розроблено структуру бази даних, структуру API-ендпоінтів і ключові UI-компоненти для студентів і викладачів, що гарантує інтуїтивну навігацію від логіну до коригування оцінок і перегляду результатів.

У третьому розділі викладено порядок розгортання системи. Було ознайомлено із міграціями Entity Framework і різними налаштуваннями системи щоб без проблем запустити сервер та клієнт через Vite. Розглянуто інтеграцію локальних Ollama-моделей і хмарного GPT-4, описано юніт-, інтеграційні та e2e-тести, а також порівняння продуктивності та якості семантичного аналізу різними LLM. Усі тести пройшли успішно, підтвердивши стабільність і безпеку платформи під реальним навантаженням.

Підсумовуючи, було реалізовано єдину робочу екосистему, здатну значно полегшити роботу викладачів і надати студентам оперативний аналітичний зворотний зв'язок. Система готова до впровадження в університетах та освітніх платформах, а її модульна архітектура відкриває шлях до подальших удосконалень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Turnitin [Електронний ресурс] – Режим доступу до ресурсу: <https://www.turnitin.com/>.
2. Grammarly [Електронний ресурс] – Режим доступу до ресурсу: <https://app.grammarly.com/>.
3. Coursera [Електронний ресурс] – Режим доступу до ресурсу: <https://www.coursera.org/>.
4. Component Diagram Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.lucidchart.com/pages/uml-component-diagram>.
5. Karcic F. A Developer's Guide to CQRS Using .NET Core and MediatR [Електронний ресурс] / Faris Karcic. – 2020. – Режим доступу до ресурсу: <https://www.rubicon-world.com/blog/2020/06/a-developers-guide-to-cqrs-using-net-core-and-mediatr/>.
6. Understanding NLP Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@bipasha_bg/understanding-nlp-architecture-3b0a50c8192f.
7. Про патерни проектування [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/paterny-proektuvannia/>.
8. Dependency injection та Singleton. Все, що треба знати, і ще трошки [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@ivanfomenko/dependency-injection-ta-singletone-vse-sho-treba-znati-i-she-troshki-2e738373eb94>.
9. Large Language Models [Електронний ресурс] – Режим доступу до ресурсу: <https://www.udemy.com/course/large-language-models-llms-fundamentals/>.
10. N-Layer Architecture In A Multi-Project Solution [Електронний ресурс] – Режим доступу до ресурсу: <https://makeitnew.io/n-layer-architecture-in-a-multi-project-solution-d72604f0e030>.
11. Unit of Work [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unit_Of_Work.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

12. Repository pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/net-core/repository-pattern-implementation-in-asp-net-core-21e01c6664d7>.

13. Result pattern [Електронний ресурс] – Режим доступу до ресурсу: https://www.reddit.com/r/dotnet/comments/1fxblvt/results_pattern_in_net/.

14. Unit testing C# in .NET using dotnet and xUnit [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-xunit>.

15. Integration tests in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-9.0&pivots=xunit>.

16. Microsoft Semantic Kernel [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/semantic-kernel/overview/>.

17. Microsoft Azure AI Fundamentals: AI Overview [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/training/paths/get-started-with-artificial-intelligence-on-azure/>.

18. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: <https://portal.azure.com/#home>.

19. Ollama – Get Started [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/ollama/ollama>.

20. Microsoft Semantic Kernel Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/MicrosoftDocs/semantic-kernel-docs?utm_source=chatgpt.com.

21. Осипчук А.О., Морозов А.В. Система автоматизованого аналізу студентських робіт. Тези Всеукраїнської науково-практичної онлайн-конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12–17 травня 2025 року. С.69-70.

22. React PDF Viewer[Електронний ресурс] – Режим доступу до ресурсу: <https://react-pdf-viewer.dev/>.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

23. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.

24. Вступ до REST API – RESTful вебсервіси [Електронний ресурс] – Режим доступу до ресурсу: <https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi>.

25. JWT Authentication in ASP.NET Core Web API [Електронний ресурс] – Режим доступу до ресурсу: <https://code-maze.com/authentication-aspnetcore-jwt-1/>.

26. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/>.

27. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/>.

28. The TypeScript Handbook [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/handbook/intro.html>.

29. Relationships in SQL – Complete Guide With Examples [Електронний ресурс] – Режим доступу до ресурсу: <https://www.debart.com/dbforge/sql/studio/sql-relationships.html>.

30. Where to store JWT refresh tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://security.stackexchange.com/questions/271157/where-to-store-jwt-refresh-tokens>.

