Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ПЕРЕВІРКИ КОРЕКТНОСТІ ВИКОНАННЯ СТУДЕНТСЬКИХ РОБІТ

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Алгаш А. М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
«4» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ
Богач І. В.
(прізвище та ініціали)
«5» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«6» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«5» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Алгашу Анатолію Михайловичу

1. Тема роботи: Програмний модуль перевірки коректності виконання студентських робіт.
керівник роботи: Крилик Людмила Вікторівна, к.т.н., доц.
затверджені наказом вищого навчального закладу «20» 03.2025 року № 97
2. Термін подання студентом роботи 30 травня 2025 р
3. Вихідні дані до роботи: мова програмування – об'єктно-орієнтована; підтримуючі формати файлів: .doc, .docx, .pdf; кількість підтримуваних платформ не менше 3; інтерфейс користувача має бути інтуїтивно зрозумілий; не має обмежень щодо мови написання студентської роботи.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки програмного модуля перевірки коректності виконання студентських робіт; проектування програмного модуля перевірки коректності виконання студентських робіт; програмна реалізація програмного модуля перевірки коректності виконання студентських робіт; висновки; список використаних джерел; додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): загальна структурна схема функціонування програмного модуля перевірки коректності виконання студентських робіт; загальна структурна схема компонентів програмного модуля перевірки коректності виконання студентських робіт; UML-діаграма класів серверної частини програмного модуля; UML-діаграма класів клієнтської частини програмного модуля; схема алгоритму роботи програмного модуля; ER-модель для бази даних програмного модуля; діаграма діяльності класу LabEntityConfiguration; діаграма діяльності класу MarkLabHandler; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Крилик Л. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання «05» травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки програмного модуля перевірки коректності виконання студентських робіт	05.05.25	07.05.25	
2	Проектування програмного модуля перевірки коректності виконання студентських робіт	08.05.25	11.05.25	
3	Програмна реалізація програмного модуля перевірки коректності виконання студентських робіт	12.05.25	15.05.25	
4	Висновки та аналіз отриманих результатів	16.05.25	26.05.25	
5	Оформлення додатків	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Алгаш А. М.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крилик Л. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 90 сторінок формату А4, які включають 23 рисунка і 7 таблиць. У списку використаних джерел зазначено 28 найменувань.

Метою роботи є розширення функціональних можливостей програмного модуля перевірки коректності виконання студентських робіт.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій доведена доцільність розробки програмного модуля перевірки коректності виконання студентських робіт. Розроблено алгоритм роботи та UML-діаграми класів клієнтської та серверної частин програмного модуля перевірки коректності виконання студентських робіт, які спрощують розробку та розуміння структури програмного забезпечення. Програмний продукт розроблений в інтегрованих середовищах Rider та WebStorm.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції програмного модуля перевірки коректності виконання студентських робіт реалізовано.

Ключові слова: програмний модуль, WEB-розробка, студентські роботи, оцінювання, автоматизована перевірка, допомога викладачам.

ABSTRACT

The bachelor's thesis consists of 90 pages of A4 format, which include 23 figures and 7 tables. The list of used sources includes 28 items.

The purpose of the work is to expand the functionality of the software module for checking the correctness of student work.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies, the feasibility of developing a software module for checking the correctness of student work is proven. An algorithm of work and UML class diagrams of the client and server parts of the software module for checking the correctness of student work have been developed, which simplify the development and understanding of the software structure. The software product is developed in the Rider and WebStorm integrated environments.

The software product has been developed and tested. The results of the development testing indicate that the main functions of the software module for checking the correctness of student work have been implemented.

Keywords: software module, WEB development, student work, evaluation, automated testing, assistance to teachers.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Головна мета вищих навчальних закладів – це забезпечити якісну освіту, вдосконалюючи студентський контингент, викладацький склад, методи навчання, а також поглиблюючи інтеграцію освіти, науки та інновацій відповідно до діючих стандартів освіти.

Важливість цього питання підкреслюють такі аспекти:

- Зростання кількості здобувачів освіти: все більше людей прагнуть здобути вищу освіту, тому кількість студентських робіт, які потрібно перевірити та оцінити викладачами збільшується. Саме тому розробка програмного модуля перевірки коректності виконання студентських робіт стає дедалі важливішою. За допомогою нього необхідність у перевірці студентських робіт вручну стає необов'язковим.

- Потреба в об’єктивності оцінювання: людський фактор при перевірці студентських робіт може призвести до суб'єктивності або помилок. Програмний модуль перевірки коректності виконання студентських робіт використовує стандартизовані алгоритми для перевірки правильності, що допомагає забезпечити прозорість та об'єктивність оцінювання.

- Економія часу викладачів: ручна перевірка великих обсягів студентських робіт займає велику кількість часу, що обмежує можливості викладачів зосередитися на індивідуальній роботі студентів, дослідницькій діяльності або розробці нових методичних матеріалів. Використання програмного модуля перевірки коректності виконання студентських робіт автоматизує процес перевірки, значно скорочуючи час, необхідний для оцінювання та підвищуючи загальну ефективність навчального процесу.

Отже, розробка програмного модуля перевірки коректності виконання студентських робіт є важливою та актуальною в контексті сучасного освітнього процесу і має практичне значення.

Метою дослідження є розширення функціональних можливостей програмного модуля перевірки коректності виконання студентських робіт.

Об’єктом дослідження є процес перевірки студентських робіт.

Предметом дослідження є програмні засоби для реалізації програмного модуля перевірки коректності виконання студентських робіт.

Задачі дослідження:

1. Обґрунтувати доцільність розробки програмного модуля перевірки коректності виконання студентських робіт;
2. Спроектувати програмний модуль перевірки коректності виконання студентських робіт;
3. Програмно реалізувати програмний модуль перевірки коректності виконання студентських робіт;
4. Виконати тестування програми та провести аналіз отриманих результатів;
5. Розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) м. Вінниці у 2025 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1] та статтю в журналі «Вісник Хмельницького національного університету» серія: «Технічні науки», який входить до переліку наукових фахових видань МОНУ (категорія Б) на тему: «Розробка програмного модуля перевірки коректності виконання студентських робіт» [2]; отримано свідоцтво про реєстрацію авторського права на твір у формі комп’ютерної програми – «Програмний модуль перевірки коректності виконання студентських робіт» [3].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ПЕРЕВІРКИ КОРЕКТНОСТІ ВИКОНАННЯ СТУДЕНТСЬКИХ РОБІТ

У сучасному світі цифрових технологій, де освіта стає доступнішою, а конкуренція між освітніми платформами та університетами зростає, розробка програмного модуля для перевірки коректності виконання робіт студентами стала не лише актуальною, а й вкрай необхідною. В умовах стрімкого розвитку інформаційних технологій виникає потреба у впровадженні ефективних рішень, які сприяють підвищенню якості освітнього процесу. Такий модуль є ключовим елементом у забезпеченні об'єктивності оцінювання та впорядкуванні щоденних академічних процедур. Наведемо переваги, які отримають викладачі, студенти та навчальні заклади використовуючи це рішення:

1. Економія часу для викладачів: автоматизація процесу перевірки студентських робіт значно скорочує час, який викладачі витрачають на аналіз та оцінювання, дозволяючи їм зосередитися на дослідницькій роботі, підготовці до нового курсу або наданні індивідуальної допомоги студентам.

2. Підвищення об'єктивності: використання програмних модулів виключає людський фактор, який може призвести до помилок або суб'єктивних підходів у процесі оцінювання. Це забезпечує справедливість та прозорість виставлення оцінок.

3. Швидка перевірка великих обсягів робіт: зі збільшенням кількості студентів, програмний модуль перевірки коректності виконання студентських робіт дозволяє швидко обробляти великі обсяги завдань, зберігаючи високий рівень точності перевірки навіть у напружені періоди, наприклад, під час екзаменів.

4. Покращення якості навчання: завдяки автоматичному аналізу завдань студенти отримують швидкий зворотній зв'язок, що дозволяє їм швидко виправляти помилки та вдосконалювати свої знання та навички.

Таким чином, розробка програмного модуля перевірки коректності виконання студентських робіт забезпечує значні переваги для викладачів, студентів і навчальних закладів, сприяючи підвищенню якості освітнього процесу, об'єктивності оцінювання та ефективності управління навчальними завданнями.

1.1 Огляд середовищ для реалізації програмного продукту

Вибір найкращого середовища для створення програмного продукту є ключовим аспектом процесу його розробки. Враховуючи різноманітність доступних платформ та їх функціональних можливостей, важливо приймати рішення на основі потреб користувачів, специфіки проекту та функціональних вимог.

В таблиці 1.1 подано порівняльна характеристика платформ для розробки програмного модуля перевірки коректності виконання студентських робіт [4–6].

Таблиця 1.1 – Характеристика платформ для розробки програмного модуля перевірки коректності виконання студентських робіт

Параметр	Персональні комп’ютери	Смартфони та планшети	WEB-застосунки
Тип доступу	Фізичний доступ до пристрою в робочому чи домашньому середовищі	Доступ будь-де за допомогою портативного пристрою	Доступ через будь-який браузер із підключенням до Інтернету
Підтримка платформ	Працює на Windows, macOS, Linux	Підтримка Android та iOS	Сумісний із сучасними WEB-браузерами
Охоплення аудиторії	Менша популярність серед мобільних користувачів	Висока популярність серед широкої аудиторії	Широка доступність для користувачів усіх типів пристроїв

Продовження таблиці 1.1

Параметр	Персональні комп’ютери	Смартфони та планшети	WEB-застосунки
Інсталяція	Вимагає завантаження та встановлення програмного забезпечення	Потребує встановлення додатка через App Store або Google Play	Не потребує встановлення, працює через WEB-браузер
Мобільність	Стаціонарне використання, обмежена портативність	Повна мобільність та зручність	Універсальний доступ із будь-якого пристрою
Технологічні інструменти	Різноманітні інструменти для розробки (.NET, Java, Python, C++)	Обмежені до екосистем Android (Java, Kotlin) та iOS (Swift, Objective-C)	WEB-фреймворки (React, Angular, Django, Flask тощо)
Гнучкість розробки	Висока, завдяки широкому вибору інструментів	Обмежена через залежність від ОС	Найвища, оскільки підтримується багатьма фреймворками

WEB-застосунки є найпопулярнішим вибором для розробки програмного забезпечення завдяки своїй універсальності та доступності. Вони не залежать від конкретної операційної системи або типу пристрою, що робить їх зручними для широкої аудиторії. Все, що потрібно для запуску WEB-застосунку − це доступ до Інтернету та браузера, без додаткового встановлення чи завантаження. Це робить WEB-застосунки доступними для користувачів персональних комп’ютерів, мобільних пристроїв і навіть смарт-телевізорів.

Серед ключових переваг WEB-застосунків − гнучкість і крос-платформеність. WEB-застосунки не потребують оновлень на стороні користувача, оскільки всі зміни та вдосконалення відбуваються на сервері. Це дозволяє швидко вносити покращення, виправляти помилки та додавати нові функції. Крім того, WEB-застосунки підтримують сучасні фреймворки та технології, такі як React, Angular або Vue.js, що робить їх швидкими, інтерактивними та простими у використанні [7].

Багатофункціональність WEB-застосунків дозволяє розробникам реалізувати широкий спектр можливостей − від простих інформаційних сторінок до складних систем управління даними або інтерактивних платформ. Вони ідеально підходять як для особистого використання, так і для корпоративного середовища, забезпечуючи функціональність, адаптовану до потреб користувача. Завдяки можливості інтеграції з хмарними сервісами, API та іншими платформами, WEB-застосунки забезпечують не лише зручність, але й масштабованість для будь-яких завдань.

На основі аналізу даних з таблиці 1.1, стає очевидним, що WEB-застосунки є найкращим вибором для розробки програмного модуля перевірки коректності виконання студентських робіт.

1.2 Огляд систем-аналогів

На ринку існує багато програмних продуктів зі схожим функціоналом, кожен з яких має свої сильні та слабкі сторони. Тому важливо проаналізувати подібні системи, щоб виявити можливості для вдосконалення та розвитку створюваного модуля.

Для порівняння, розглянемо такі сервіси, як «Grammarly», «Codio», «SonarCube» [4 − 6].

Grammarly − це онлайн-сервіс для перевірки правильності написаного тексту англійською мовою. Він виправляє граматичні, орфографічні та пунктуаційні помилки, надає стилістичні поради та допомагає покращити якість написання тексту (рис. 1.1).

Переваги онлайн-сервісу Grammarly:

- рекомендації щодо стилю написання тексту;
- корпоративні рішення для узгодження стилю для групи людей;
- підтримка різних тонів написання тексту [4].

Недоліки:

- підтримка лише англійської мови;

- обмежена безкоштовна версія. Основні функції безкоштовні, але їх недостатньо для складних текстів;

- недосконалість AI. Іноді рекомендації можуть бути неприйнятними або некоректними для контексту.

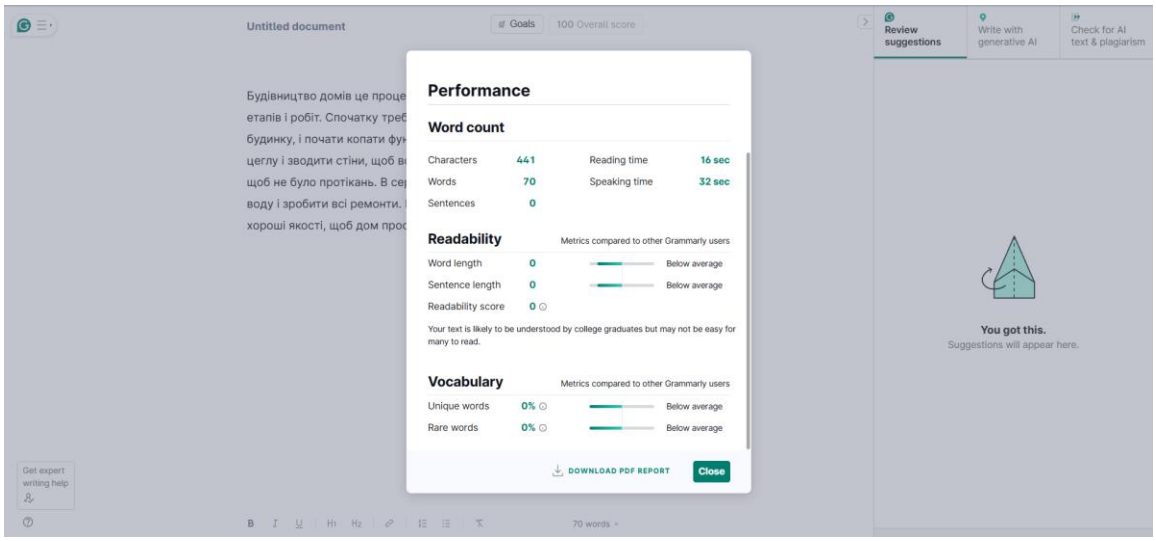

Рисунок 1.1 − Загальний вигляд інтерфейсу робочого полотна Grammarly

Codio − це хмарна платформа, яка автоматизує тестування знань з програмування. Вона пропонує інтерактивне навчання, автоматизує завдання для оцінювання та надає інструменти для покращення навичок програмування (рис. 1.2).

Переваги хмарної платформи Codio:

- автоматизоване оцінювання. Codio дозволяє перевіряти завдання автоматично, що зменшує навантаження на викладача та пришвидшує процес зворотного зв'язку;

- тестування коду. Платформа підтримує написання автоматизованих тестів для перевірки правильності програм, наданих студентами;

- гнучкість перевірки. Викладачі можуть налаштовувати критерії оцінювання, включаючи логіку коду, ефективність та стиль;

- вбудована підтримка середовищ. Codio дозволяє перевіряти знання різних мов програмування, що стає в нагоді для багатопрофільних курсів [5].

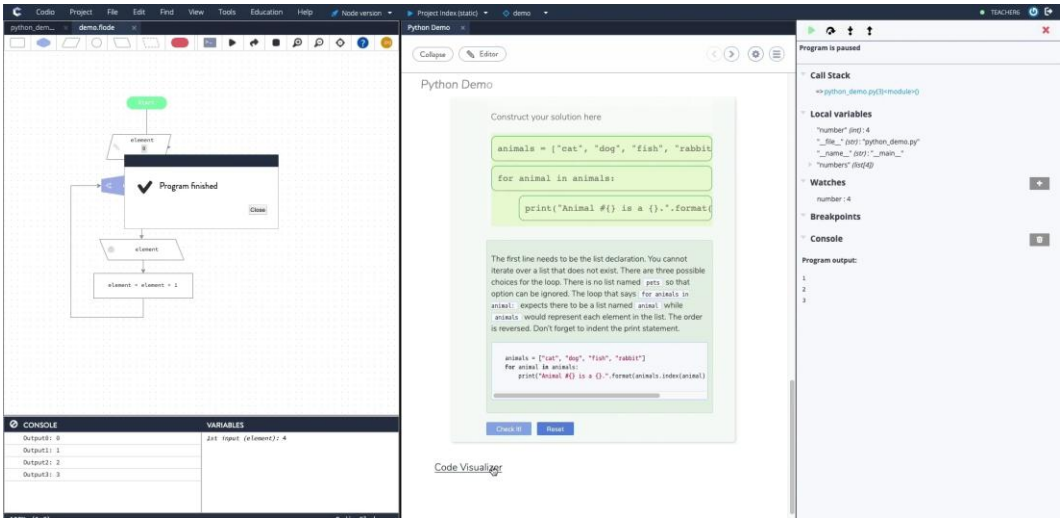

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна хмарної платформи Codio

Недоліки хмарної платформи Codio:

- обмеження у складних завданнях. Платформа іноді не в змозі ефективно тестувати складні або творчі завдання, які потребують ручного оцінювання;

- проблеми з автоматизацією. Автоматизовані тести можуть не враховувати контекстні або нестандартні рішення учнів (студентів), що може призвести до заниження балів;

- складність налаштувань. Викладачам може бути складно кастомізувати завдання з унікальними вимогами, які потребують додаткових зусиль.

SonarQube − це платформа статичного аналізу коду, яка допомагає виявляти помилки, вразливості, дублювання коду, проблеми зі стилем та покращувати загальну якість програмного забезпечення (рис. 1.3).

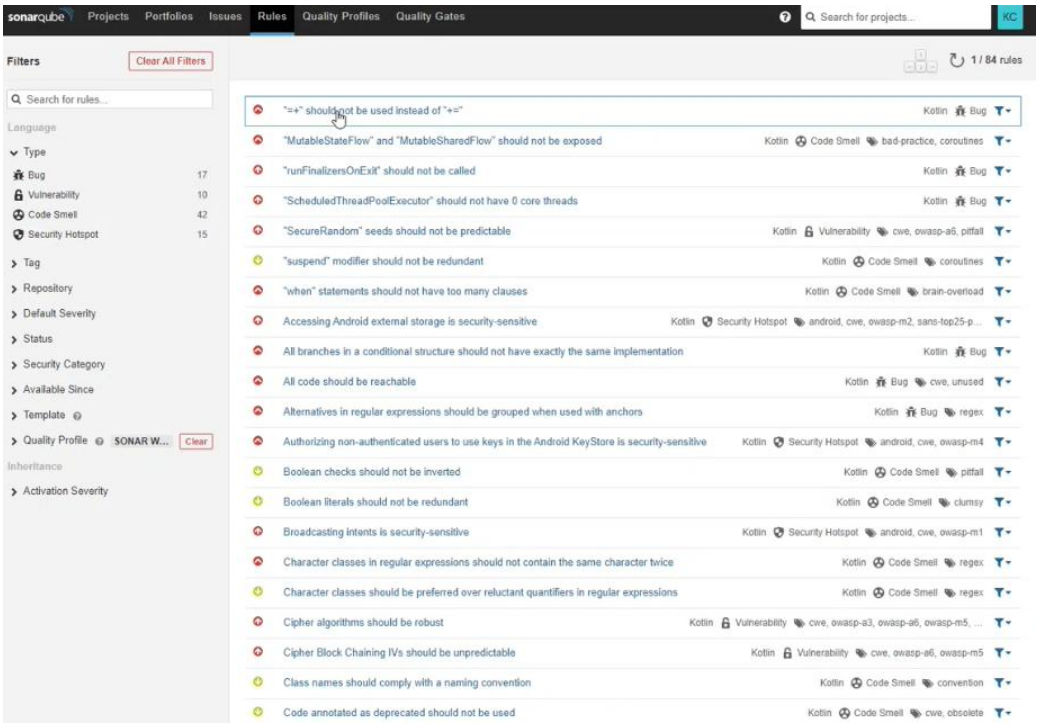

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна SonarQube

Переваги платформи SonarQube:

- статичний аналіз коду. Виявлення помилок, вразливостей, дублювання коду, порушень стилю та інших проблем, що впливають на якість коду;

- оцінка технічного боргу. Платформа аналізує технічний борг, оцінюючи час, який знадобиться для виправлення проблем у вашому коді;

- підтримка багатьох мов програмування. Підтримує понад 25 мов, включаючи Java, C#, JavaScript, Python, PHP та інші;

- метрики якості. Надає детальні показники, такі як покриття тестів, кількість вразливостей, складність коду тощо [6].

Недоліки платформи SonarQube:

- високі вимоги до ресурсів. Для великих проектів потрібні сервери з потужним апаратним забезпеченням;

- складність налаштування. Початкова конфігурація може бути складною для нових користувачів;

- часовитратність аналізу. Для великих проектів повний аналіз може зайняти значну кількість часу.

1.3 Формулювання вимог та постановка задачі

Зі збільшенням кількості студентів ускладняється процес перевірки студентських робіт для викладачів. Часто перевірка забирає занадто багато часу та зусиль, тому використання програмного забезпечення для перевірки коректності виконання студентських робіт може значно спростити процес і вирішити ці проблеми.

SonarQube, Codio та Grammarly пропонують різноманітні інструменти для аналізу та перевірки тексту або коду, кожен з яких має свої унікальні сильні та слабкі сторони. SonarQube пропонує статичний аналіз, оцінку технічних помилок, підтримку декількох мов програмування та детальні показники якості, але є ресурсоємним, складним у налаштуванні та може зайняти багато часу для аналізу. Codio автоматизує перевірку завдань, підтримує тестування коду та різні мови програмування, але має обмеження щодо складних завдань, проблем з автоматизацією нестандартних рішень та складнощів у створенні унікальних критеріїв оцінювання. Grammarly допомагає покращити стиль тексту, підтримує корпоративні рішення та різні стилі написання, але доступний лише для англійської мови, має обмежену кількість безкоштовних функцій, а його підказки зі штучним інтелектом не завжди контекстуально релевантні.

Альтернативою SonarQube, Codio та Grammarly є сервіс Assessor, який здатен оцінити будь-яку роботу студента, незалежно від її типу. Assessor поєднує в собі автоматичну перевірку тексту та програмного коду, аналіз стилю, логіки, правильності виконання завдань та творчих робіт. Він підтримує мультидисциплінарні курси, легко налаштовується під унікальні вимоги, враховує контекст завдання, дозволяє оцінювати технічні та гуманітарні аспекти. Assessor

вирішує проблеми складного налаштування, враховуючи мову написання роботи студентом, ресурсів та неточностей у процесі оцінювання, забезпечуючи універсальне рішення для викладачів та студентів.

Програмний модуль перевірки коректності виконання студентських робіт матиме такі функції:

- Автоматизована перевірка студентських робіт.

- Вікно з переглядом останніх перевірених робіт.

- Вікно для налаштування параметрів нейромережі для перевірки студентських робіт.

У роботі потрібно створити WEB-сервіс, який дасть можливість користувачам з легкістю перевіряти студентські роботи.

WEB-сервіс Assessor буде розроблений з урахуванням принципів користувацького досвіду (UX), забезпечуючи інтуїтивно зрозумілий інтерфейс для користувачів. Введення даних буде максимально простим та зручним, що дозволить користувачам легко налаштовувати параметри оцінювання. Результати перевірки будуть представлені у зрозумілій та структурованій формі, щоб користувачі могли швидко знайти необхідну інформацію без зайвих зусиль.

Для написання коректного програмного модуля перевірки коректності виконання студентських робіт потрібно вирішити такі завдання:

- реалізувати модуль для автоматизованої перевірки студентських робіт;
- реалізувати модуль для взаємодії з останніми перевіреними роботами;
- реалізувати модуль для налаштування параметрів нейромережі для перевірки студентських робіт;
- здійснити тестування програмного модуля перевірки коректності виконання студентських робіт.

1.4 Висновок

У цьому розділі проаналізовано існуючі платформи для реалізації програмного модуля перевірки коректності виконання студентських робіт. На основі проведеного аналізу було прийнято рішення реалізувати програмний модуль у вигляді WEB-застосунку в браузері. Таке рішення обґрунтовано тим, що WEB-застосунки можуть запускатися на будь-якому сучасному браузері, незалежно від операційної системи або пристрою. WEB-застосунки не потребують інсталяції, доступні з будь-якого місця через мережу Інтернет, дозволяють легко оновлювати функціонал без необхідності виконання користувачем додаткових дій.

Було проаналізовано такі системи моделювання, як Grammarly, Codio та SonarQube. Незважаючи на потужність цих платформ, вони не повністю задовольняють потреби користувачів, оскільки всі вони зосереджені на вузькоспеціалізованих завданнях.

Програмний модуль для перевірки коректності виконання студентських робіт є актуальним, оскільки забезпечить: автоматизовану перевірку; об'єктивність оцінювання та швидкий зворотній зв'язок зі студентами; зекономить час викладачів. Це значно спрощує управління завданнями, особливо у випадку великої кількості студентів та сприяє підвищенню якості навчального процесу.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ПЕРЕВІРКИ КОРЕКТНОСТІ ВИКОНАННЯ СТУДЕНТСЬКИХ РОБІТ

2.1 Розробка структури програмного модуля перевірки коректності виконання студентських робіт

Програмний модуль перевірки коректності виконання студентських робіт є сучасним WEB-застосунком, який складається з двох основних частин − клієнтської та серверної. Клієнтська частина відповідає за взаємодію користувача з системою через зручний інтерфейс, а серверна частина забезпечує виконання основної логіки роботи модуля, включаючи перевірку студентських робіт, обробку запитів та генерацію результатів. Для спрощення процесу розробки та підвищення зручності підтримки кожна з цих частин поділена на модулі, які відповідають за конкретні функції. У серверній частині модулі називаються хендлерами і вся архітектура проєкту базується на принципі CQS (Command Query Separation), що дозволяє ефективно організувати бізнес-логіку.

Простими словами, CQS (Command Query Separation) − це архітектурний підхід у програмному забезпеченні, який розділяє всі операції на дві категорії: команди (Command), які змінюють стан системи та запити (Query), які лише отримують дані без змін. Це розділення дозволяє зробити структуру проєкту зрозумілішою, а код − легшим для підтримки, внесення змін та тестування. У програмному модулі перевірки коректності виконання студентських робіт цей підхід використовується для чіткої організації бізнес-логіки. Для взаємодії між різними службами модуля використовується спільний інтерфейс у вигляді API, який забезпечує швидку і стандартизовану комунікацію між компонентами системи [7].

Оскільки CQS-архітектура чітко розділяє логіку виконання на команди та запити, вона дозволяє значно покращити продуктивність та модульність

програмного забезпечення. Розділення завдань зменшує залежності між компонентами, що спрощує управління змінами та внесення оновлень. Крім того, завдяки такому підходу кожен модуль або хендлер стає легким для тестування, оскільки він виконує лише одну конкретну функцію. Також використання CQS забезпечує кращу масштабованість системи, оскільки розробники можуть додавати нові функції без ризику вплинути на існуючу логіку.

Хендлери є основою серверної частини програмного модуля і забезпечують виконання окремих завдань системи. Кожен хендлер виконує конкретну функцію, що робить код структурованим і зрозумілим для розробників. Наприклад, хендлери відповідають за перевірку студентської роботи, надання детальних результатів, включаючи оцінки, зауваження та висновки, отримання списку останніх перевірених робіт, а також налаштування параметрів нейромережі. Такий підхід дозволяє з легкістю додавати нові функції, підтримувати існуючі компоненти та масштабувати систему відповідно до потреб [8].

Для програмного модуля перевірки коректності виконання студентських робіт було створено кілька ключових модулів, кожен з яких виконує важливу роль у загальній системі:

- модуль перевірки студентських робіт – основний компонент, що автоматизує процес перевірки завдань, аналізує їх на відповідність заданим критеріям, надає об’єктивні оцінки та формує розгорнутий висновок;
- модуль перегляду останніх перевірених робіт − дозволяє користувачам отримати доступ до результатів попередніх перевірок, надаючи зручний інструмент для аналізу та перегляду історії перевірок;
- модуль налаштування параметрів нейромережі – забезпечує можливість адаптувати систему до специфічних потреб викладачів та студентів шляхом налаштування параметрів нейромережі, таких як температура, кількість токенів, параметри ймовірності тощо.

Кожен із цих модулів інтегрований у загальну структуру системи, що забезпечує її ефективність, зручність у використанні та відповідність сучасним вимогам освітніх процесів. Загальна структурна схема функціонування програмного модуля перевірки коректності виконання студентських робіт зображена на рисунку 2.1.

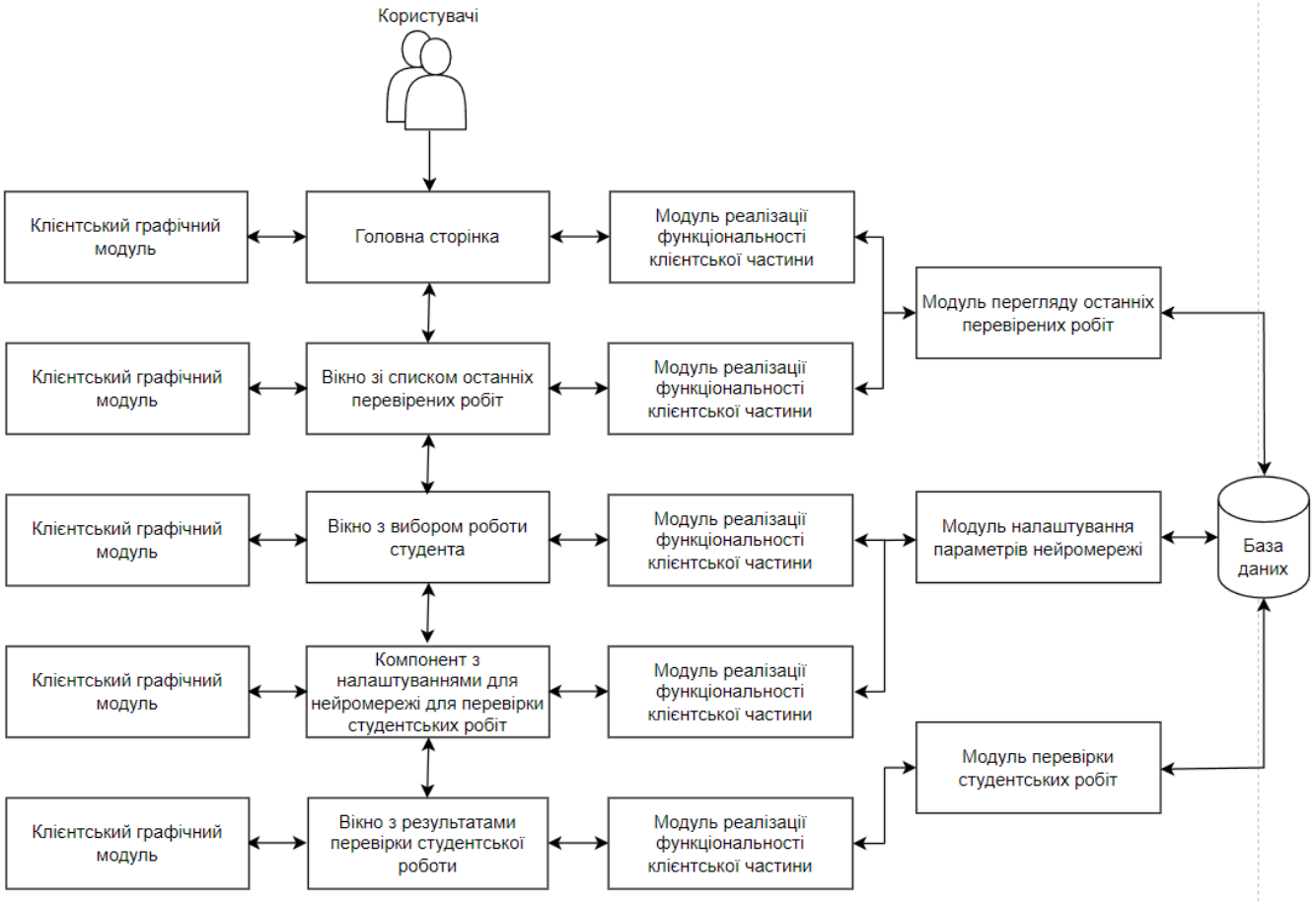

Рисунок 2.1 – Загальна структурна схема функціонування програмного модуля перевірки коректності виконання студентських робіт

Головна сторінка є стартовою точкою взаємодії користувача з програмним модулем. Вона містить вікно зі списком останніх перевірених робіт, де користувач може переглянути вже проаналізовані завдання. Це вікно формує API-запит до модуля перегляду останніх перевірених робіт, який обробляє запит на сервері, отримує дані з бази даних та відправляє відповідь до клієнтської частини. Усі

отримані дані, включаючи назву роботи, оцінку, дату перевірки та зауваження, виводяться у зручному форматі.

На головній сторінці також розташована кнопка для завантаження файлу роботи студента. Після натискання кнопки відкривається вікно з вибором роботи студента, де користувач може вибрати потрібний файл. Запит із завантаженням цього файлу передається на сервер у модуль налаштування параметрів нейромережі, який зберігає інформацію про файл у базі даних для подальшої обробки.

Наступний етап – відкриття компонента з налаштуваннями для нейромережі, де користувач може задати такі параметри, як температура, максимальна кількість токенів, Топ Р, штраф за частотність та присутність. Введені значення передаються через API-запит до того ж модуля налаштування параметрів нейромережі, який записує налаштування в базу даних.

Після налаштування параметрів користувач натискає кнопку для запуску перевірки роботи. Запит надходить до модуля перевірки студентських робіт, який виконує основний аналіз завантаженого файлу. Цей модуль використовує задані параметри нейромережі для перевірки коректності виконання роботи, аналізує завдання та генерує результати. Після завершення аналізу результати (оцінка, зауваження, висновок) зберігаються у базі даних.

Кінцевим етапом є вікно з результатами перевірки студентської роботи, яке відображає результати аналізу. Всі дані отримуються з бази через API-запит, що передається клієнтським модулем до модуля перевірки студентських робіт.

Сервер зберігає всі дані у реляційній базі даних, яка є основним сховищем інформації про параметри нейромережі, завантажені студентські роботи та результати їх перевірки. Ця база даних забезпечує швидкий доступ до збереженої інформації, а також підтримує її оперативне оновлення та додавання нових записів. Завдяки цьому система функціонує ефективно навіть за умови значної кількості запитів від користувачів.

Усі API-запити від клієнтської частини до серверних мікросервісів проходять через балансувальник навантаження. Основна задача балансувальника – рівномірно розподіляти навантаження між мікросервісами, забезпечуючи оптимальну продуктивність системи. Це дозволяє уникнути перевантаження окремих серверів, забезпечити стабільність роботи всієї системи та мінімізувати час відповіді на запити користувачів. Балансувальник також відіграє ключову роль у підвищенні надійності системи, гарантуючи доступність сервісу навіть за умов пікових навантажень.

Загальна структурна схема компонентів програмного модуля перевірки коректності виконання студентських робіт зображена на рисунку 2.2.

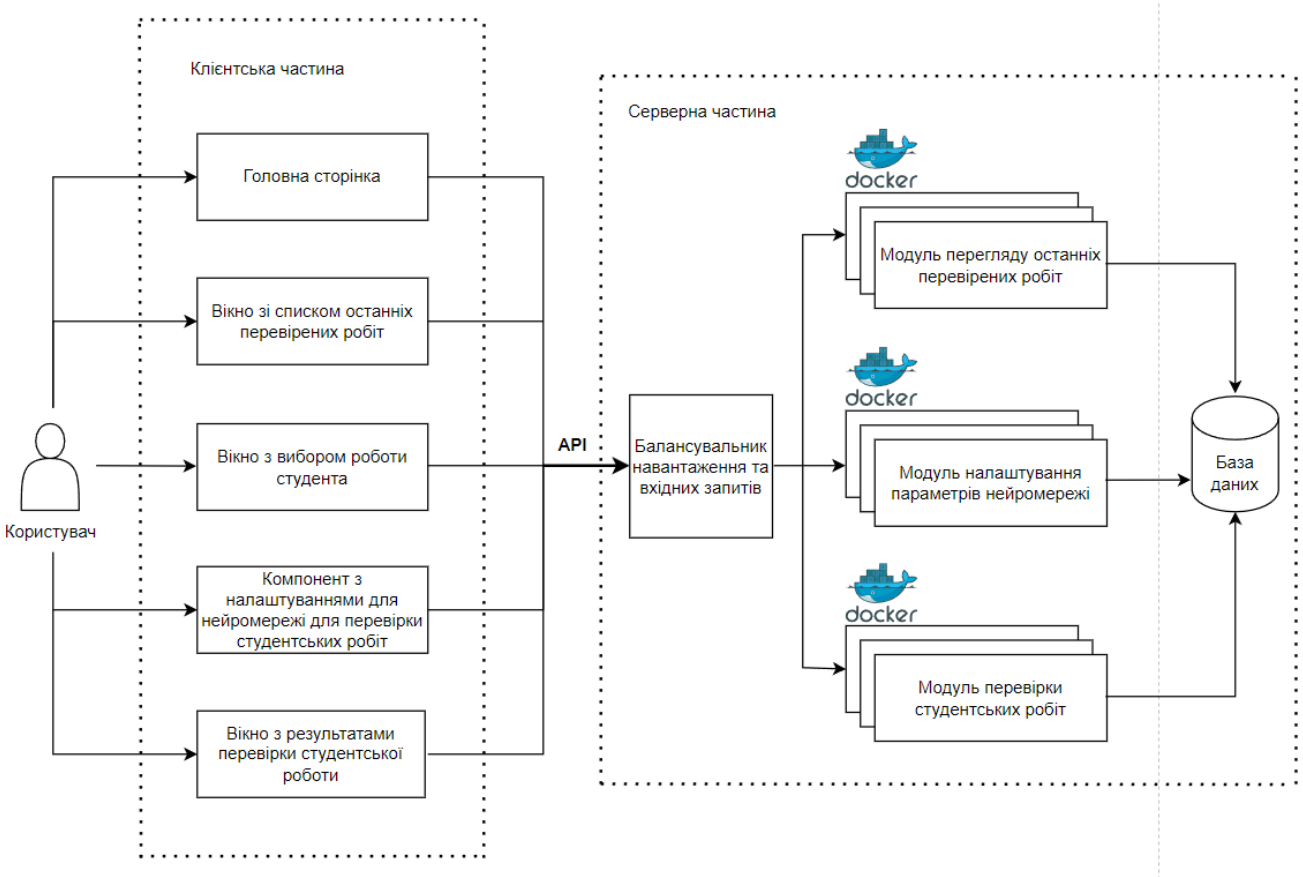

Рисунок 2.2 –Загальна структурна схема компонентів програмного модуля перевірки коректності виконання студентських робіт

2.2 Розробка UML-діаграми класів

Мова програмування C# була створена з урахуванням концепцій об’єктно-орієнтованого програмування (ООП), що є ключовою парадигмою для розробки сучасного програмного забезпечення. Використання ООП у C# дозволяє будувати масштабовані, гнучкі та легко підтримувані програми, що особливо важливо для складних програмних рішень. Завдяки підтримці основних концепцій ООП, C# дозволяє створювати код, що має чітку структуру, добре організований та легко розширюється, це сприяє ефективній розробці великих програмних систем.