31. React-Vite: Getting Started[Електронний ресурс] – Режим доступу до ресурсу: <https://vite.dev/guide/>.

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

ДОДАТКИ

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": ""
  },
  "JwtSettings": {
    "Secret": "",
    "Issuer": "",
    "Audience": "",
    "ExpirationMinutes": 0,
    "RefreshTokenExpirationDays": 0
  },
  "Storage": {
    "Provider": "Disk",
    "BasePath": "SGS.Infrastructure/FileStorage"
  },
  "SemanticKernelSettings": {
    "Endpoint": "",
    "ApiKey": "",
    "DeploymentName": "",
    "Language": "en" // for different languages to semantic module
  }
}
```

vite.config.ts

```
import {defineConfig} from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
  optimizeDeps: {
    include: ['react-pdf', 'pdfjs-dist/build/pdf.worker.entry'],
  },
  server: {
    port: 5173,
    host: true,
    proxy: {
      '/api': {
        target: 'https://localhost:7248',
        secure: false,
        changeOrigin: true
      }
    }
  }
})
```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

api.ts

```

import axios, {AxiosError} from "axios";

interface RetryConfig {
  retry: number;
  retryDelay: number;
  retryCount: number;
}

const api = axios.create({
  baseURL: "/api",
  headers: {
    "Content-Type": "application/json",
  },
  withCredentials: true,
});

api.defaults.retry = 1;
api.defaults.retryDelay = 1000;
api.defaults.retryCount = 0;

api.interceptors.response.use(
  (response) => response,
  async (error: AxiosError) => {
    const config = error.config as RetryConfig & any;

    if (!config || !config.retry) {
      return Promise.reject(error);
    }

    config.retryCount = config.retryCount || 0;

    if (config.retryCount >= config.retry) {
      console.error('API Error (Max retries reached):', {
        url: config.url,
        method: config.method,
        status: error.response?.status,
        data: error.response?.data,
        retryCount: config.retryCount
      });
      return Promise.reject(error);
    }

    config.retryCount += 1;
    const delay = config.retryDelay * Math.pow(2, config.retryCount - 1);
    console.log(`Retrying request (${config.retryCount}/${config.retry}) after ${delay}ms`);
    await new Promise(resolve => setTimeout(resolve, delay));

    return api(config);
  }
);

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

    }
  );

  api.interceptors.request.use(
    (config) => {
      const token = localStorage.getItem("accessToken");
      if (token && config.headers) {
        config.headers.Authorization = `Bearer ${token}`;
      }
      console.log('Request Config:', {
        url: config.url,
        method: config.method,
        headers: config.headers,
        data: config.data,
        retry: config.retry,
        retryCount: config.retryCount
      });
      return config;
    },
    (error) => Promise.reject(error)
  );

```

export default api;

useAuth.ts

```

import { LoginDto, RegisterUserCommand } from "../types/AuthTypes.ts";
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
import * as authApi from "../api/apiEndpoints/AuthApi.ts";
import * as userApi from "../api/apiEndpoints/UserApi.ts";
import { RefreshTokenRequestDto } from "../types/TokenTypes.ts";

```

```

export const useAuth = () => {
  const queryClient = useQueryClient();

  const loginMutation = useMutation({
    mutationFn: async (data: LoginDto) => {
      try {
        return await authApi.login(data);
      } catch (error) {
        console.error('Login Mutation Error:', error);
        throw error;
      }
    },
    onSuccess: (data) => {
      localStorage.setItem("accessToken", data.token);
      localStorage.setItem("refreshToken", data.refreshToken);
      queryClient.setQueryData(["currentUser"], data.user);
    },
    onError: (error) => {
      console.error('Login Mutation Error Handler:', error);
    }
  });

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```

});

const registerMutation = useMutation({
  mutationFn: (data: RegisterUserCommand) => authApi.registerUser(data)
});

const logoutMutation = useMutation({
  mutationFn: (data: RefreshTokenRequestDto) => authApi.logout(data),
  onSuccess: () => {
    localStorage.removeItem("accessToken");
    localStorage.removeItem("refreshToken");

    queryClient.invalidateQueries({queryKey: ["currentUser"]});
  }
});

const refreshTokenMutation = useMutation({
  mutationFn: (data: RefreshTokenRequestDto) => authApi.refreshToken(data),
  onSuccess: (data) => {
    localStorage.setItem("accessToken", data.token);
    localStorage.setItem("refreshToken", data.refreshToken);

    queryClient.setQueryData(["currentUser"], data.user);
  },
});

const currentUserQuery = useQuery({
  queryKey: ["currentUser"],
  queryFn: () => userApi.getCurrentUser(),
  enabled: !!localStorage.getItem("accessToken"),
});

const user = currentUserQuery.data;

return {
  user,
  loginMutation,
  logoutMutation,
  registerMutation,
  refreshTokenMutation,
  currentUserQuery
};
};

```

Program.cs