Об’єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка організовує код у вигляді об'єктів, що взаємодіють між собою. Об'єкти є сутностями, які поєднують у собі дані (поля, властивості) та методи (функції), що визначають їх поведінку. Основною ідеєю ООП є створення програмного забезпечення, яке базується на моделюванні реальних об’єктів та їх відносин між собою. Такий підхід значно спрощує розробку, тестування, масштабування та підтримку програмного коду. ООП дозволяє зменшити дублювання коду, підвищує його повторне використання та забезпечує зручну організацію програмних компонентів.

ООП базується на чотирьох основних принципах: інкапсуляція, наслідування, поліморфізм та абстракція [8].

- Інкапсуляція − це механізм обмеження доступу до внутрішніх даних об'єкта, дозволяючи працювати з ними лише через спеціально визначені методи (гетери та сетери). Це забезпечує захист даних від небажаних змін і підвищує безпеку коду. У C# це реалізується за допомогою модифікаторів доступу (private, public, protected).

- Наслідування – це механізм, що дозволяє одному класу (підкласу) успадковувати властивості та методи іншого класу (батьківського). Це сприяє

повторному використанню коду та спрощує підтримку програмного забезпечення. У C# наслідування реалізується через ключове слово : base [9].

- Поліморфізм − це можливість об'єктів різних класів реагувати на однакові виклики методів по-різному. Це дозволяє створювати гнучкий та розширюваний код. У C# поліморфізм реалізується за допомогою віртуальних методів (virtual), методів перевизначення (override) та інтерфейсів (interface) [10].
- Абстракція – це концепція, яка дозволяє виділити загальні характеристики об'єкта, приховуючи деталі його реалізації. Це забезпечує простоту у використанні класів і підвищує рівень узагальнення коду. У C# абстракція реалізується через абстрактні класи (abstract class) та інтерфейси (interface) [11].

Завдяки цим принципам ООП у C# дозволяє створювати ефективні, добре організовані та масштабовані програмні рішення, що є основою сучасного програмування [11].

У серверній частині програмного модуля перевірки коректності виконання студентських робіт є такі ключові класи: LabController, MarkLabHandler, OpenAiHttpClient.

Клас LabController є частиною серверної логіки програмного модуля перевірки коректності виконання студентських робіт і відповідає за обробку запитів, пов’язаних із перевіркою та отриманням результатів оцінювання. Він містить два основні методи: MarkLab та GetResult. Метод MarkLab приймає файл студентської роботи та параметри нейромережі, такі як температура, максимальна кількість токенів, топ P, штраф за частотність і штраф за присутність, після чого формує команду MarkLabCommand та передає її в Mediator для подальшої обробки. Метод GetResult отримує список результатів перевірених студентських робіт з бази даних через Entity Framework Core, використовуючи AsNoTracking для підвищення продуктивності і повертає структуровану відповідь у форматі JSON, а саме, оцінку, зауваження, висновок та параметри перевірки. Контролер використовує AssessorContext для роботи з базою даних, а також наслідується від

ApiControllerBase, що містить загальні механізми для роботи з API-запитами. Таким чином, LabController забезпечує взаємодію між клієнтською частиною та сервером, обробляючи запити на перевірку студентських робіт та надання результатів аналізу.

Клас MarkLabHandler є обробником команди MarkLabCommand у патерні CQRS і відповідає за перевірку студентських робіт. Він використовує IAiHttpClient для взаємодії з AI-сервісом, IServiceProvider для отримання потрібного сервісу зчитування файлів та AssessorContext для взаємодії з базою даних. Метод Handle визначає тип файлу, перевіряє його підтримку та використовує відповідний IFileReaderService для отримання тексту. Далі створюється модель completionSettingsModel, яка містить параметри для AI-аналізу і запит передається в IAiHttpClient.GetCompletion. Якщо аналіз успішний, формується сутність ResultEntity, що містить оцінку, зауваження, висновок та параметри нейромережі, після чого результат зберігається у базі даних. Якщо жоден рядок не було змінено, повертається помилка, інакше клієнт отримує відповідь із результатами перевірки.

Клас OpenAiHttpClient реалізує інтерфейс IAiHttpClient та відповідає за взаємодію із сервісом OpenAI для отримання результатів обробки студентських робіт. Він використовує HttpClient для виконання HTTP-запитів і OpenAiOptions для збереження налаштувань OpenAI, таких як модель, API-ключ та URL. Метод GetCompletion приймає текст повідомлення та параметри нейромережі (CompletionSettingsModel), формує JSON-запит із необхідною структурою, що містить ролі системи, користувача та асистента, а також параметри генерації тексту (температуру, максимальну кількість токенів, top_p, штраф за частотність та присутність). Далі запит надсилається до OpenAI через HttpClient, з відповідними заголовками для авторизації (Bearer-токен) та вказаного медіа-типу (application/json). Якщо відповідь успішна, вона десеріалізується у OpenAiResponse, після чого з отриманого об'єкта вилучається текстовий результат (Message.Content). Якщо дані відсутні або їх не вдається десеріалізувати, повертається відповідна помилка. В іншому випадку, десеріалізований об'єкт Content містить оцінку,

зауваження та висновок, які повертаються у вигляді результату. Таким чином, OpenAiHttpClient забезпечує отримання перевірених даних від OpenAI та обробку їх для подальшого використання в системі.

На рисунку 2.3 зображена UML-діаграма класів серверної частини програмного модуля перевірки коректності виконання студентських робіт.

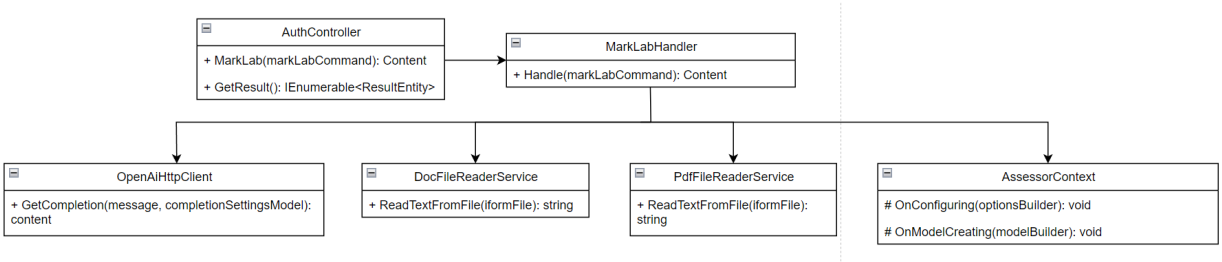

Рисунок 2.3 – UML-діаграма класів серверної частини програмного модуля перевірки коректності виконання студентських робіт

У клієнтській частині програмного модуля перевірки коректності виконання студентських робіт є такі ключові класи: FileUploader, EvaluationSettings, EvaluationResultsDialog, LatestEvaluations, App.

FileUploader – цей компонент відповідає за завантаження файлів студентських робіт. Він містить кнопку вибору файлу, яка відкриває файловий провідник для вибору документа у форматах .doc, .docx або .pdf. Користувач може натиснути на іконку завантаження, після чого файл додається до стану програми, а його ім'я відображається під кнопкою. Компонент передає вибраний файл до головного додатку через onFileSelect.

EvaluationSettings – забезпечує користувача інтерфейсом для введення параметрів нейромережі, які використовуються при аналізі студентських робіт. Він включає кілька полів для налаштувань: температура, максимальна кількість токенів, топ P, штраф за частотність та штраф за присутність. Кожне поле дозволяє вводити числові значення та оновлювати стан налаштувань через onSettingsChange.

EvaluationResultsDialog – цей компонент відповідає за відображення результатів перевірки роботи після її аналізу. Він представлений у вигляді модального вікна, яке відкривається після завершення перевірки. У ньому містяться основні показники: оцінка, список зауважень та підсумковий висновок щодо виконання роботи. Використовує пропси open для керування відображенням і onClose для закриття діалогового вікна.

LatestEvaluations – компонент, який містить список останніх перевірених робіт, представлений у вигляді бічного Drawer. Він отримує список перевірених робіт через пропси results і відображає назву кожної роботи, її оцінку та час перевірки. Завдяки цьому компоненту користувач може швидко переглянути історію перевірок та їх результати.

App – головний компонент, який об'єднує всі модулі для формування єдиного інтерфейсу. Він містить логіку для завантаження файлів, виклику API для перевірки роботи, а також відображення результатів та списку останніх перевірок. App керує станами файлів, налаштувань, результатів та діалогового вікна, взаємодіючи з сервером через HTTP-запити.

На рисунку 2.4 зображена UML-діаграма класів клієнтської частини програмного модуля перевірки коректності виконання студентських робіт.

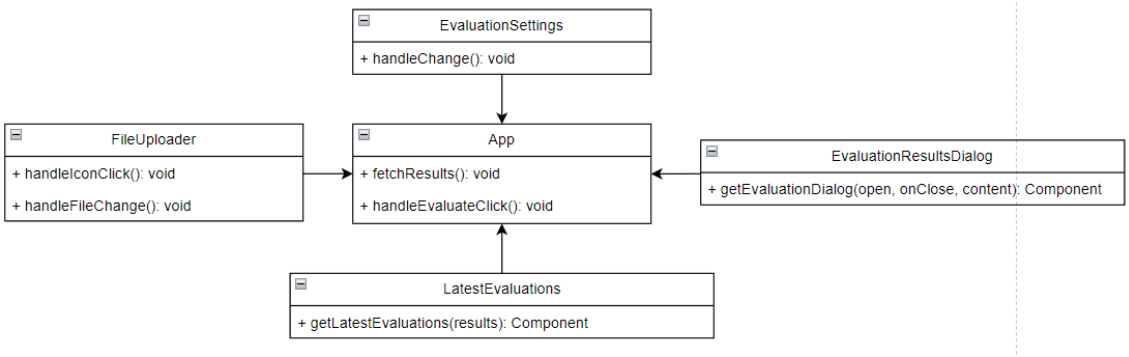

Рисунок 2.4 – UML-діаграма класів клієнтської частини програмного модуля перевірки коректності виконання студентських робіт

2.3 Розробка ER-моделі бази даних

Для проєктування бази даних програмного модуля перевірки коректності виконання студентських робіт створено ER-модель, яка є важливим інструментом у процесі розробки. Вона дозволяє наочно представити взаємозв’язки між основними сутностями, такими як налаштування запитів, роботи студентів та результати оцінювання. Це забезпечує зрозумілу структуру даних та спрощує їхню подальшу реалізацію в базі даних.

Мета створення ER-моделі − забезпечити оптимальну організацію даних у базі, уникнути надлишковості та забезпечити цілісність даних. Важливі компоненти моделі включають такі сутності та їхні атрибути [12]:

- налаштування запитів: параметри, такі як температура, максимальна кількість токенів, ймовірність вибору слова та використання токенів, що впливають на аналіз робіт;

- роботи студентів: атрибути, що описують роботи, включаючи назву, формат файлу, вміст роботи та ідентифікатор;

- результати оцінювання: інформація про оцінку, примітки, висновки, які є результатом аналізу студентських робіт.

ER-модель також визначає важливі типи зв’язків між сутностями:

- зв’язок ОДИН-ДО-БАГАТЬОХ (1:Б) між «Результатами оцінювання» та «Роботами студентів», який відображає, що одна робота може мати кілька результатів оцінювання;

- зв’язок ОДИН-ДО-ОДНОГО (1:1) між «Роботами студентів» та «Налаштуваннями запитів», який вказує на унікальність налаштувань для кожної роботи.

Практична важливість ER-моделі полягає в тому, що вона допомагає:

1. Забезпечити логічну структуризацію даних.
2. Покращити розуміння розробниками взаємозв’язків між сутностями.

3. Спроєктувати базу даних таким чином, щоб вона відповідала вимогам модульності, продуктивності та цілісності даних.

4. Підготувати основу для подальшої реалізації складних запитів та функціональності програмного продукту.

Характеристики відношень між сутностями програмного модуля перевірки коректності виконання студентських робіт представлено у таблиці 2.1.

Таблиця 2.1 – Характеристики відношень між сутностями програмного модуля перевірки коректності виконання студентських робіт

Ім’я сутності 1	Ім’я сутності 2	Тип зв’язку	Клас належності
Результати оцінювання	Роботи студентів	1:Б	Обов’язковий
Роботи студентів	Налаштування запитів	1:1	Обов’язковий

Представлення ER-моделі для бази даних програмного модуля перевірки коректності виконання студентських робіт зображено на рисунку 2.5.

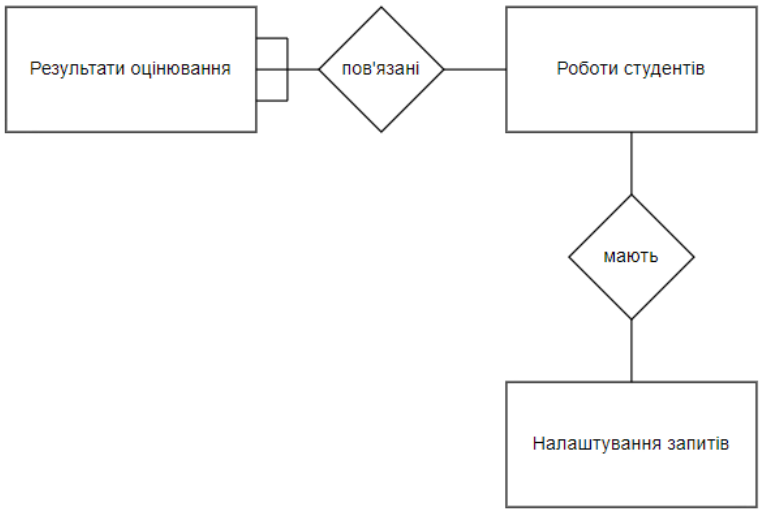

Рисунок 2.5 – ER-модель для бази даних програмного модуля перевірки коректності виконання студентських робіт

2.4 Розробка схеми алгоритму роботи програмного модуля перевірки коректності виконання студентських робіт

Алгоритм – це впорядкована та впевнена послідовність дій або операцій, виконання яких переводить систему чи користувача від початкових вхідних даних до бажаного результату. Кожен крок алгоритму має чітко визначене призначення та умови виконання, що забезпечує однозначність і передбачуваність вихідного результату незалежно від зовнішніх обставин. Завдяки строгій структурі алгоритму можна аналізувати його коректність, ефективність і складність, а також легко вносити зміни чи оптимізації без ризику порушити загальну логіку. Таким чином, алгоритм виступає не лише як послідовність команд, але й як концептуальна модель розв’язання задачі, яку можна реалізувати у вигляді програми, схеми чи словесного опису.

Схема алгоритму – це графічне відображення логіки виконання команд, що допомагає візуалізувати загальну структуру процесу та взаємозв’язки між окремими діями. На схемі кожен тип оператора (наприклад, введення/виведення даних, обчислювальні дії, умови, цикли) позначається стандартним геометричним символом, з’єднаним стрілками, які демонструють напрямок і порядок виконання. Такий графічний підхід спрощує спільну роботу над проєктом, дозволяючи швидко виявити можливі вузькі місця, альтернативні гілки обробки та точки перевірки коректності. Крім того, схема значно полегшує документування алгоритму, його модифікацію та передачу знань іншим розробникам чи користувачам.

Для опису роботи програмного модуля перевірки коректності виконання студентських робіт було обрано блок-схему, яка дозволяє наочно відобразити логіку виконання процесу [13]. Схема алгоритму роботи програмного модуля перевірки коректності виконання студентських робіт зображено на рисунку 2.5.

Рисунок 2.6 – Схема алгоритму роботи програмного модуля перевірки коректності виконання студентських робіт

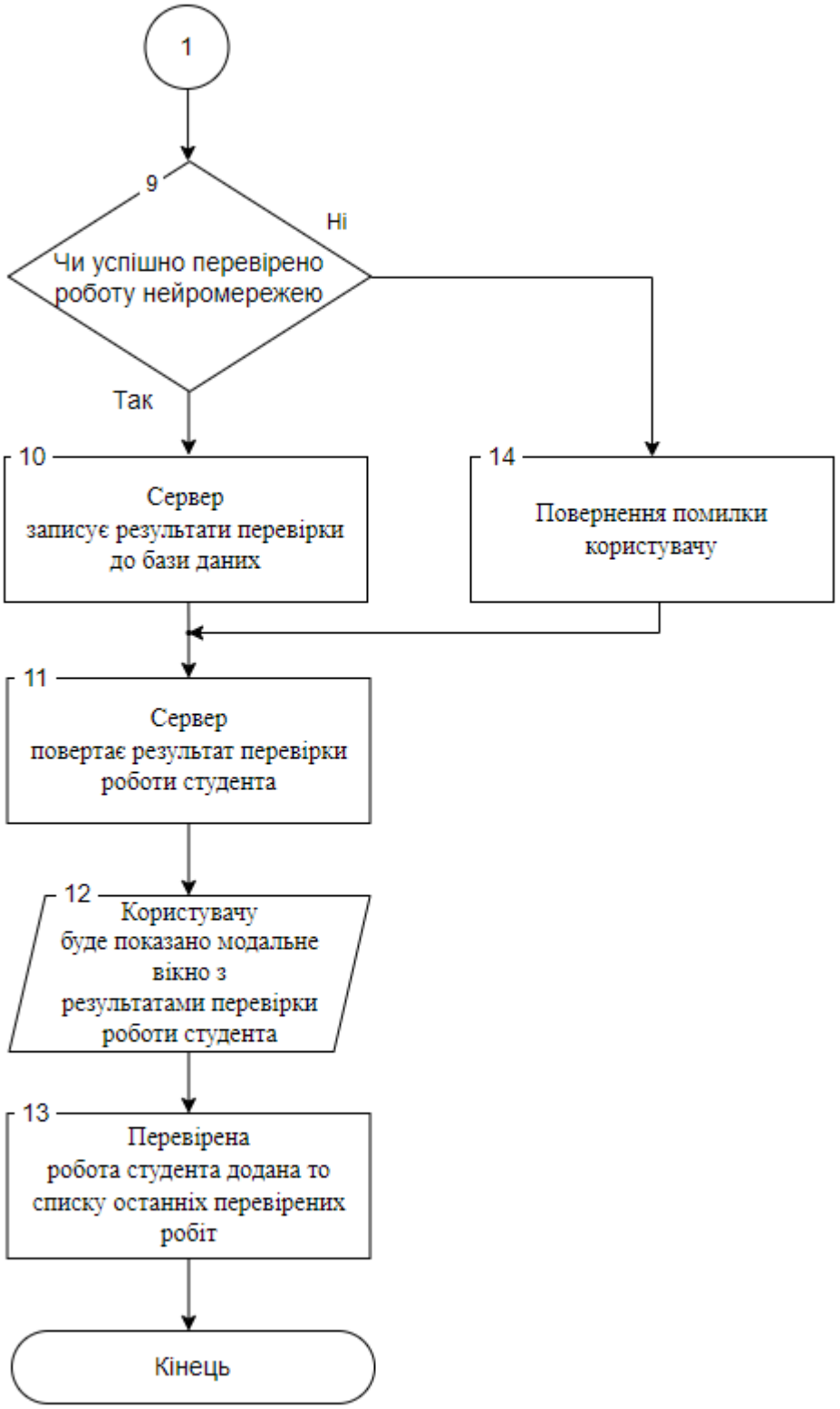

Рисунок 2.6, аркуш 2

Алгоритм оцінювання роботи студента складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку та потрапляє на головну сторінку.

Крок 2. Користувач натискає на іконку для завантаження файлу.

Крок 3. Користувач вибирає файл роботи студента у спливаючому вікні.

Крок 4. Користувач коригує параметри модуля для перевірки коректності виконання студентських робіт.

Крок 5. Користувач натискає кнопку перевірки роботи студента.

Крок 6. Запит з файлом роботи студента та параметрами для нейромережі потрапляють на сервер.

Крок 7. Сервер на основі запиту від користувача робить запит до нейромережі.

Крок 8. Сервер отримує результати оціювання роботи від нейромережі.

Крок 9. Сервер перевіряє, чи успішно перевірено роботу студента.

Крок 10. Сервер записує результати перевірки до бази даних.

Крок 11. Сервер повертає результат перевірки роботи студента.

Крок 12. Користувачу буде показано модальне вікно з результатами перевірки роботи студента.

Крок 13. Перевірена робота студента буде додана то списку останніх перевірених робіт.

Крок 14. Відображення помилки користувачу, якщо не вдалося отримати відповідь від нейромережі.

2.5 Висновок

У цьому розділі було розроблено структуру програмного модуля перевірки коректності виконання студентських робіт, що дозволяє впорядкувати процес розробки програмного забезпечення та зробити його більш зрозумілим. Завдяки чітко визначеним компонентам та їх взаємодії між собою вдалося спростити процес розробки та реалізації програмного модуля.

Крім того, були створені UML-діаграми класів для серверної та клієнтської частини модуля, а також описані їхні методи. Це дозволяє наочно уявити архітектуру системи, взаємозв’язки між класами та їхню функціональність. UML-діаграми слугують важливою частиною документації проєкту, яка допомагає розробникам та іншим учасникам команди швидко зрозуміти внутрішню логіку програмного модуля.

Було розроблено ER-модель бази даних, що спрощує процес проєктування бази та забезпечує розуміння взаємозв’язків між її сутностями.

Також було розроблено схему алгоритму роботи програмного модуля перевірки коректності виконання студентських робіт, яка дозволяє детально проаналізувати покрокове виконання всіх процесів. Це значно спрощує розробку, знижує ймовірність помилок та допомагає покращити продуктивність програми. Чітко розроблений алгоритм дозволяє оптимізувати швидкість та ефективність виконання завдань.

Усі ці заходи є ключовими для ефективного процесу розробки, оскільки вони допомагають глибше зрозуміти логіку роботи системи, грамотно організувати робочий процес і виявити потенційні проблеми ще на початкових етапах розробки. Завдяки цьому підходу забезпечується успішна реалізація програмного продукту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ ПЕРЕВІРКИ КОРЕКТНОСТІ ВИКОНАННЯ СТУДЕНТСЬКИХ РОБІТ

3.1 Обґрунтування вибору мови та бібліотек програмування

Програмна реалізація модуля перевірки коректності виконання студентських робіт включає в собі клієнтський і серверний додатки. У цьому розділі буде розглянуто порівняння мов програмування та обґрунтування їх вибору окремо для кожного з цих додатків.

Для реалізації клієнтської частини програмного модуля перевірки коректності виконання студентських робіт доцільно розглянути дві мови програмування: JavaScript та TypeScript. Обидві широко використовуються для розробки WEB-застосунків і мають свої переваги та недоліки.

JavaScript – це мова програмування, що використовується для створення інтерактивного та динамічного контенту на WEB-сторінках. Вона є мовою з динамічною типізацією, яка виконується безпосередньо в браузері. JavaScript підтримує об'єктно-орієнтований, імперативний та функціональний підходи до програмування, що робить її універсальною для різноманітних задач, таких як обробка подій, робота з DOM та інтеграція з API [14].

TypeScript – це мова програмування, створена на базі JavaScript, яка додає до неї підтримку статичної типізації. Вона розроблена Microsoft і дозволяє розробникам виявляти помилки на етапі компіляції, що покращує надійність і масштабованість додатків. TypeScript компілюється у стандартний JavaScript, що робить його сумісним з усіма браузерами та платформами [15].

У таблиці 3.1 наведено основні відмінності між мовами програмування JavaScript та TypeScript.

Таблиця 3.1 – Порівняння мов програмування JavaScript та TypeScript

Параметр	JavaScript	TypeScript
Типізація	Динамічна, визначається під час виконання	Статична, визначається під час компіляції
Перевірка помилок	Виявляє помилки лише під час виконання	Помилки виявляються під час компіляції
Синтаксис	Простий, без додаткових конструкцій	Більш складний, підтримує інтерфейси, типи та класи
Навчання	Легше для початківців	Вимагає базового розуміння JavaScript
Інструменти	Не вимагає спеціальної конфігурації	Потребує налаштування компілятора (наприклад, tsconfig.json)
Використання	Безпосередньо виконується у браузері	Потрібна компіляція у JavaScript

Як видно з результатів порівняння у таблиці 3.1, TypeScript має перевагу над мовою програмування JavaScript, оскільки він пропонує переваги статичної типізації, що забезпечує більш високу надійність та полегшує підтримку коду в довгостроковій перспективі. Його використання дозволяє зменшити кількість помилок, покращити читаємість коду та спростити роботу у команді завдяки більш чітким типам даних. Крім того, TypeScript добре інтегрується з сучасними інструментами розробки та фреймворками, такими як Angular та React.

Для серверного додатку порівняємо мови програмування C#, Python та Scala.

C# – це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона працює на платформі .NET і є потужним інструментом для створення серверних додатків, WEB-додатків, хмарних сервісів та інших програмних рішень. C# поєднує в собі зручний синтаксис, підтримку асинхронного програмування та високий рівень продуктивності [16].

Python – це мова програмування з високим рівнем абстракції, відома своїм простим синтаксисом та універсальністю. Вона широко використовується у WEB-

розробці, аналізі даних, машинному навчанні та автоматизації процесів. Python забезпечує швидку розробку прототипів, однак має нижчу продуктивність порівняно з C# та Scala у багатопотокових задачах [17].

Scala – це функціонально-об’єктно-орієнтована мова програмування, яка працює на платформі Java Virtual Machine (JVM). Вона створена для розробки високопродуктивних систем і часто використовується в аналізі даних, обробці потоків даних у реальному часі та розподілених обчисленнях. Scala поєднує в собі функціональний підхід з можливостями Java, що робить її дуже гнучкою [18].

У таблиці 3.2 наведено основні відмінності між мовами програмування C#, Python та Scala.

Таблиця 3.2 – Порівняння мов програмування C#, Python та Scala

Параметр	C#	Python	Scala
Платформа	.NET	Незалежна, вимагає інтерпретатора	JVM
Типізація	Статична	Динамічна	Статична
Продуктивність	Висока	Середня	Висока
Складність навчання	Середня	Низька	Висока
Підтримка асинхронності	Потужна, вбудована в мову	Є, але менш ефективна	Є, з підтримкою через Futures та Akka
Сфера застосування	WEB-додатки, сервери, хмарні сервіси, десктопні програми	Прототипування, аналіз даних, автоматизація, машинне навчання	Обробка великих даних, високопродуктивні системи
Інструменти розробки	Потужна екосистема (Visual Studio, Rider)	Простий набір (PyCharm, Jupyter)	Розширений набір для JVM (IntelliJ IDEA)

Як видно з результатів порівняння у таблиці 3.2, мова програмування C# має низку переваг, які роблять її оптимальним вибором для розробки серверного

додатку. Завдяки високій продуктивності платформи .NET, C# забезпечує швидке виконання операцій, що є критично важливим для роботи з великим обсягом запитів або в режимі реального часу. Крім того, мова підтримує потужні вбудовані механізми асинхронного програмування, які дозволяють легко створювати масштабовані та ефективні серверні рішення. Розвинена екосистема інструментів розробки, таких як Visual Studio, сприяє швидкому створенню, тестуванню та підтримці додатків, а також надає широкі можливості для інтеграції з хмарними технологіями та мікросервісною архітектурою. Таким чином, для розробки серверного додатку була обрана мова програмування C#.

Для розробки клієнтської частини було використано бібліотеку React та фреймворк Material UI. React – це сучасна JavaScript-бібліотека для створення інтерфейсів користувача, яка дозволяє будувати динамічні, інтерактивні та масштабовані WEB-додатки за допомогою компонентної архітектури. Завдяки віртуальному DOM і високій продуктивності, React забезпечує швидку роботу інтерфейсу навіть при великій кількості користувачів. Material UI, у свою чергу, надає набір готових компонентів, створених відповідно до принципів Material Design від Google. Це дозволяє розробникам швидко створювати стильні та зручні інтерфейси, зменшуючи час на розробку дизайну і кастомізацію компонентів [19].

Для розробки серверної частини було використано ASP.NET, потужний фреймворк від Microsoft, який працює на платформі .NET. ASP.NET надає інструменти для створення високопродуктивних, надійних і безпечних WEB-додатків, WEB API та сервісів. Завдяки підтримці асинхронного програмування та вбудованим механізмам аутентифікації й авторизації, ASP.NET забезпечує стабільну роботу серверної частини навіть у випадках високих навантажень. Інтеграція з Visual Studio значно спрощує процес розробки, тестування та розгортання серверного додатку як у локальному середовищі, так і в хмарі [20].

3.2 Обґрунтування вибору мови програмування для управління організованою базою даних

Для управління організованою базою даних розглянемо дві мови: OQL (Object Query Language) та SQL (Structured Query Language). Ці мови широко використовуються для роботи з різними типами баз даних і мають свої особливості.

OQL (Object Query Language): призначена для роботи з об’єктно-орієнтованими базами даних. Вона підтримує запити до об’єктів, включаючи складні зв’язки між ними, використовуючи синтаксис, схожий на SQL. OQL інтегрується в об'єктно-орієнтовані мови програмування і дозволяє працювати з даними як із реальними об’єктами.

SQL (Structured Query Language) – це стандартна мова для роботи з реляційними базами даних. Вона забезпечує можливості для створення, оновлення, вилучення та маніпуляції даними. SQL є універсальною та підтримується майже всіма системами керування базами даних (СКБД), такими як MySQL, PostgreSQL, SQL Server та Oracle.

У таблиці 3.3 наведено порівняння основних характеристик OQL (Object Query Language) та SQL (Structured Query Language).

Таблиця 3.3 – Порівняння основних характеристик OQL та SQL мов програмування для управління організованою базою даних

Критерій	OQL	SQL
Тип бази даних	Об’єктно-орієнтовані	Реляційні
Підтримка стандартів	Відсутній єдиний стандарт	Має ISO та ANSI стандарти
Складність навчання	Вища через об'єктно-орієнтовану специфіку	Нижча завдяки зрозумілому синтаксису

Продовження таблиці 3.3

Критерій	OQL	SQL
Сумісність з мовами	Інтегрується з об’єктно-орієнтованими мовами, як C++, Java	Сумісна з усіма популярними мовами програмування
Поширеність	Обмежене використання	Широке використання у різних галузях
Ефективність роботи	Краще працює з об’єктними моделями	Краще працює з табличними структурами
Інструменти	Обмежений вибір інструментів для роботи	Широка підтримка інструментів та СКБД

Для управління організованою базою даних було обрано SQL, оскільки ця мова є універсальним стандартом для роботи з реляційними базами даних. SQL підтримується широким спектром систем керування базами даних (MySQL, PostgreSQL, SQL Server, Oracle), що забезпечує гнучкість і простоту інтеграції з існуючими рішеннями. SQL має зрозумілий синтаксис, що полегшує навчання і використання, а також надає потужні можливості для створення складних запитів, аналізу даних та їхньої обробки. Завдяки своїй поширеності, SQL має розвинену екосистему інструментів та документації, що дозволяє легко реалізовувати навіть складні проекти.

3.3 Обґрунтування вибору середовища розробки

Інтегроване середовище розробки (IDE) – це програмне забезпечення, яке об’єднує всі необхідні інструменти для розробки, тестування та підтримки програмного забезпечення в єдиному інтерфейсі. IDE значно спрощує процес розробки, забезпечуючи підсвічування синтаксису, автодоповнення коду, налагодження, інтеграцію з системами контролю версій та інші функції.

Під час вибору середовища розробки для клієнтської та серверної частин проєкту було розглянуто такі IDE: Visual Studio Code (VS Code), WebStorm, Visual

Studio та Rider. Кожне з них має свої особливості та переваги, які важливі для різних етапів розробки.

Visual Studio Code (VS Code) – це безкоштовний текстовий редактор з розширеною функціональністю, розроблений Microsoft. Він підтримує підсвічування синтаксису, автодоповнення, інтеграцію з Git та велику кількість розширень, які дозволяють працювати з різними мовами програмування і технологіями. VS Code особливо популярний серед розробників завдяки легкості використання та низьким системним вимогам [21].

WebStorm – це потужне IDE для розробки JavaScript-додатків, розроблене компанією JetBrains. WebStorm пропонує глибоку інтеграцію з фреймворками та бібліотеками JavaScript, такими як React, Angular та Vue.js, а також зручне управління npm-скриптами та інтеграцію з системами контролю версій. Воно також надає інструменти для тестування та налагодження клієнтської частини проєктів [22].

Visual Studio – це повноцінне середовище розробки від Microsoft, яке підтримує широкий спектр мов програмування, зокрема C#, Python, C++, та інші. Visual Studio надає вбудовані інструменти для налагодження, роботи з базами даних, тестування та створення проєктів для різних платформ, включаючи десктоп, WEB та хмару [23].

Rider – це кросплатформне IDE для .NET-розробки, створене компанією JetBrains. Rider підтримує роботу з C#, ASP.NET, Unity та іншими технологіями, а також інтегрується з платформою .NET Core. Завдяки потужному аналізу коду, швидкому рефакторингу та зручним інструментам для роботи з тестами та базами даних, Rider є одним з найкращих інструментів для серверної розробки [24].

У таблиці 3.4 наведено порівняння основних характеристик Visual Studio Code (VS Code), WebStorm, Visual Studio та Rider.

Таблиця 3.4 – Порівняння середовищ розробки Visual Studio Code (VS Code), WebStorm, Visual Studio та Rider

Параметр	VS Code	WebStorm	Visual Studio	Rider
Призначення	Загальне використання	WEB-розробка	Розробка для всіх платформ	.NET-розробка
Платформи	Windows, macOS, Linux	Windows, macOS, Linux	Windows	Windows, macOS, Linux
Підтримка мов	Багато мов, через розширення	Основний акцент на JavaScript	Широкий набір мов	Переважно .NET та Java
Функціональність	Легка, мінімалістична	Потужна для WEB-додатків	Потужна для різних типів додатків	Потужна для .NET-додатків
Ціна	Безкоштовно	Безкоштовна Community-версія, платна Professional/Enterprise	Безкоштовна Community-версія, платна Professional/Enterprise	Безкоштовна Community-версія, платна Professional/ Enterprise
Складність навчання	Легке	Середнє	Середнє	Середнє
Інтеграція з Git	Хороша	Вбудована	Вбудована	Вбудована
Продуктивність	Висока, легке середовище	Середня	Висока	Висока

Для реалізації клієнтської частини було обрано WebStorm, оскільки воно надає потужні інструменти для роботи з React і підтримує сучасні WEB-технології, забезпечуючи ефективність розробки. Для серверної частини вибрано Rider, оскільки це середовище є прийнятним для роботи з C# і платформою .NET завдяки глибокому аналізу коду, зручному рефакторингу та підтримці тестування. Таким чином, для створення ефективного та зручного процесу розробки будуть використовуватися WebStorm для клієнтської частини та Rider для серверної.

3.4 Розробка клієнтської та серверної частин

Для розробки програмного модуля перевірки коректності виконання студентських робіт використано бібліотеку React та фреймворк ASP.NET.

ASP.NET – це потужний WEB-фреймворк від Microsoft, який забезпечує створення сучасних WEB-додатків, API та мікросервісів. Завдяки підтримці WebSockets, ASP.NET дозволяє реалізовувати двостороннє спілкування між клієнтом і сервером у реальному часі, що особливо корисно для чатів, онлайн-ігор та фінансових додатків. Також фреймворк підтримує gRPC, який забезпечує високопродуктивну міжпроцесну комунікацію з мінімальною затримкою, що робить його ідеальним для мікросервісної архітектури. Крім цього, ASP.NET має можливість інтеграції з базами даних за допомогою Entity Framework (EF), що дозволяє працювати з базою даних на основі об’єктно-реляційного відображення (ORM), забезпечуючи зручну роботу з даними, автоматичне створення запитів та підтримку складних реляційних структур [25].

Для взаємодії з базою даних використовується Entity Framework, але для того, щоб коректно взаємодіяти з нею потрібно спочатку створити класи об’єктів, які Entity Framework конвертує в таблиці бази даних [20]. У розроблювальному ресурсі буде три класи:

- CompletionSettingsEntity – цей клас відповідає за створення, редагування та отримання налаштувань запитів;
- LabEntity – цей клас відповідає за створення, редагування та отримання студентських робіт;
- ResultEntity – цей клас відповідає за збереження та отримання результатів оцінення студентських робіт.

Також створено класи для налаштування зв’язків між сутностями: LabEntityConfiguration, який містить в собі конфігурацію LabEntity, CompletionSettingsEntityConfiguration, який містить в собі конфігурацію

CompletionSettingsEntity, ResultEntityConfiguration який містить в собі конфігурацію ResultEntity.

Діаграма діяльності класу LabEntityConfiguration зображена на рисунку 3.1.

Рисунок 3.1 – Діаграма діяльності класу LabEntityConfiguration

Реалізація класу LabEntityConfiguration наведена нижче:

```
public class LabEntityConfiguration : IEntityTypeConfiguration<LabEntity>
{
   public void Configure(EntityTypeBuilder<LabEntity> builder)
   {
      builder.HasOne(p => p.CompletionSettings)
         .WithOne(p => p.Lab)
         .HasForeignKey<LabEntity>(p => p.CompletionSettingsId);
```

```
    }
}
```

Діаграма діяльності класу CompletionSettingsEntityConfiguration зображена на рисунку 3.2.

Рисунок 3.2 – Діаграма діяльності класу CompletionSettingsEntityConfiguration

Реалізація класу CompletionSettingsEntityConfiguration наведена нижче:

```
public class CompletionSettingsEntityConfiguration :
IEntityTypeConfiguration<CompletionSettingsEntity>
{
    public void Configure(EntityTypeBuilder<CompletionSettingsEntity> builder)
    {
        builder.HasOne(p => p.Lab)
            .WithOne(p => p.CompletionSettings)
            .HasForeignKey<CompletionSettingsEntity>(p => p.LabId);
```

```
    }
}
```

Діаграма діяльності класу ResultEntityConfiguration зображена на рисунку 3.3.

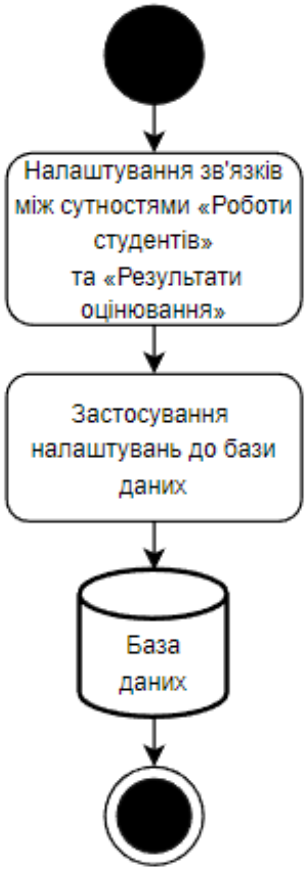

Рисунок 3.3 – Діаграма діяльності класу ResultEntityConfiguration

Реалізація класу ResultEntityConfiguration наведена нижче:

```
public class ResultEntityConfiguration : IEntityTypeConfiguration<ResultEntity>
{
    public void Configure(EntityTypeBuilder<ResultEntity> builder)
    {
        builder.HasOne(p => p.Lab)
            .WithMany(p => p.Results)
            .HasForeignKey(p => p.LabId);
```

```
    }
}
```

Створено LabController, який призначений для забезпечення взаємодії між клієнтським додатком і сервером у рамках перевірки коректності виконання студентських робіт. Він дозволяє клієнту завантажувати файли робіт для аналізу із зазначенням параметрів перевірки, а також отримувати результати аналізу, включаючи оцінки, коментарі, висновки та детальну інформацію про використані налаштування перевірки.