```

using Microsoft.EntityFrameworkCore;
using SGS.API.Extensions.Configurations;
using SGS.API.Middleware;
using SGS.Application;
using SGS.Infrastructure;
using SGS.Infrastructure.Data;
using SGS.SemanticAnalysis;

var builder = WebApplication.CreateBuilder(args);
builder.Logging.ClearProviders();
builder.Logging.AddConsole();
builder.Logging.AddDebug();

if (!builder.Environment.IsEnvironment("Testing"))
{
    builder.Services.AddAuthServices(builder.Configuration);
}

builder.Services.AddInfrastructure(builder.Configuration, builder.Environment);
builder.Services.AddApplication(builder.Configuration);
builder.Services.AddSemanticKernel(builder.Configuration);
builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerWithAuth();

var allowedOrigins = new[] { "http://localhost:5173" };
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(b =>
    {
        b.WithOrigins(allowedOrigins)
        .AllowAnyMethod()
        .AllowAnyHeader()
        .AllowCredentials();
    });
});

var app = builder.Build();

app.UseCors();
app.UseSwagger();
if (app.Environment.IsDevelopment())
{
    app.UseSwaggerUI(options => { options.SwaggerEndpoint("/swagger/v1/swagger.json", "SGS
API v1"); });
}

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

```
app.UseMiddleware<ExceptionHandlerMiddleware>();
app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
```

```
app.Run();
```

```
public abstract partial class Program;
```

AuthController.cs

```
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SGS.API.Controllers.Base;
using SGS.API.DTOs;
using SGS.Application.DTOs;
using SGS.Application.Interfaces;
using SGS.Domain.Entities;
```

```
namespace SGS.API.Controllers;
```

```
[ApiController]
```

```
[Route("api/auth")]
```

```
public class AuthController(
```

```
    IAuthService authService,
```

```
    ITokenService tokenService,
```

```
    IRefreshTokenService refreshTokenService,
```

```
    IUserService userService,
```

```
    IMapper mapper)
```

```
: BaseApiController
```

```
{
```

```
    [AllowAnonymous]
```

```
    [HttpPost("register")]
```

```
    [ProducesResponseType(StatusCodes.Status201Created)]
```

```
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
```

```
    public async Task<IActionResult> Register([FromBody] RegisterUserCommand command)
```

```
    {
```

```
        if (!ModelState.IsValid)
```

```
            return BadRequest(ModelState);
```

```
        var result = await authService.RegisterUserAsync(command);
```

```
        return HandleCreatedResult(result, nameof(Login), new { email = command.Email });
```

```
    }
```

```
    [AllowAnonymous]
```

```
    [HttpPost("login")]
```

```
    [ProducesResponseType(StatusCodes.Status200OK)]
```

```
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
```

```
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

```

public async Task<IActionResult> Login([FromBody] LoginDto dto)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var result = await authService.LoginAsync(dto.Email, dto.Password);

    if (!result.IsSuccess)
        return Unauthorized(new ProblemDetails { Detail = result.Error });

    return Ok(result.Data);
}

[AllowAnonymous]
[HttpPost("logout")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public async Task<IActionResult> Logout([FromBody] RefreshTokenRequestDto request)
{
    if (string.IsNullOrEmpty(request.RefreshToken))
        return BadRequest(new ProblemDetails { Detail = "Refresh token is required." });

    var tokenResult = await refreshTokenService.GetRefreshTokenAsync(request.RefreshToken);
    if (!tokenResult.IsSuccess)
        return BadRequest(new ProblemDetails { Detail = tokenResult.Error });

    var existingRefreshToken = tokenResult.Data;
    if (existingRefreshToken is not { IsActive: true })
        return Unauthorized(new ProblemDetails { Detail = "Invalid or expired refresh token." });

    var revokeResult = await refreshTokenService.RevokeRefreshTokenAsync(request.RefreshToken);
    if (!revokeResult.IsSuccess)
        return BadRequest(new ProblemDetails { Detail = revokeResult.Error });

    return Ok(new { message = "Logged out successfully" });
}

[Authorize]
[HttpPost("refresh")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public async Task<IActionResult> RefreshToken([FromBody] RefreshTokenRequestDto request)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    if (string.IsNullOrEmpty(request.RefreshToken))
        return BadRequest(new ProblemDetails { Detail = "Refresh token is required." });
}