Діаграма діяльності класу LabController зображена на рисунку 3.4.

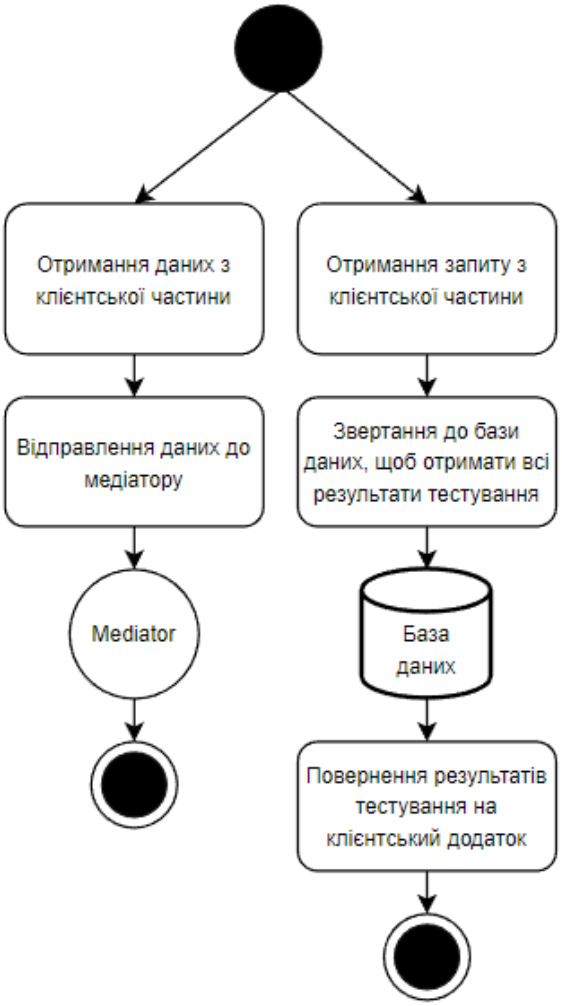

Рисунок 3.4 – Діаграма діяльності класу LabController

Реалізація класу LabController наведена нижче:

```
using Assessor.Server.Api.Controllers.Abstract;
using Assessor.Server.Api.Extensions;
using Assessor.Server.Application.Lab.Commands.MarkLab;
using Assessor.Server.Domain.Contexts;
using MediatR;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace Assessor.Server.Api.Controllers;
public class LabController : ApiControllerBase
{
    private readonly AssessorContext _context;
    public LabController(IMediator mediator, AssessorContext context) : base(mediator)
    {
        _context = context;
    }
    [HttpPost("[action]")]
    public async Task<IActionResult> MarkLab(
        [FromForm] IFormFile formFile,
        [FromQuery] double temperature,
        [FromQuery] int maxTokens,
        [FromQuery] double topP,
        [FromQuery] int? frequencyPenalty,
        [FromQuery] int? presencePenalty,
        CancellationToken cancellationToken)
    {
        var markLabCommand = new MarkLabCommand(formFile, temperature, maxTokens,
topP, frequencyPenalty, presencePenalty);
        var result = await Mediator.Send(markLabCommand, cancellationToken);
        return result.GetActionResult();
```

```
}
[HttpGet]
public async Task<IActionResult> GetResult()
{
    var results = await _context.Results
        .AsNoTracking()
        .Select(result => new
        {
            result.Id,
            result.Mark,
            result.Remark,
            result.Conclusion,
            result.CreatedAt,
            result.UpdatedAt,
            Lab = new
            {
                result.Lab.Id,
                result.Lab.Name,
                result.Lab.Extension,
                CompletionSettings = new
                {
                    result.Lab.CompletionSettings.Id,
                    result.Lab.CompletionSettings.Temperature,
                    result.Lab.CompletionSettings.MaxTokens,
                    result.Lab.CompletionSettings.TopP,
                    result.Lab.CompletionSettings.FrequencyPenalty,
                    result.Lab.CompletionSettings.PresencePenalty,
                    result.Lab.CompletionSettings.CreatedAt,
                    result.Lab.CompletionSettings.UpdatedAt
                }
```

```
            }
        })
        .ToListAsync();
    return Ok(results);
  }
}
```

Також створено класи для читання PDF та DOC файлів з назвами PdfFileReaderService та DocFileReaderService відповідно.

Діаграма діяльності класу PdfFileReaderService зображена на рисунку 3.5.

Рисунок 3.5 – Діаграма діяльності класу PdfFileReaderService

Реалізація класу PdfFileReaderService має такий зміст:

```
using System.Net;
using System.Text;
using Assessor.Server.Application.Common.Interfaces;
using Assessor.Server.Domain.Extensions;
using Assessor.Server.Domain.Models;
using iText.Kernel.Pdf;
using iText.Kernel.Pdf.Canvas.Parser;
using Microsoft.AspNetCore.Http;
namespace Assessor.Server.Infrastructure.Services;
public class PdfFileReaderService : IFileReaderService
{
    public async Task<Result<string, Error>> ReadTextFromFile(IFormFile file)
    {
        if (!file.IsPdfFile())
        {
            return new Error(HttpStatusCode.BadRequest, "Invalid file extension");
        }
        using var memoryStream = new MemoryStream();
        await file.CopyToAsync(memoryStream);
        memoryStream.Position = 0;
        var stringBuilder = new StringBuilder();
        using var pdfReader = new PdfReader(memoryStream);
        using var pdfDocument = new PdfDocument(pdfReader);
        for (var i = 1; i <= pdfDocument.GetNumberOfPages(); i++)
        {
            var text = PdfTextExtractor.GetTextFromPage(pdfDocument.GetPage(i));
            stringBuilder.Append(text);
        }
        return stringBuilder.ToString();
```

```
    }
}
```

Діаграма діяльності класу DocFileReaderService зображена на рисунку 3.6.

Рисунок 3.6 – Діаграма діяльності класу DocFileReaderService

Реалізація класу DocFileReaderService має такий зміст:

```
using System.Net;
using System.Text;
using Assessor.Server.Application.Common.Interfaces;
using Assessor.Server.Domain.Extensions;
```

```
using Assessor.Server.Domain.Models;
using Microsoft.AspNetCore.Http;
using NPOI.XWPF.UserModel;
namespace Assessor.Server.Infrastructure.Services;
public class DocFileReaderService : IFileReaderService
{
   public async Task<Result<string, Error>> ReadTextFromFile(IFormFile file)
   {
      if (!file.IsDocFile())
      {
         return new Error(HttpStatusCode.BadRequest, "Invalid file extension");
      }
      using var memoryStream = new MemoryStream();
      await file.CopyToAsync(memoryStream);
      memoryStream.Position = 0;
      var stringBuilder = new StringBuilder();
      using var document = new XWPFDocument(memoryStream);
      foreach (var paragraph in document.Paragraphs)
      {
         stringBuilder.AppendLine(paragraph.Text);
      }
      return stringBuilder.ToString();
   }
}
```

Далі наведено клас MarkLabHandler за допомогою якого відбувається оцінювання роботи студентів.

Діаграма діяльності класу MarkLabHandler зображена на рисунку 3.7.

Рисунок 3.7 – Діаграма діяльності класу MarkLabHandler

Реалізація класу MarkLabHandler має такий зміст:

```
using System.Net;
using Assessor.Server.Application.Common.Interfaces;
using Assessor.Server.Application.Extensions;
using Assessor.Server.Domain.Contexts;
using Assessor.Server.Domain.Entities;
using Assessor.Server.Domain.Enums;
using Assessor.Server.Domain.Extensions;
using Assessor.Server.Domain.Models;
using MediatR;
using Microsoft.Extensions.DependencyInjection;
namespace Assessor.Server.Application.Lab.Commands.MarkLab;
public class MarkLabHandler : IRequestHandler<MarkLabCommand, Result<Content,
Error>>
{
   private readonly IAiHttpClient _aiHttpClient;
   private readonly IServiceProvider _serviceProvider;
   private readonly AssessorContext _context;
   public MarkLabHandler(IAiHttpClient aiHttpClient, IServiceProvider serviceProvider,
AssessorContext context)
   {
      _aiHttpClient = aiHttpClient;
      _serviceProvider = serviceProvider;
      _context = context;
   }
   public async Task<Result<Content, Error>> Handle(MarkLabCommand request,
CancellationToken cancellationToken)
   {
      var fileReaderType = request.LabFile.GetFileReaderType();
      if (fileReaderType is FileReaderTypes.Unknown)
```

```
        {
            return new Error(HttpStatusCode.BadRequest, "Invalid file type");
        }
        var scope = _serviceProvider.CreateScope();
        var fileReaderService =
scope.ServiceProvider.GetRequiredKeyedService<IFileReaderService>(fileReaderType);
        var readTextResult = await fileReaderService.ReadTextFromFile(request.LabFile);
        if (!readTextResult.IsSuccess)
        {
            return readTextResult.Error;
        }
        var completionSettingsModel = request.MapToCompletionSettings();
        var result = await _aiHttpClient.GetCompletion(readTextResult.Value,
completionSettingsModel);
        if (result.IsSuccess)
        {
            var resultEntity = new ResultEntity
            {
                Mark = result.Value.Mark,
                Remark = result.Value.Remark,
                Conclusion = result.Value.Conclusion,
                Lab = new LabEntity
                {
                    Name = request.LabFile.FileName,
                    Extension = fileReaderType.ToString(),
                    Data = await request.LabFile.ConvertToBase64WithPrefixAsync(),
                    CompletionSettings = new CompletionSettingsEntity
                    {
                        Temperature = request.Temperature,
                        MaxTokens = request.MaxTokens,
```

```
                TopP = request.TopP,
                FrequencyPenalty = request.FrequencyPenalty,
                PresencePenalty = request.PresencePenalty,
            }
        }
    };
    _context.Results.Add(resultEntity);
    var rowsAffected = await _context.SaveChangesAsync(cancellationToken);
    if (rowsAffected is 0)
    {
        return new Error(HttpStatusCode.BadRequest, "No rows affected");
    }
  }
  return result;
 }
}
```

3.5 Тестування роботи програми та аналіз результатів

Тестування програмного забезпечення є важливим етапом у процесі розробки, який спрямований на забезпечення якості, надійності, безпеки та продуктивності продукту. Воно дозволяє мінімізувати витрати на виправлення помилок, підвищити задоволеність користувачів і забезпечити відповідність програмного забезпечення встановленим вимогам та специфікаціям.

Розроблений WEB-застосунок був ретельно протестований, із проведенням понад 100 тестових запусків. Після запуску додатку з’являється головна сторінка, яка зображена на рисунку 3.8.

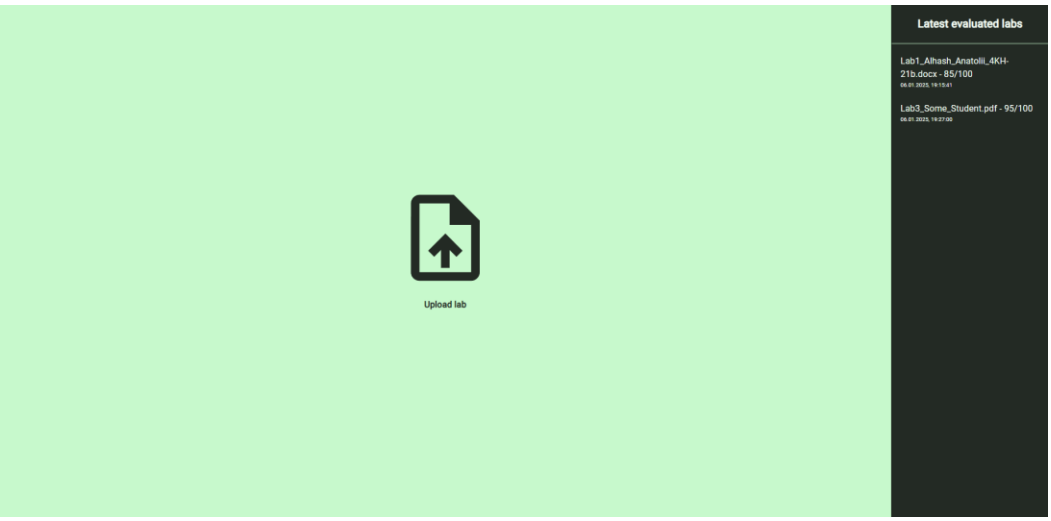

Рисунок 3.8 – Загальний вигляд інтерфейсу головної сторінки розробки

Якщо натиснути на іконку для завантаження файлу, то з’явиться вікно з вибором роботи студента. Вигляд цього вікна зображено на рисунку 3.9.

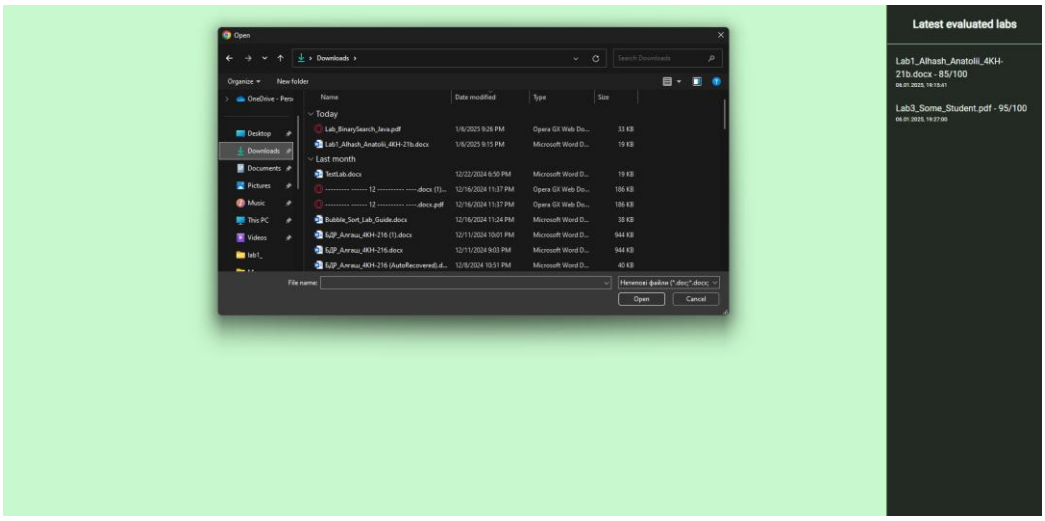

Рисунок 3.9 – Загальний вигляд інтерфейсу вікна з вибором роботи студента

Після успішного вибору файлу роботи студента, користувачу застосунка буде показано параметри, за допомогою яких він може коригувати параметри модуля для перевірки коректності виконання студентських робіт. Ці параметри показані на рисунку 3.10.

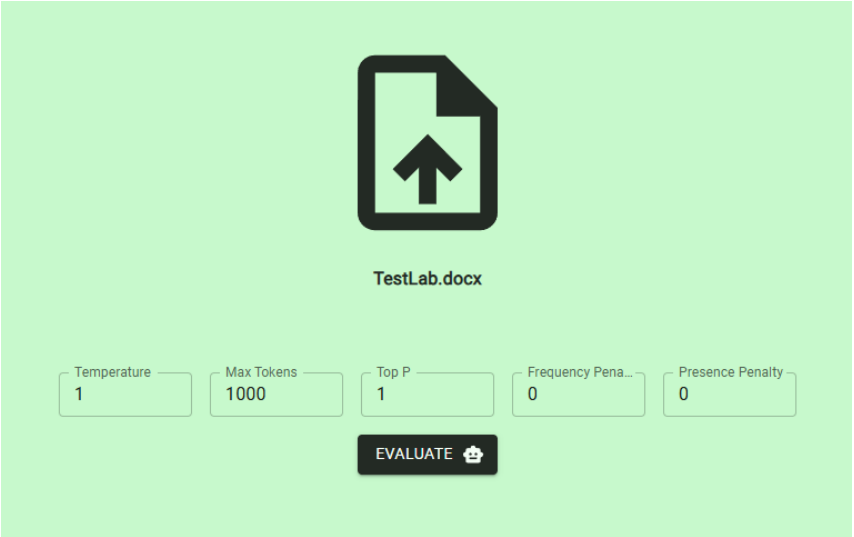

Рисунок 3.10 – Параметри модуля перевірки коректності виконання студентських робіт

Потім користувач може натиснути кнопку «EVALUATE», вказані налаштування з роботою студента будуть відправлені на сервер програмного модуля перевірки коректності виконання студентських робіт. Після цього сервер дасть відповідь із результатами виконання роботи студентом. Загальний вигляд інтерфейсу вікна з результатами виконання роботи студента зображено на рисунку 3.11.

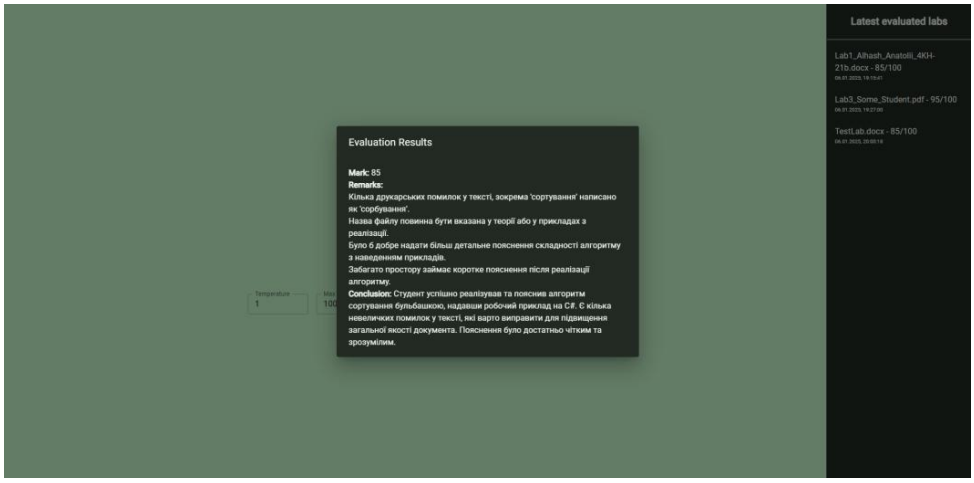

Рисунок 3.11 – Загальний вигляд інтерфейсу вікна з результатами виконання роботи студента

Після успішної перевірки коректності виконання студентської роботи, оцінена робота з’явиться у правому вікні, де зображені останні перевірені роботи та їх оцінки. Це вікно зображено на рисунку 3.12.

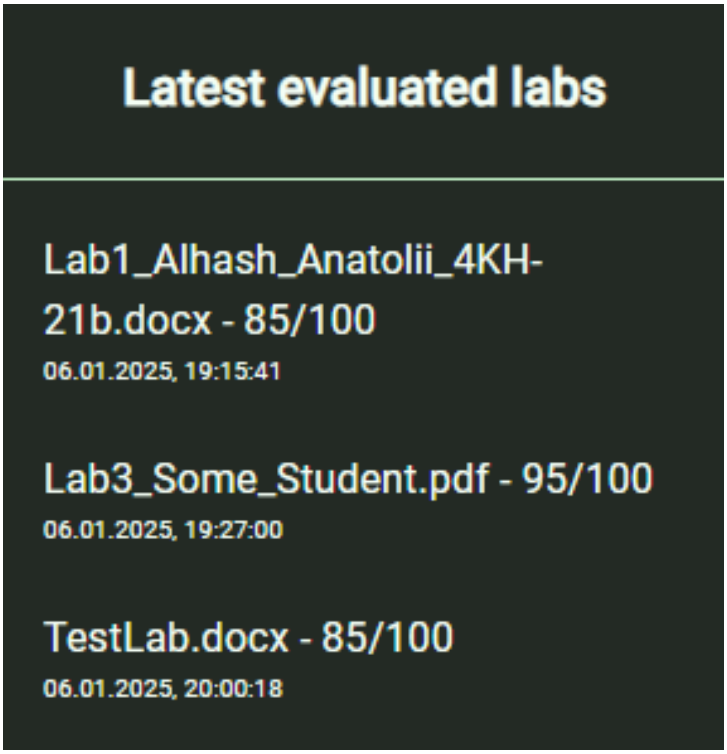

Рисунок 3.12 – Загальний вигляд інтерфейсу вікна із зображенням перевірки останніх студентських робіт та їх оцінки

Розроблений програмний модуль перевірки коректності виконання студентських робіт успішно пройшов тестування та відповідає всім заданим вимогам.

У таблиці 3.5 подано результати тестування розробленого програмного продукту та аналогів.

В результаті розробки було реалізовано функціонал, який був відсутній в системах-аналогах, а саме:

- локалізація усіх мов світу;
- підтримка будь-якого формату файлів робіт студентів;

- глибокий аналіз коректності виконання роботи студента;
- гнучке налаштування параметрів модуля.

Таблиця 3.5 – Результати тестування розробленого програмного продукту та аналогів

Характеристика	Grammarly	Codio	SonarQube	Розроблений модуль
Основна функція	Перевірка тексту англійською мовою	Автоматизоване тестування програмування	Статичний аналіз коду	Перевірка коректності студентських робіт
Локалізація	Тільки англійська	Обмежена підтримка	Обмежена підтримка	Локалізація усіх мов світу
Формати файлів	Текстові файли	Підтримка обмеженої кількості форматів	Код програмування	Підтримка будь-якого формату файлів
Глибокий аналіз	Стилістичні та граматичні рекомендації	Аналіз коду, але обмежений автоматизацією	Статичний аналіз якості коду	Глибокий аналіз коректності виконання роботи
Гнучкість налаштувань	Мінімальні налаштування	Гнучкість у межах налаштувань оцінювання коду	Складні налаштування для аналізу	Гнучке налаштування параметрів модуля
Підтримка мов програмування	Не підтримується	Обмежена кількість мов	Понад 25 мов	Підтримка всіх мов, необхідних для студентських робіт
Продуктивність	Висока для текстових перевірок	Середня, залежить від складності завдань	Висока, але залежить від розміру проекту	Висока для всіх типів завдань
Автоматизація	Виправлення стилю та граматики	Тестування програмного коду	Автоматичний аналіз коду	Повністю автоматизована перевірка будь-яких завдань

Продовження таблиці 3.5

Характеристика	Grammarly	Codio	SonarQube	Розроблений модуль
Ресурсомісткість	Низька	Середня	Висока	Оптимізована для роботи на стандартному обладнанні
Додаткові можливості	Рекомендації щодо стилю	Інтерактивне навчання	Метрики якості, оцінка технічного боргу	Інноваційні функції відсутні в аналогах

Особливістю розробленого програмного модуля перевірки коректності виконання студентських робіт є його висока швидкість роботи, точність оцінювання та можливість надання розгорнутого висновку щодо виконання роботи. На рисунках 3.13 та 3.14 представлено аналіз швидкості перевірки та якості оцінювання студентських робіт порівняно з аналогічними системами, такими як Grammarly, Codio та SonarQube. Аналіз було проведено з використанням Apache JMeter та K6, що дозволило отримати об'єктивні результати щодо ефективності роботи системи [26, 27].

На рисунку 3.13 наведено графік, який ілюструє залежність швидкості перевірки студентських робіт від їх кількості.

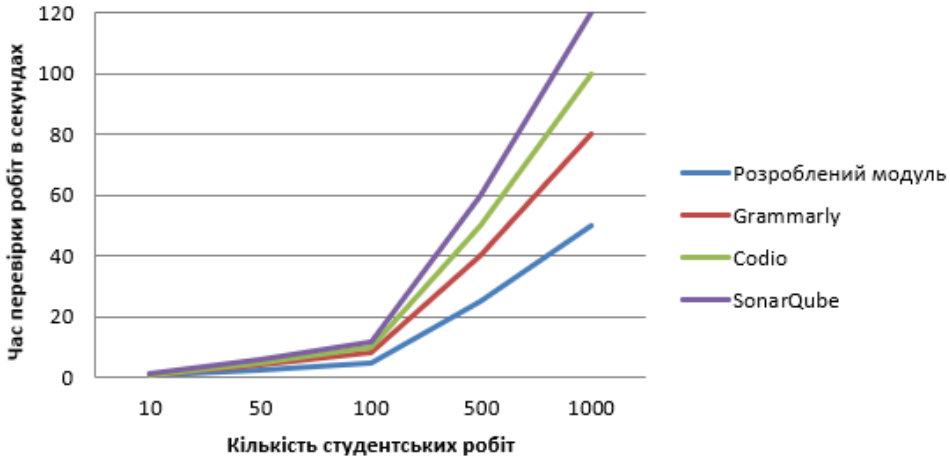

Рисунок 3.13 – Графік швидкості перевірки коректності виконання студентських робіт

З рис. 3.13 видно, що розроблений програмний модуль демонструє найвищу продуктивність, особливо за великої кількості робіт. Наприклад, при 1000 роботах час перевірки розробки становить лише 50 секунд, тоді як для Grammarly – 80 секунд, Codio – 100 секунд, а SonarQube – 120 секунд. Це свідчить про оптимізацію алгоритмів модуля, що дозволяє значно скоротити час обробки даних порівняно з системами-аналогами.

На рисунку 3.14 представлено порівняння якості оцінювання студентських робіт за трьома параметрами: глибина аналізу, точність оцінки, детальний висновок. Розроблений модуль отримав найвищі оцінки за всіма параметрами. Глибина аналізу розробки становить 9.5 балів і перевищує Grammarly (7 балів), Codio (8 балів), SonarQube (8.5 балів). Точність оцінки розробленого програмного продукту досягає 9.8 балів, що значно краще за Grammarly (8 балів) та SonarQube (8 балів), Codio (7.5 балів). Детальний висновок розробки оцінений у 9.7 балів і перевищує показники Grammarly (7.5 балів), Codio (8.2 балів) і SonarQube (8.5 балів).

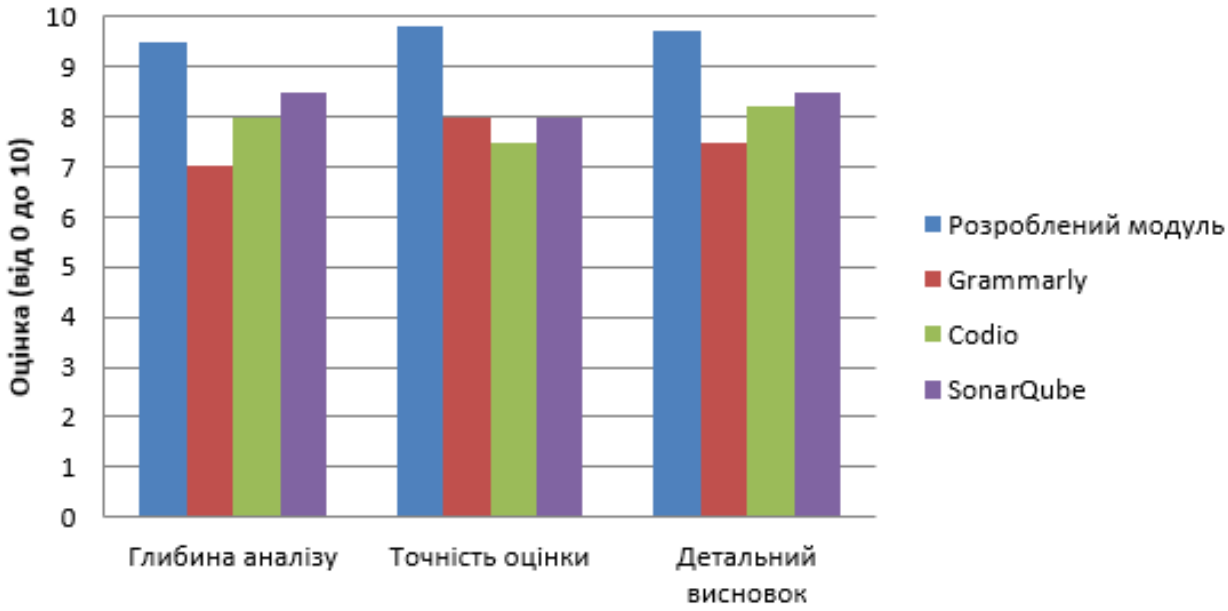

Рисунок 3.14 – Графік якості оцінки студентських робіт

Таким чином, результати аналізу та тестування розробленого програмного модуля підтвердили досягнення поставлених цілей. Як видно з таблиці 3.5, рисунка 3.13 та рисунка 3.14, програмний модуль демонструє покращений функціонал, високу швидкість перевірки та якість оцінювання робіт порівняно з аналогами. Це забезпечує підвищення ефективності та задоволеності користувачів при використанні такого програмного продукту.

3.6 Висновок

У цьому розділі було проведено аналіз мов програмування для розробки клієнтської частини (JavaScript, TypeScript) та серверної частини (C#, Python, Scala). Було розглянуто їх переваги, недоліки, особливості та доцільність використання. За результатами аналізу для розробки клієнтської частини обрано TypeScript, оскільки він забезпечує статичну типізацію, що підвищує надійність та масштабованість коду. Для розробки серверної частини обрано C#, завдяки його високій продуктивності, зручній підтримці асинхронного програмування та потужній екосистемі для розробки сучасних WEB-застосунків.

Також було виконано аналіз мов програмування для управління організованою базою даних – SQL та OQL. У результаті аналізу обрано SQL, оскільки ця мова є стандартом для роботи з реляційними базами даних, підтримується широким спектром систем керування базами та забезпечує зручність у створенні складних запитів.

Для середовища розробки клієнтської частини обрано WebStorm, оскільки воно пропонує розширені інструменти для роботи з сучасними фреймворками, такими як React, забезпечуючи ефективність розробки. Для серверної частини обрано Rider, який є найкращим середовищем для розробки на платформі .NET завдяки потужному аналізу коду, зручному рефакторингу та інтеграції з платформою.

Також реалізовано програмний модуль перевірки коректності виконання студентських робіт.

Виконано тестування програмного модуля перевірки коректності виконання студентських робіт та здійснено його порівняння з аналогічними системами. У результаті було встановлено, що всі основні функції, передбачені для реалізації, успішно впроваджені. Розроблений програмний продукт демонструє покращений функціонал порівняно з аналогами на 4 можливості, тобто: локалізацію усіх мов світу, підтримку будь-якого формату файлів робіт студентів, глибокий аналіз коректності виконання роботи студента та гнучке налаштування параметрів модуля. Завдяки цьому програмний модуль забезпечує високу швидкість перевірки та якість оцінювання у порівнянні з існуючими системами-аналогами.

ВИСНОВКИ

Всі задачі, поставлені для реалізації програмного модуля перевірки коректності виконання студентських робіт були виконані в повному обсязі, а саме: обґрунтовано доцільність розробки програмного модуля перевірки коректності виконання студентських робіт; спроектовано програмний модуль перевірки коректності виконання студентських робіт; програмно реалізовано програмний модуль перевірки коректності виконання студентських робіт; виконано тестування програми та проведено аналіз отриманих результатів; розроблено інструкцію користувача.

Для реалізації програмного модуля перевірки коректності виконання студентських робіт були використані сучасні технології та мови програмування. Серверна частина створена на базі C# та ASP.NET, це забезпечує стабільність, продуктивність та можливість масштабування. Клієнтська частина розроблена за допомогою TypeScript та React, що гарантує гнучкість, інтерактивність і зручний інтерфейс. Для роботи з базою даних використовувалася SQL, що забезпечує ефективне управління збереженими даними. Як середовище виконання обрано WEB-браузери, це дозволяє використовувати програмний модуль без необхідності встановлення додаткового програмного забезпечення.

Під час роботи була розроблена структурна модель програмного модуля перевірки коректності виконання студентських робіт, яка містить UML-діаграми класів для серверної та клієнтської частин, ER-модель бази даних, а також схему алгоритму роботи. Це дозволило систематизувати процес розробки, оптимізувати взаємодію між компонентами і спростити подальшу підтримку та розвиток програмного продукту.

Під час тестування програмний модуль перевірки коректності виконання студентських робіт продемонстрував високу ефективність та переваги перед аналогічними системами. У порівнянні з існуючими рішеннями, він надає

користувачам щонайменше 4 додаткові можливості, це покращує якість та зручність його використання. Крім того, програмний модуль забезпечує автоматизовану перевірку студентських робіт, що значно зменшує навантаження на викладачів та забезпечує об’єктивність оцінювання. В розробці реалізовано вікно з переглядом останніх перевірених робіт, яке дозволяє швидко отримати доступ до історії перевірок та переглянути детальні результати аналізу, а також вікно для налаштування параметрів нейромережі, що дає змогу гнучко адаптувати процес перевірки відповідно до конкретних вимог і покращує точність оцінювання.

Мета роботи, яка полягала в розширенні функціональних можливостей програмного модуля перевірки коректності виконання студентських робіт досягнута за рахунок введення таких 4 додаткових можливостей: локалізації усіх мов світу, що дозволяє перевіряти роботи незалежно від мови їх написання; підтримки будь-якого формату файлів студентських робіт, що забезпечує універсальність використання; глибокого аналізу коректності виконання роботи студента, який сприяє більш точному оцінюванню результатів; гнучкого налаштування параметрів модуля, що дає можливість адаптувати процес перевірки відповідно до вимог викладачів.

Робота виконана у повному обсязі відповідно до поставлених цілей, вимог та методичних рекомендацій щодо підготовки бакалаврських кваліфікаційних робіт. Усі етапи були реалізовані належним чином, що підтверджує якість, завершеність і практичну цінність розробленого рішення [28].

Отже, програмний модуль перевірки коректності виконання студентських робіт відповідає заданим вимогам, забезпечує високу продуктивність, точність оцінювання та розширені можливості налаштування. Він є ефективним інструментом для автоматизації перевірки студентських робіт, що значно спрощує процес оцінювання, забезпечує об’єктивність та зменшує витрати часу викладачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгаш А. М., Крилик Л. В. Аналіз передумов розробки програмного модуля перевірки коректності виконання студентських робіт. *LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23045 (дата звернення: 04.05.2025).

2. Крилик Л. В., Алгаш А. М. Розробка програмного модуля перевірки коректності виконання студентських робіт. Вісник Хмельницького національного університету. Серія «Технічні науки». 2025. №2(349). С. 198–205.

3. Свідоцтво про реєстрацію авторського права на твір № 134193. Комп'ютерна програма «Програмний модуль перевірки коректності виконання студентських робіт» / Крилик Л. В., Алгаш А. М., дата реєстрації 10.03.2025 р.

4. Grammarly. URL: https://www.grammarly.com (дата звернення: 04.05.2025).

5. Codio. URL: https://www.codio.com (дата звернення: 04.05.2025).

6. SonarCube. URL: https://www.sonarsource.com (дата звернення: 04.05.2025).

7. Річардсон К. Мікросервіси. Патерни розробки та рефакторингу / Пер. з англ. Київ: Видавництво «Діалектика», 2019. 432 с.

8. Григорович В. Г. Об'єктно-орієнтоване програмування. Частина 1. Львів: Магнолія 2006, 2023. 284 с.

9. Як використовувати наслідування. URL: https://gamedev.dou.ua/blogs/inheritance-in-object-oriented-programming (дата звернення: 04.05.2025).

10. Поліморфізм у мовах програмування: види та приклади, як використовувати в коді. URL: https://highload.tech/uk/polymorphism (дата звернення: 04.05.2025).

11. Гришанович Т. О., Глинчук Л. Я. Основи об'єктно-орієнтованого програмування. Луцьк: Волинський національний університет імені Лесі Українки, 2022. 120 с.

12. Остапченко К. Б. Бази даних: навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2022. 152 с.

13. Савчук Т. О., Ваховська Л. М. Основи інформатики та обчислювальної техніки. Навчальні посібники Вінницького національного технічного університету. Мережні електронні видання. URL: https://web.posibnyky.vntu.edu.ua/fitki/1savchuk_osnovy_inform_obchisl_tehn/Topic5.html (дата звернення: 04.05.2025).

14. Фленаган Д. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. Київ: O'Райлі Медіа, 2020. 706 с.

15. Вандеркам Д. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. Київ: O'Райлі Медіа, 2019. 261 с.

16. Прайс М. C# 10 і .NET 6. Сучасна кросплатформенна розробка. 6-те видання. Київ: Видавництво «Діалектика», 2023. 820 с.

17. Васильєв О. Програмування мовою Python. Тернопіль: Навчальна книга–Богдан, 2019. 320 с.

18. Осійчук І. В. Особливості функційного програмування на Scala // *Матеріали VI Міжнародної студентської науково-технічної конференції «Природничі та гуманітарні науки. Актуальні питання»*, 27–28 квітня 2023 р. Тернопіль: ТНТУ, 2023. С. 163–164.

19. Вієруч Р. The Road to React: Your journey to master plain yet pragmatic React.js. Київ: Видавництво «Діалектика», 2022. 200 с.

20. Нейлор Лі. ASP.NET MVC з Entity Framework та CSS. Нью-Йорк: Apress, 2016. 608 с.

21. Visual Studio Code: Microsoft. Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs (дата звернення: 04.05.2025).

22. WebStorm: JetBrains. WebStorm: The Smartest JavaScript IDE. URL: https://www.jetbrains.com/webstorm/ (дата звернення: 04.05.2025).

23. Visual Studio: Microsoft. Visual Studio IDE – Developer Tools for Software Developers and Teams. URL: https://visualstudio.microsoft.com/ (дата звернення: 04.05.2025).

24. Rider: JetBrains. Rider: The Cross-Platform .NET IDE. URL: https://www.jetbrains.com/rider/ (дата звернення: 04.05.2025).

25. Сміт Джон П. Entity Framework Core в дії. Нью-Йорк: Manning Publications, 2018. 568 с.