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

```

var tokenResult = await refreshTokenService.GetRefreshTokenAsync(request.RefreshToken);
if (!tokenResult.IsSuccess)
    return BadRequest(new ProblemDetails { Detail = tokenResult.Error });

var existingRefreshToken = tokenResult.Data;
if (existingRefreshToken is not { IsActive: true })
    return Unauthorized(new ProblemDetails { Detail = "Invalid or expired refresh token." });

var revokeResult = await refreshTokenService.RevokeRefreshTokenAsync(request.RefreshToken);
if (!revokeResult.IsSuccess)
    return BadRequest(new ProblemDetails { Detail = revokeResult.Error });

var userResult = await userService.GetUserByIdAsync(existingRefreshToken.UserId);
if (!userResult.IsSuccess || userResult.Data is null)
    return Unauthorized(new ProblemDetails { Detail = userResult.Error ?? "User not found." });

var user = userResult.Data;
var accessToken = tokenService.GenerateToken(mapper.Map<User>(user));

var newTokenResult = await refreshTokenService.CreateRefreshTokenAsync(user.Id);
if (!newTokenResult.IsSuccess || newTokenResult.Data is null)
    return BadRequest(new ProblemDetails { Detail = newTokenResult.Error });

return Ok(new AuthResponseDto
{
    Token = accessToken,
    RefreshToken = newTokenResult.Data.Token,
    User = user
});
}
}

```

IUserService.cs

```

using SGS.Application.DTOS;
using SGS.Application.Models;
using SGS.Domain.Entities;

namespace SGS.Application.Interfaces;

public interface IUserService
{
    Task<Result<UserDto>> GetUserByEmailAsync(string email, CancellationToken cancellationToken = default);

    Task<Result<IEnumerable<UserDto>>> GetTeachersAsync(CancellationToken cancellationToken = default);
}

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

```

Task<Result<IEnumerable<UserDto>>>> GetStudentsAsync(CancellationTok
cancellationToken = default);

Task<Result<UserDto?>> GetUserByIdAsync(Guid id, CancellationTok
default);

Task<Result<UserDto>> UpdateUserAsync(Guid id, UserDto updateUserDto,
CancellationToken cancellationToken = default);

Task<Result<bool>> DeleteUserAsync(Guid id, CancellationTok
default);
}

```

UserService.cs

```

public class UserService(
    IUnitOfWork unitOfWork,
    ILogger<UserService> logger,
    IMapper mapper)
: BaseService(unitOfWork, logger, mapper), IUserService
{
    public async Task<Result<UserDto>> UpdateUserAsync(Guid id, UserDto updateUserDto,
        CancellationToken cancellationToken = default)
    {
        Logger.LogInformation("Updating user with ID: {UserId}", id);

        try
        {
            var existingUser = await UnitOfWork.Users.GetByIdAsync(id, cancellationToken);
            if (existingUser == null)
            {
                Logger.LogWarning("User not found with ID: {UserId}", id);
                return Result<UserDto>.Failure("User not found");
            }

            Mapper.Map(updateUserDto, existingUser);

            UnitOfWork.Users.Update(existingUser);

            if (existingUser.Role == UserRole.Student && updateUserDto.GroupId.HasValue)
            {
                var currentGroup = await UnitOfWork.StudentsGroups.GetByStudentIdAsync(id,
cancellationToken);
                var newGroup =
                    await UnitOfWork.StudentsGroups.GetByIdAsync(updateUserDto.GroupId.Value,
cancellationToken);

                if (newGroup == null)
                {
                    Logger.LogWarning("Group not found with ID: {GroupId}",
updateUserDto.GroupId.Value);
                    return Result<UserDto>.Failure("Specified group not found");
                }
            }
        }
    }
}

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

    }

    if (currentGroup == null || currentGroup.Id != newGroup.Id)
    {
        await UnitOfWork.StudentsGroups.AddStudentToGroupAsync(newGroup.Id, id,
cancellationTokens);
    }
}

Logger.LogInformation("Successfully updated user with ID: {UserId}", id);
return Result<UserDto>.Success(Mapper.Map<UserDto>(updateUserDto));
}
catch (Exception ex)
{
    Logger.LogError(ex, "Error updating user with ID: {UserId}", id);
    return Result<UserDto>.Failure($"Failed to update user: {ex.Message}");
}
}

```

IUserRepository.cs