26. Apache JMeter. URL: https://jmeter.apache.org/ (дата звернення: 04.05.2025).

27. K6. URL: https://k6.io/ (дата звернення: 04.05.2025).

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця : ВНТУ, 2023. (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль перевірки коректності виконання студентських робіт

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,00 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Крилик Л.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Алгаш А.М.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

Лістинг програми

```
using Asp.Versioning;
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace Assessor.Server.Api.Controllers.Abstract;

[ApiController]
[Route("api/v{version:apiVersion}")]
[ApiVersion("1.0")]
public abstract class ApiControllerBase : ControllerBase
{
    protected readonly IMediator Mediator;

    protected ApiControllerBase(IMediator mediator)
    {
        Mediator = mediator;
    }
}

using Assessor.Server.Api.Controllers.Abstract;
using Assessor.Server.Api.Extensions;
using Assessor.Server.Application.Lab.Commands.MarkLab;
using Assessor.Server.Domain.Contexts;
using MediatR;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Assessor.Server.Api.Controllers;

public class LabController : ApiControllerBase
{
    private readonly AssessorContext _context;
    public LabController(IMediator mediator, AssessorContext context) : base(mediator)
    {
        _context = context;
    }
```

```
    [HttpPost("[action]")]
    public async Task<IActionResult> MarkLab(
        [FromForm] IFormFile formFile,
        [FromQuery] double temperature,
        [FromQuery] int maxTokens,
        [FromQuery] double topP,
        [FromQuery] int? frequencyPenalty,
        [FromQuery] int? presencePenalty,
        CancellationToken cancellationToken)
    {
        var markLabCommand = new MarkLabCommand(formFile, temperature,
maxTokens, topP, frequencyPenalty, presencePenalty);
        var result = await Mediator.Send(markLabCommand, cancellationToken);
        return result.GetActionResult();
    }

    [HttpGet]
    public async Task<IActionResult> GetResult()
    {
        var results = await _context.Results
            .AsNoTracking()
            .Select(result => new
            {
                result.Id,
                result.Mark,
                result.Remark,
                result.Conclusion,
                result.CreatedAt,
                result.UpdatedAt,
                Lab = new
                {
                    result.Lab.Id,
                    result.Lab.Name,
                    result.Lab.Extension,
                    CompletionSettings = new
                    {
                        result.Lab.CompletionSettings.Id,
                        result.Lab.CompletionSettings.Temperature,
                        result.Lab.CompletionSettings.MaxTokens,
                        result.Lab.CompletionSettings.TopP,
                        result.Lab.CompletionSettings.FrequencyPenalty,
```

```
                    result.Lab.CompletionSettings.PresencePenalty,
                    result.Lab.CompletionSettings.CreatedAt,
                    result.Lab.CompletionSettings.UpdatedAt
                }
            }
        })
        .ToListAsync();


    return Ok(results);
  }
}

using Assessor.Server.Api.Middlewares;

namespace Assessor.Server.Api.Extensions;

public static class GlobalExceptionHandlerExtensions
{
  public static IApplicationBuilder UseGlobalExceptionHandler(this
IApplicationBuilder builder)
  {
    return builder.UseMiddleware<GlobalExceptionHandlerMiddleware>();
  }
}

using Assessor.Server.Domain.Models;
using Microsoft.AspNetCore.Mvc;

namespace Assessor.Server.Api.Extensions;

public static class ResultExtensions
{
  public static IActionResult GetActionResult<TValue, TError>(this Result<TValue,
TError> result) where TError : Error
  {
    if (result.IsSuccess)
    {
      return new OkObjectResult(result.Value);
    }
```

```
        var objectResult = new ObjectResult(result.Error.Message)
        {
            StatusCode = (int)result.Error.StatusCode
        };

        return objectResult;
    }
}

using FluentValidation;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace Assessor.Server.Api.Filters;

public class FluentValidationFilter : IAsyncActionFilter
{
    private readonly IServiceProvider _serviceProvider;

    public FluentValidationFilter(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public async Task OnActionExecutionAsync(ActionExecutingContext context,
ActionExecutionDelegate next)
    {
        foreach (var parameter in context.ActionDescriptor.Parameters)
        {
            var isBindingSourceBody = parameter.BindingInfo?.BindingSource ==
BindingSource.Body;
            var isBindingSourceQuery = parameter.BindingInfo?.BindingSource ==
BindingSource.Query;
            var isParameterClass = parameter.ParameterType.IsClass;

            if (isBindingSourceBody || (isBindingSourceQuery && isParameterClass))
            {
                var validatorObj =
_serviceProvider.GetService(typeof(IValidator<>).MakeGenericType(parameter.Parame
terType));
```

```
            if (validatorObj is IValidator validator)
            {
                var subject = context.ActionArguments[parameter.Name];

                if (subject is not null)
                {
                    var result = await validator.ValidateAsync(new
ValidationContext<object>(subject), context.HttpContext.RequestAborted);

                    if (!result.IsValid)
                    {
                        var error = result.Errors.FirstOrDefault()?.ErrorMessage;

                        context.Result = new BadRequestObjectResult(error);
                        return;
                    }
                }
            }
        }

        await next();
    }
}

using System.Net;
using Assessor.Server.Domain.Constants;
using Assessor.Server.Domain.Models;
using Newtonsoft.Json;

namespace Assessor.Server.Api.Middlewares;

public class GlobalExceptionHandlerMiddleware : IMiddleware