```

using SGS.Domain.Entities;
using SGS.Domain.Enums;
using SGS.Infrastructure.Repositories.BaseRepository;

namespace SGS.Infrastructure.Repositories.Interfaces;

public interface IUserRepository : IBaseRepository<User>
{
    Task<User?> GetByEmailAsync(string email, CancellationToken cancellationToken = default);
    Task<IEnumerable<User>> GetByRoleAsync(UserRole role, CancellationToken
cancellationTokens = default);
    Task AddUserAsync(User user, CancellationToken cancellationToken = default);
}

```

UserRepository.cs

```

using Microsoft.EntityFrameworkCore;
using SGS.Domain.Entities;
using SGS.Domain.Enums;
using SGS.Infrastructure.Data;
using SGS.Infrastructure.Repositories.BaseRepository;
using SGS.Infrastructure.Repositories.Interfaces;

namespace SGS.Infrastructure.Repositories;

public class UserRepository(AppDbContext context)
    : BaseRepository<User>(context), IUserRepository
{
    public async Task<User?> GetByEmailAsync(string email, CancellationToken
cancellationTokens = default)
    {

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

```

        return await Context.Users
            .FirstOrDefaultAsync(u => u.Email == email, cancellationToken);
    }

    public async Task<IEnumerable<User>> GetByRoleAsync(UserRole role, CancellationToken
cancellationToken = default)
    {
        return await Context.Users
            .Where(u => u.Role == role)
            .ToListAsync(cancellationToken);
    }

    public async Task AddUserAsync(User user, CancellationToken cancellationToken = default)
    {
        if (string.IsNullOrEmpty(user.Email))
            throw new ArgumentException("User must have an email");

        if (string.IsNullOrEmpty(user.FullName))
            throw new ArgumentException("User must have a full name");

        var existingUser = await GetByEmailAsync(user.Email, cancellationToken);
        if (existingUser != null)
            throw new InvalidOperationException($"Email {user.Email} is already in use");

        await CreateAsync(user, cancellationToken);
        switch (user.Role)
        {
            case UserRole.Student:
                var student = new Student
                {
                    Id = Guid.NewGuid(),
                    UserId = user.Id,
                    User = user
                };
                await Context.Students.AddAsync(student, cancellationToken);
                break;

            case UserRole.Teacher:
                var teacher = new Teacher
                {
                    Id = Guid.NewGuid(),
                    UserId = user.Id,
                    User = user
                };
                await Context.Teachers.AddAsync(teacher, cancellationToken);
                break;

            case UserRole.Admin:
                break;
            default:
                throw new ArgumentOutOfRangeException(nameof(user.Role), user.Role, "Invalid user
role");
        }
    }

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

    }
}
}

```

Result.cs

```
namespace SGS.Application.Models;
```

```

public class Result<T>
{
    public bool IsSuccess { get; }
    public T? Data { get; set; }
    public string? Error { get; }

    protected Result(bool isSuccess, T? data, string? error)
    {
        IsSuccess = isSuccess;
        Data = data;
        Error = error;
    }

    public static Result<T> Success(T data) => new(true, data, null);
    public static Result<T> Failure(string error) => new(false, default, error);
}

```

UnitOfWork.cs

```

using Microsoft.EntityFrameworkCore.Storage;
using SGS.Infrastructure.Data;
using SGS.Infrastructure.Repositories.Interfaces;