{
    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        try
        {
            await next(context);
        }
```

```
        catch (Exception ex)
        {
            var error = new Error(HttpStatusCode.InternalServerError, ex.Message);

            var serializedError = JsonConvert.SerializeObject(error);

            context.Response.ContentType = MediaTypeConstants.JsonMediaType;
            context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

            await context.Response.WriteAsync(serializedError);
        }
    }
}

using Assessor.Server.Application.Lab.Commands.MarkLab;
using Assessor.Server.Domain.Extensions;
using FluentValidation;

namespace Assessor.Server.Api.Validators;

public class MarkLabCommandValidator  : AbstractValidator<MarkLabCommand>
{
    public MarkLabCommandValidator()
    {
        RuleFor(x => x.LabFile)
            .NotNull().WithMessage("Lab file is required.")
            .Must(file => file.IsDocFile() || file.IsPdfFile())
            .WithMessage("File must be either a .doc, .docx, or .pdf file.");
        RuleFor(x => x.Temperature).GreaterThanOrEqualTo(0).LessThanOrEqualTo(2);
        RuleFor(x =>
x.MaxTokens).GreaterThanOrEqualTo(0).LessThanOrEqualTo(15000);
        RuleFor(x => x.TopP).GreaterThanOrEqualTo(0).LessThanOrEqualTo(1);
        RuleFor(x => x.FrequencyPenalty)
            .Must(f => f is null or >= 0 and <= 2)
            .WithMessage("FrequencyPenalty must be null or between 0 and 2.");
        RuleFor(x => x.PresencePenalty)
            .Must(p => p is null or >= 0 and <= 2)
            .WithMessage("PresencePenalty must be null or between 0 and 2.");
    }
}
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ПЕРЕВІРКИ КОРЕКТНОСТІ ВИКОНАННЯ СТУДЕНТСЬКИХ РОБІТ»

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Алгаш А.М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
« 3 » червня 2025 р.

Вінниця ВНТУ - 2025 рік

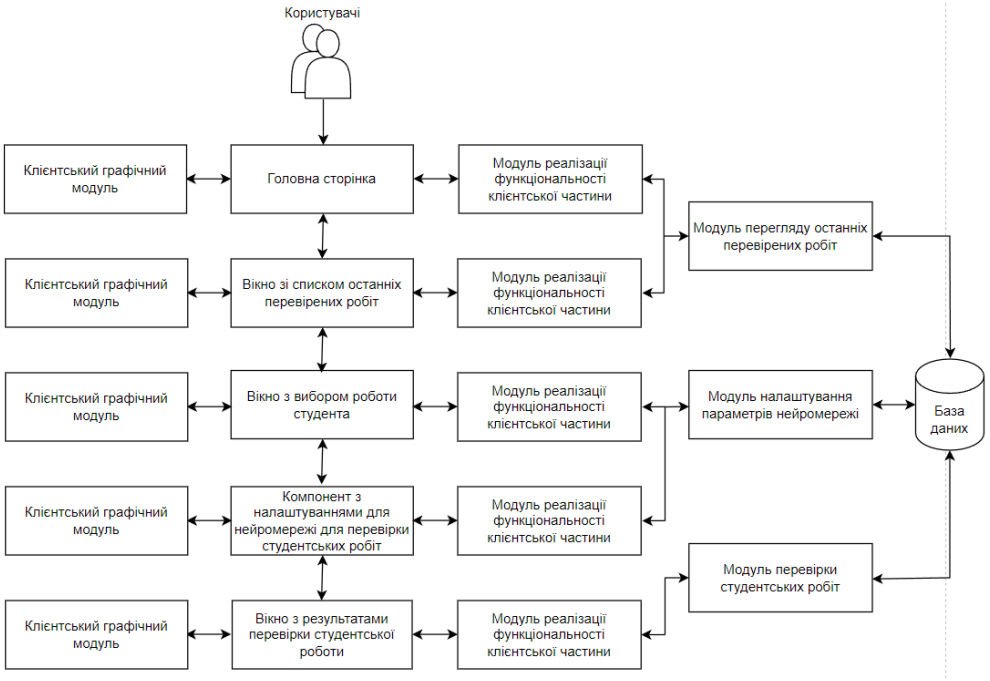

Рисунок В.1 − Загальна структурна схема функціонування програмного модуля перевірки коректності виконання студентських робіт

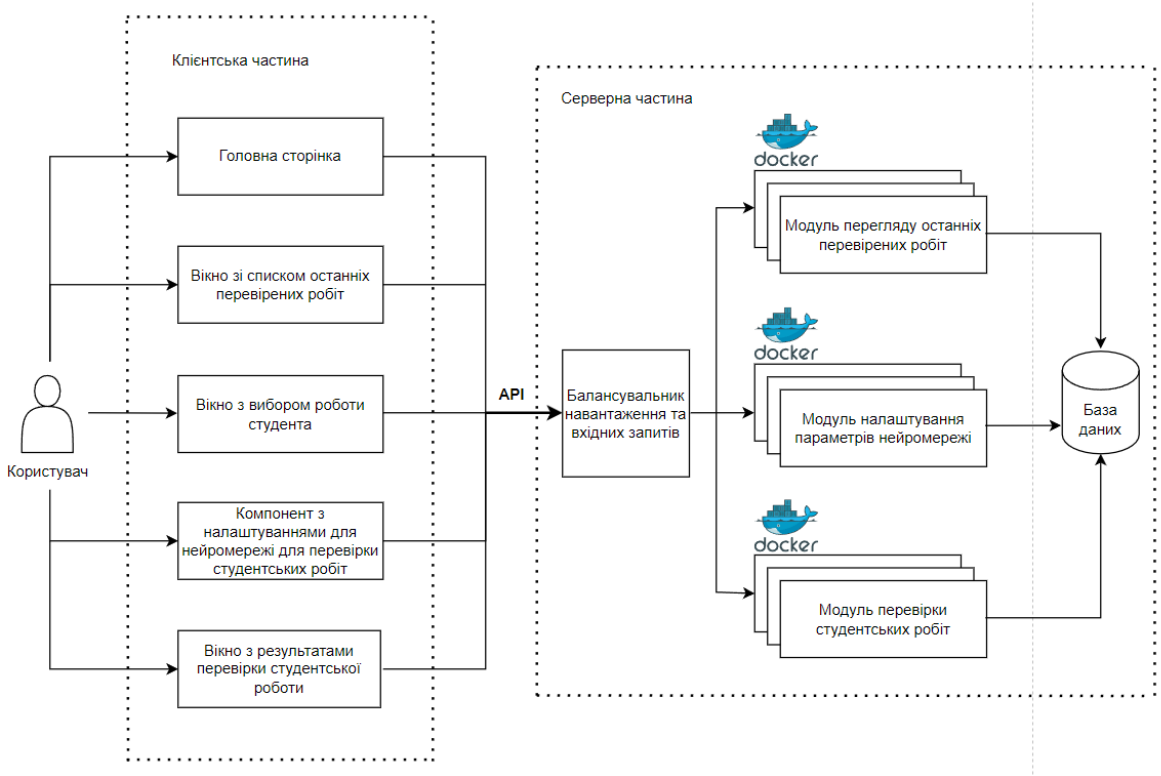

Рисунок В.2 –Загальна структурна схема компонентів програмного модуля перевірки коректності виконання студентських робіт

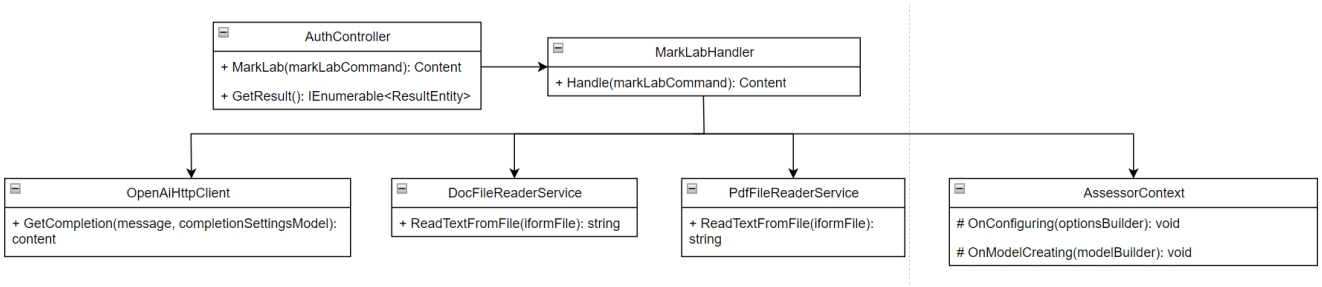

Рисунок В.3 – UML-діаграма класів серверної частини програмного модуля

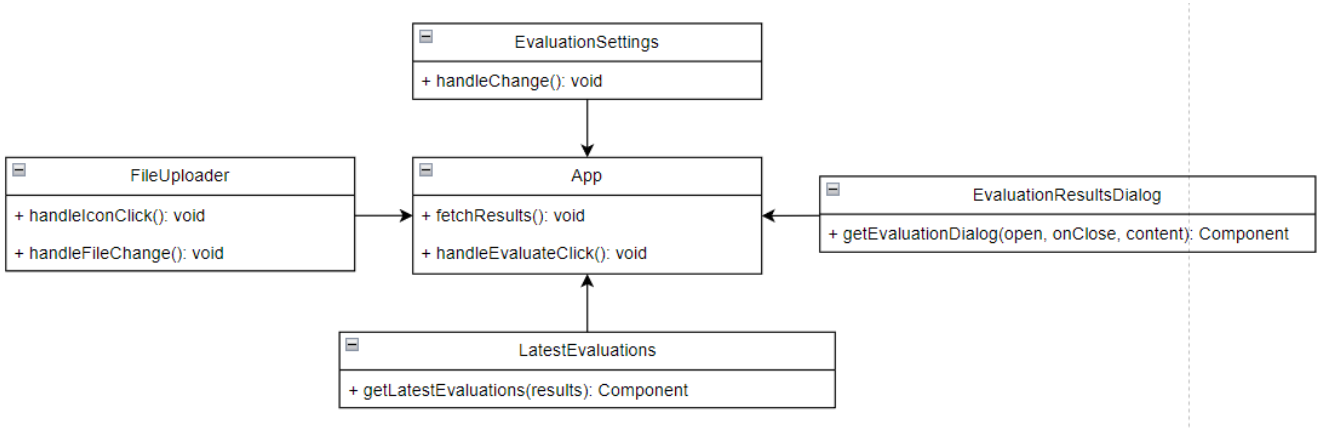

Рисунок В.4 – UML-діаграма класів клієнтської частини програмного модуля

Рисунок В.5 – Схема алгоритму роботи програмного модуля

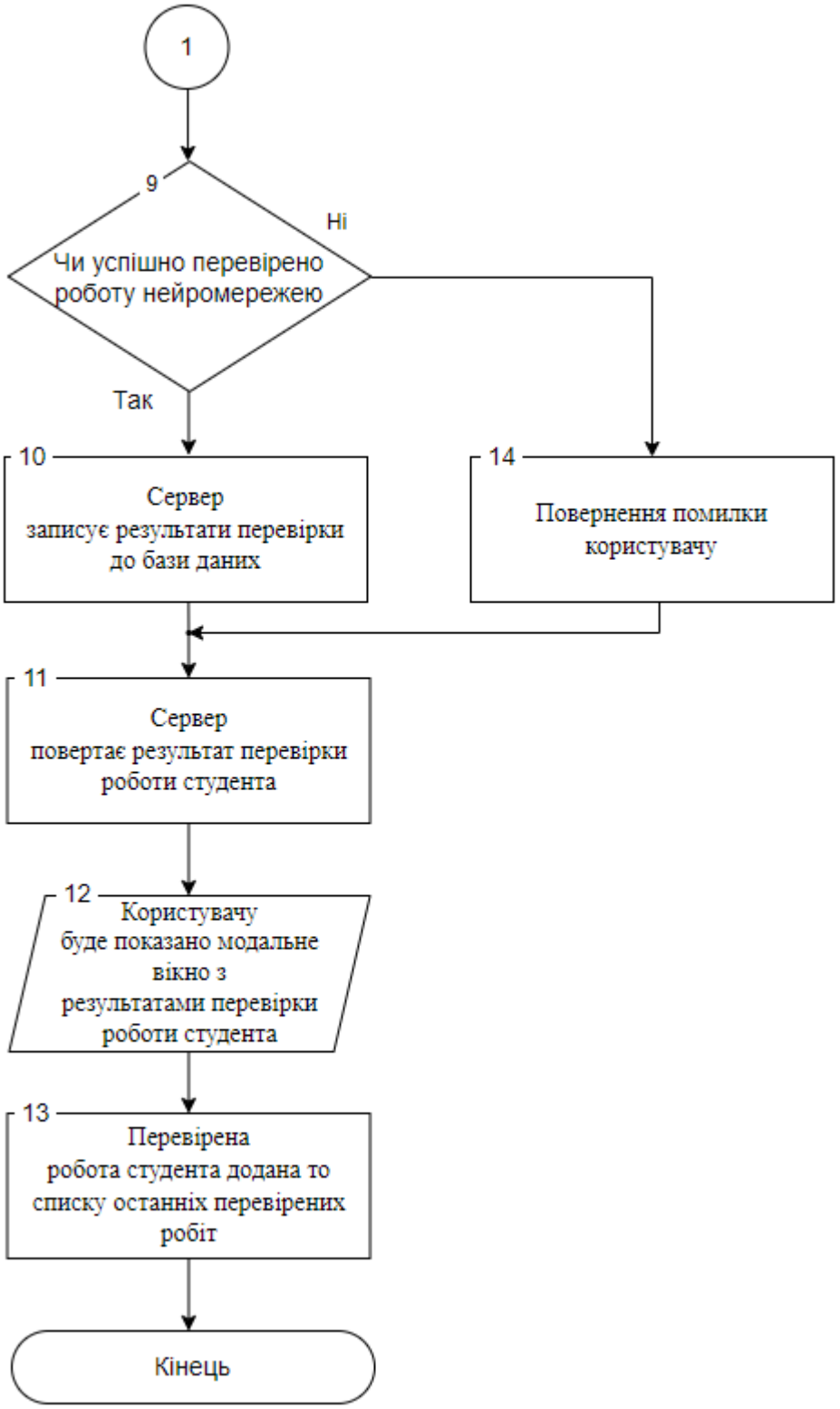

Рисунок В.5, аркуш 2

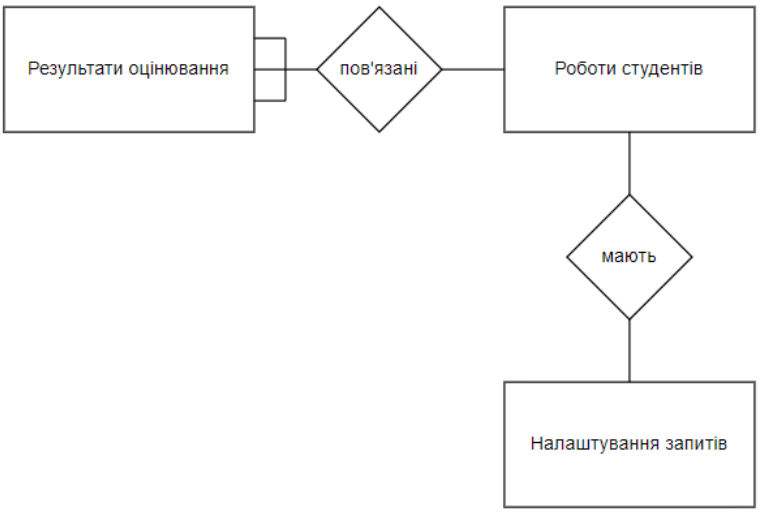

Рисунок В.6 – ER-модель для бази даних програмного модуля

Рисунок В.7 – Діаграма діяльності класу LabEntityConfiguration

Рисунок В.8 – Діаграма діяльності класу MarkLabHandler

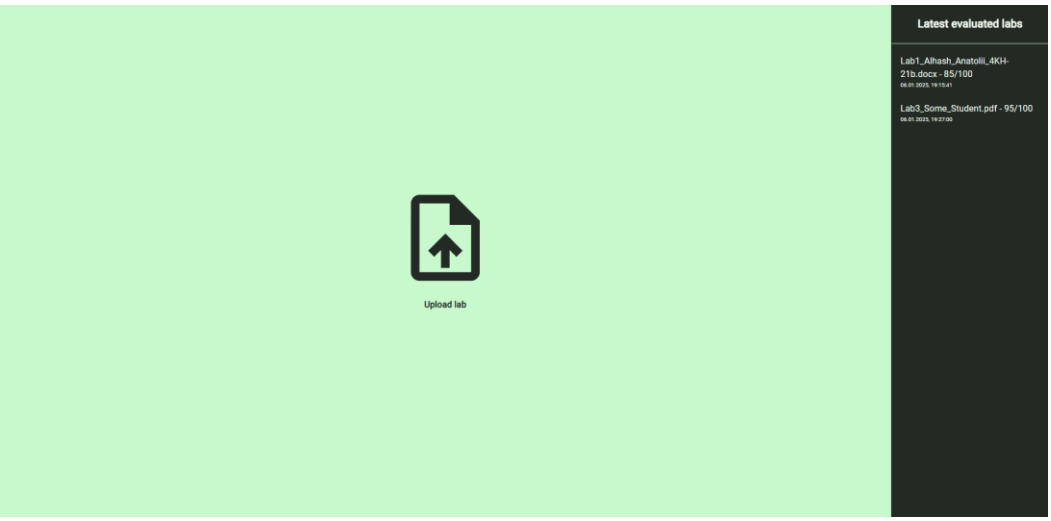

Рисунок В.9 – Загальний вигляд інтерфейсу головної сторінки розробки

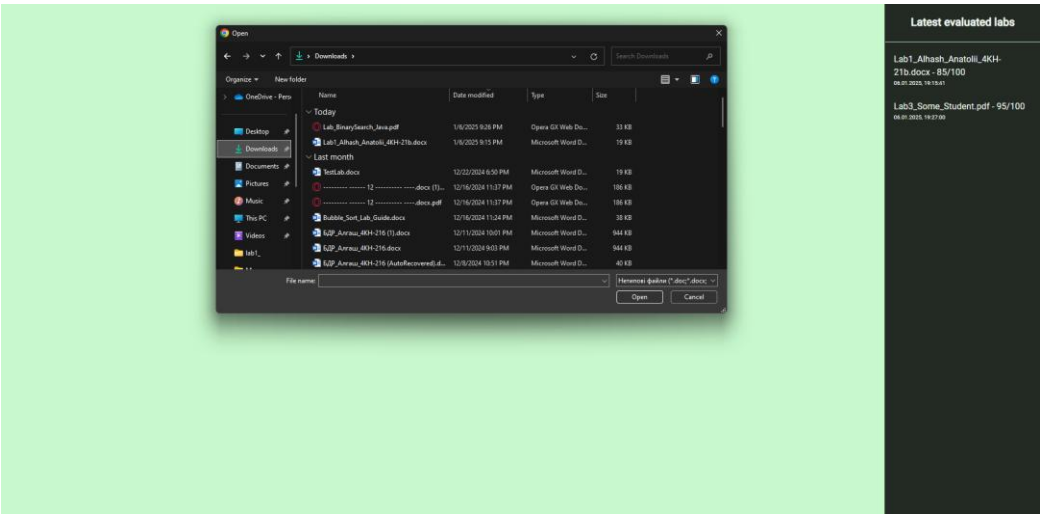

Рисунок В.10 – Загальний вигляд інтерфейсу вікна з вибором роботи студента

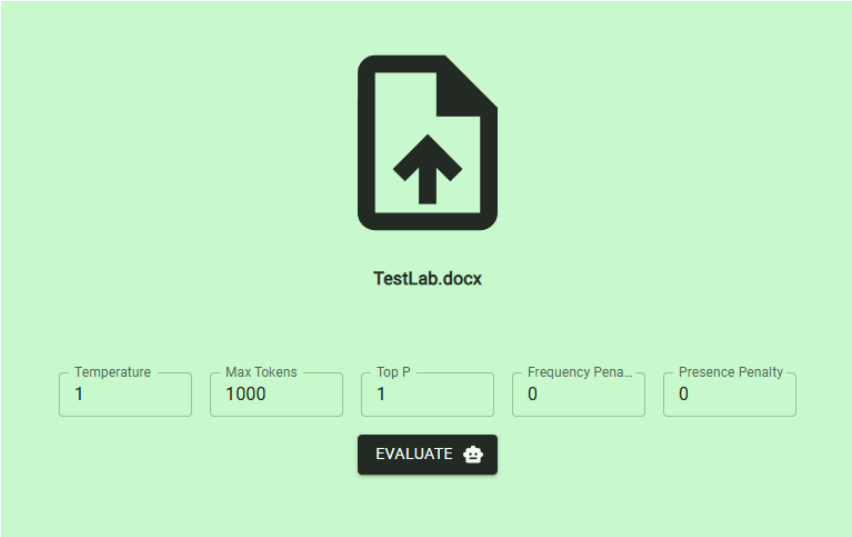

Рисунок В.11 – Параметри модуля перевірки коректності виконання студентських робіт

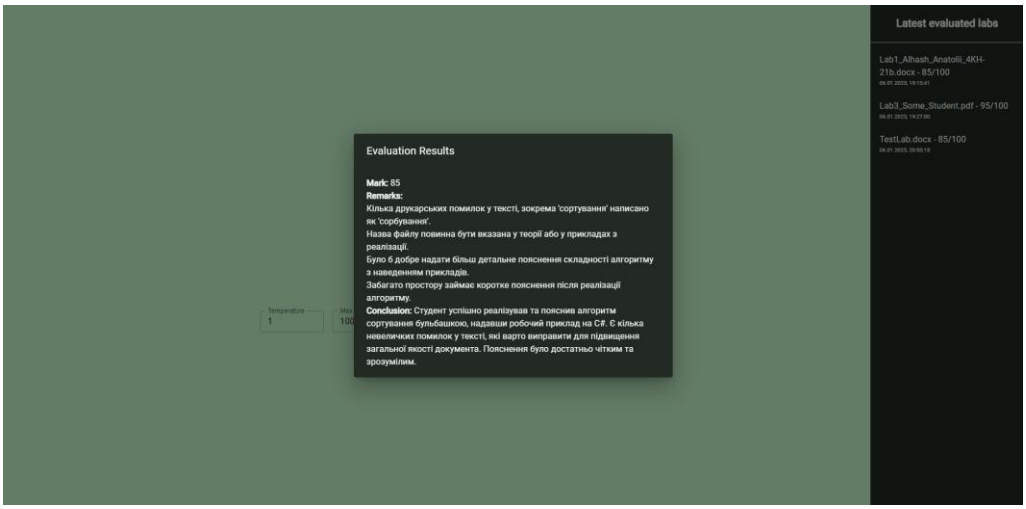

Рисунок В.12 – Загальний вигляд інтерфейсу вікна з результатами виконання роботи студента

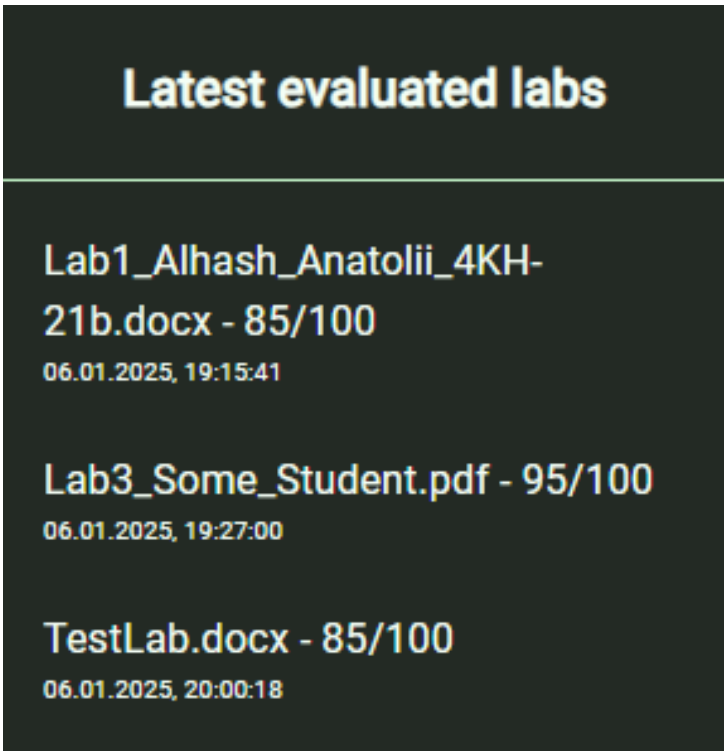

Рисунок В.13 – Загальний вигляд інтерфейсу вікна із зображенням перевірки останніх студентських робіт та їх оцінки

ДОДАТОК Г (довідниковий)

Інструкція користувача

Після запуску додатку з’являється головна сторінка, яка зображена на рисунку Г.1.

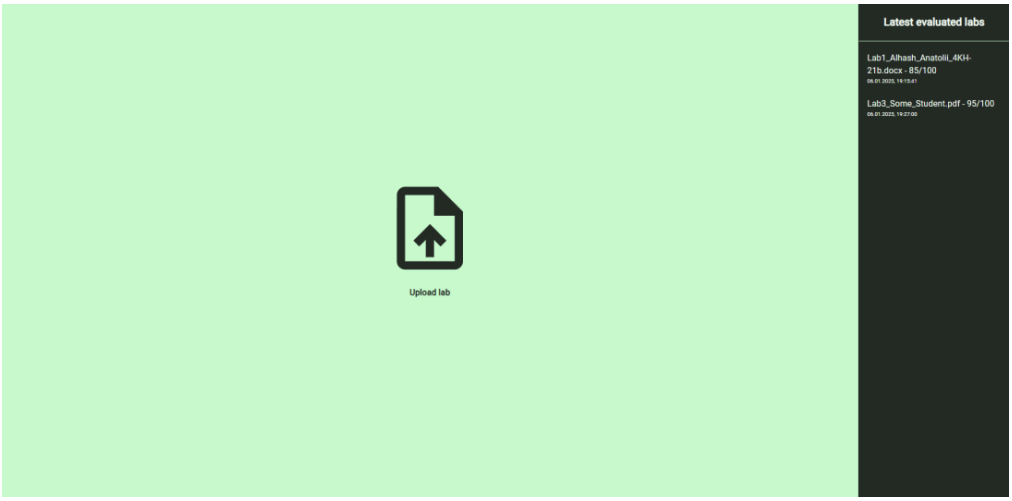

Рисунок Г.1 – Загальний вигляд інтерфейсу головної сторінки розробки

Якщо натиснути на іконку для завантаження файлу, то з’явиться вікно з вибором роботи студента. Вигляд цього вікна зображено на рисунку Г.2.

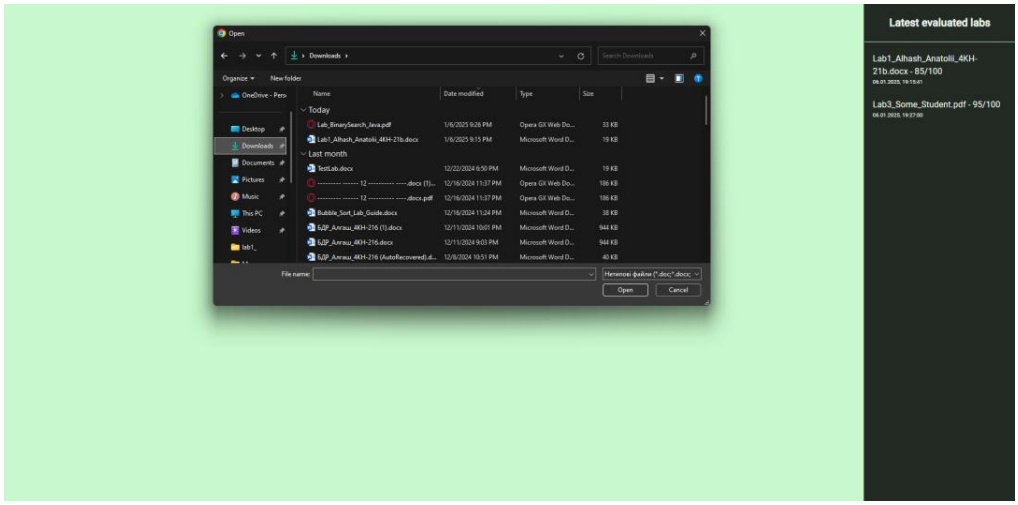

Рисунок Г.2 – Загальний вигляд інтерфейсу вікна з вибором роботи студента

Після успішного вибору файлу роботи студента, користувачу застосунка буде показано параметри, за допомогою яких він може коригувати параметри модуля для перевірки коректності виконання студентських робіт. Ці параметри показані на рисунку Г.3.

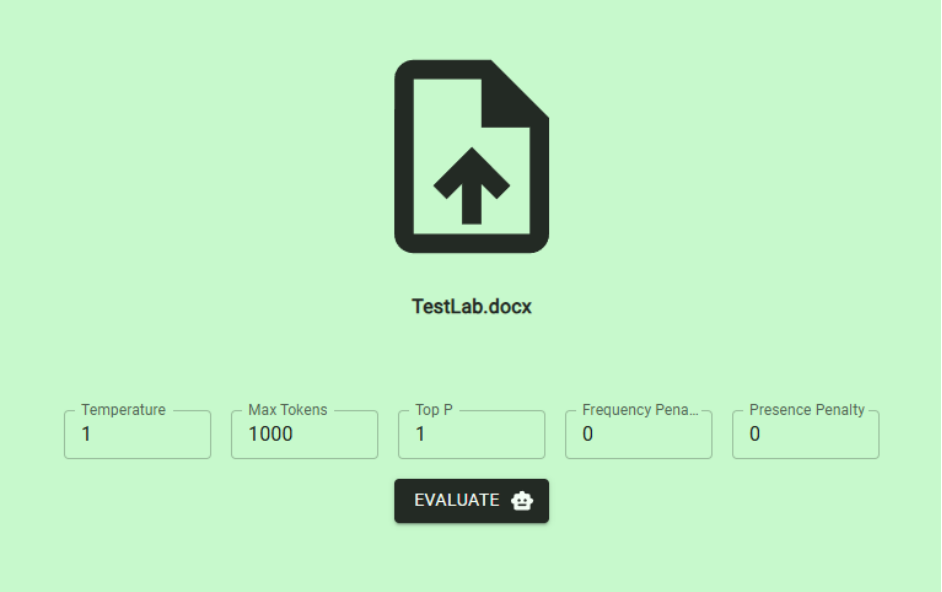

Рисунок Г.3 – Параметри модуля перевірки коректності виконання студентських робіт

Потім користувач може натиснути кнопку «EVALUATE», вказані налаштування з роботою студента будуть відправлені на сервер програмного модуля перевірки коректності виконання студентських робіт. Після цього сервер дасть відповідь із результатами виконання роботи студентом. Загальний вигляд інтерфейсу вікна з результатами виконання роботи студента зображено на рисунку Г.4.

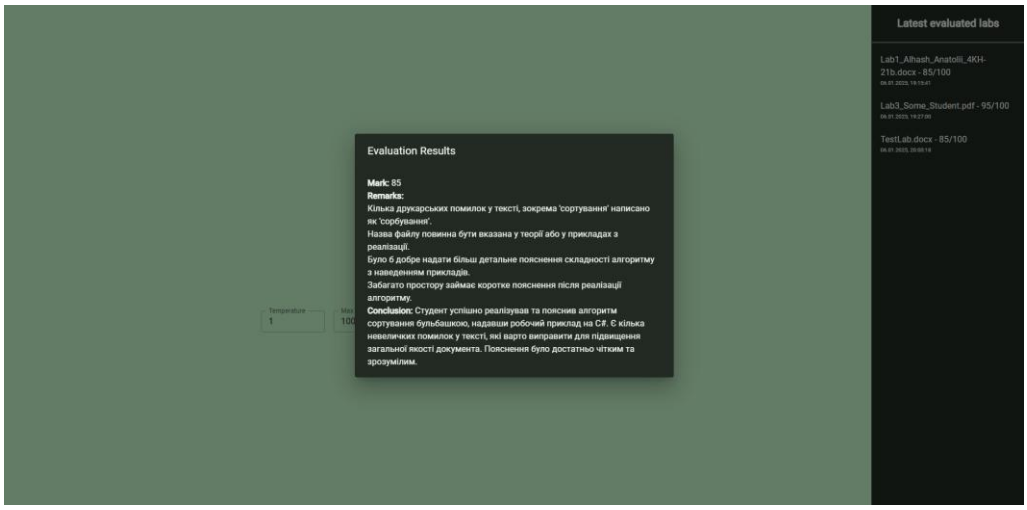

Рисунок Г.4 – Загальний вигляд інтерфейсу вікна з результатами виконання роботи студента

Після успішної перевірки коректності виконання студентської роботи, оцінена робота з’явиться у правому вікні, де зображені останні перевірені роботи та їх оцінки. Це вікно зображено на рисунку Г.5.

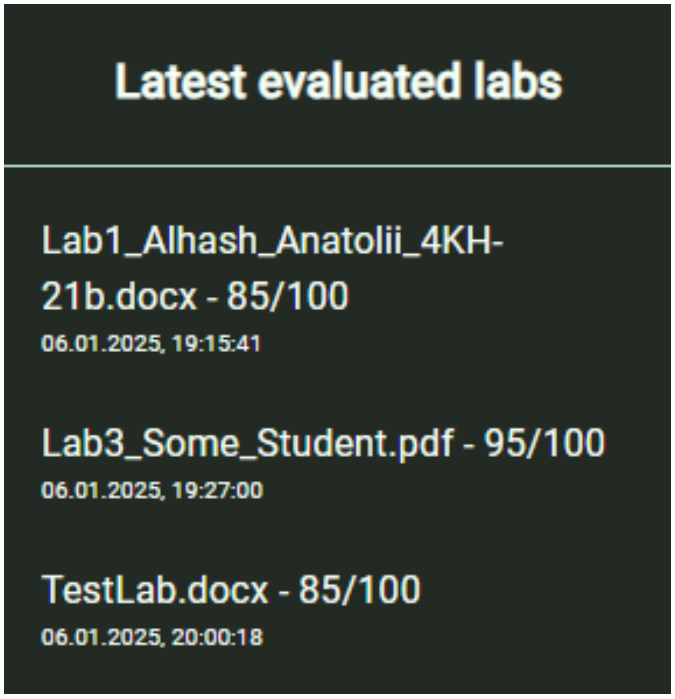

Рисунок Г.5 – Загальний вигляд інтерфейсу вікна із зображенням перевірки останніх студентських робіт та їх оцінки

ДОДАТОК Д

Свідоцтво про реєстрацію авторського права на твір

УКРАЇНА

СВІДОЦТВО

про реєстрацію авторського права на твір

№ 134193

Комп'ютерна програма «Програмний модуль перевірки коректності виконання студентських робіт»

(вид, назва твору)

Автор (співавтори) **Крилик Людмила Вікторівна, Алгаш Анатолій Михайлович**

(прізвище, ім'я, по батькові (за наявності), псевдонім (за наявності))

Авторські майнові права належать спільно **Крилик Людмила Вікторівна, вул. Політехнічна, 3, кв. 314, м. Вінниця, 21021; Алгаш Анатолій Михайлович, вул. Келецька, 98, м. Вінниця, 21012**

(прізвище, ім'я, по батькові (за наявності) фізичної особи / найменування юридичної особи, адреса)

Дата реєстрації 10 березня 2025 р.

Директор Державної організації «Український національний офіс інтелектуальної власності та інновацій» **Олена ОРЛЮК**

УКРНОІВІ