```

```
namespace SGS.Infrastructure.Repositories.UnitOfWork;
```

```

public class UnitOfWork(AppDbContext context) : IUnitOfWork
{
    private IDbContextTransaction? _transaction;
    private bool _disposed;

    public IAnalysisResultsRepository AnalysisResults { get; } = new
    AnalysisResultsRepository(context);
    public IAssignmentRepository Assignments { get; } = new AssignmentRepository(context);
    public ICourseRepository Courses { get; } = new CourseRepository(context);
    public IDocumentRepository Documents { get; } = new DocumentRepository(context);
    public IGradeRepository Grades { get; } = new GradeRepository(context);
    public IGradingCriteriaRepository GradingCriteria { get; } = new
    GradingCriteriaRepository(context);
    public IRefreshTokenRepository RefreshTokens { get; } = new
    RefreshTokenRepository(context);
    public IStudentsGroupRepository StudentsGroups { get; } = new
    StudentsGroupRepository(context);
    public IUserRepository Users { get; } = new UserRepository(context);

    public async Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

    {
        return await context.SaveChangesAsync(cancellationToken);
    }

    public async Task BeginTransactionAsync(CancellationToken cancellationToken = default)
    {
        _transaction = await context.Database.BeginTransactionAsync(cancellationToken);
    }

    public async Task CommitTransactionAsync(CancellationToken cancellationToken = default)
    {
        try
        {
            await context.SaveChangesAsync(cancellationToken);
            await _transaction?.CommitAsync(cancellationToken!);
        }
        catch
        {
            await RollbackTransactionAsync(cancellationToken);
            throw;
        }
        finally
        {
            _transaction?.Dispose();
            _transaction = null;
        }
    }

    public async Task RollbackTransactionAsync(CancellationToken cancellationToken = default)
    {
        await _transaction?.RollbackAsync(cancellationToken!);
        _transaction?.Dispose();
        _transaction = null;
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed && disposing)
        {
            context.Dispose();
            _transaction?.Dispose();
        }

        _disposed = true;
    }

```

EvaluateCriterionService.cs

```

using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.PromptTemplates.Handlebars;
using SGS.SemanticAnalysis.Helpers;
using SGS.SemanticAnalysis.Kernels;
using SGS.SemanticAnalysis.Services.Bases;
using SGS.SemanticAnalysis.Structures.PromptRequests;
using SGS.SemanticAnalysis.Structures.PromptResponses;

namespace SGS.SemanticAnalysis.Services;

public class EvaluateCriterionService(SemanticKernelFactory semanticKernelFactory,
IPromptBuilder promptBuilder)
: BasePromptService, IPromptService<EvaluateCriteriaRequest, EvaluateCriteriaResponse>
{
    protected override string GetYamlFileName() => semanticKernelFactory.PromptLanguage ==
"en"
    ? "EvaluateCriterion.prompt.en.yaml"
    : "EvaluateCriterion.prompt.ua.yaml";

    public async Task<EvaluateCriteriaResponse?> ExecuteAsync(EvaluateCriteriaRequest request,
CancellationTokens cancellationTokens = default)
    {
        var kernel = semanticKernelFactory.CreateBasic();
        var kernelFunction = kernel.CreateFunctionFromPromptYaml(
            PromptYaml,
            promptTemplateFactory: new HandlebarsPromptTemplateFactory());

        var input = promptBuilder.BuildCriterionInputs(request);

        var functionResult = await kernelFunction.InvokeAsync(kernel, input, cancellationTokens);
        var result = functionResult.GetValue<string>();

        return DeserializeJsonFromResponse<EvaluateCriteriaResponse?>(result);
    }
}

```

EvaluateCriterion.prompt.yaml

```

name: EvaluateCriteria
description: |
    Evaluates a student document against multiple grading criteria and assigns a score to each.
template_format: handlebars

template: |
    You are an academic evaluator assessing a student submission.

# Student Submission
{{DocumentText}}

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

```

# Grading Criteria
{{#each Criteria}}
## Criterion {{this.Title}} (weight {{this.Weight}})
Description: {{this.Description}}
{{/each}}

# Instructions
For each criterion above:
1. Rate how well the submission meets the criterion on a scale from 0-100
2. Provide brief justification (~2 sentences) with specific references to the submission
3. Include specific strengths and areas for improvement
4. No explanations or other text before or after the JSON object.

# Output Format:
{
  "criteriaResults": [
    {
      "id": "",
      "title": "",
      "score": number,
      "weight": double,
      "justification": "",
      "strengths": [""],
      "improvements": [""],
    },
    ...
  ]
}

```

Output =>

```

input_variables:
- name: DocumentText
  description: The student's document text to evaluate
  is_required: true
- name: Criteria
  description: Array of criteria objects with id, title, description, and weight
  is_required: true

```

SemanticKernelFactory.cs

```

using Microsoft.Extensions.Options;
using Microsoft.SemanticKernel;
using SGS.SemanticAnalysis.Settings;

namespace SGS.SemanticAnalysis.Kernels;

public class SemanticKernelFactory(IOptions<SemanticKernelSettings> settings, IServiceProvider
services)
{
    public readonly string PromptLanguage = settings.Value.Language ?? "en";

```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

```
public virtual Kernel CreateBasic()
{
    return new SemanticKernelBuilder(settings)
        .AddLogging()
        .AddOpenAiChatCompletion()
        .Build();
}
```

		Осипчук А.О.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		80