ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття рівня вищої освіти «бакалавр»

(рівень вищої освіти)

Програмно-апаратний комплекс автоматизації взаємодії викладача зі студентами

(тема кваліфікаційної роботи українською мовою)

Software and hardware complex for automating teacher-student interaction

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач ВО денної форми навчання

спеціальності 123 – Комп'ютерна інженерія.

(шифр і назва напряму підготовки, спеціальності)

Освітня програма «Комп'ютерна інженерія»

(назва освітньої програми)

Лебеденко Богдан Андрійович

(прізвище, ім'я, по-батькові)

Керівник к. ф-м. н., Антоненко О. С.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к. т. н., доц. Волощук Л. А.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2025 р.

Завідувач кафедри

_____ Євгеній МАЛАХОВ

(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № 5

протокол № __ від «__» ____ 2025 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Світлана АНТОЩУК

(підпис) (ім'я, прізвище)

Одеса - 2025

АНОТАЦІЯ

У даній кваліфікаційній роботі розглянуто тему «Програмно-апаратний комплекс автоматизації взаємодії викладача зі студентами».

Метою роботи є розробка комплексу в якому зможуть взаємодіяти викладачі та студенти. Передбачено автоматизацію деяких завдань для зменшення навантаження на користувачів та покращення процесу взаємодії.

У роботі описано основи теорії аналізу коду, прокторингу, а також застосування цих підходів у навчальному процесі. Проведено поетапний розбір поставлених задач та методи їх вирішення. Розглянуто сучасні та найбільш популярні програми для організації навчання, засоби аналізу коду та засоби прокторингу. Розроблено програму для взаємодії викладача зі студентами. Робота програми полягає в автоматизації взаємодії викладача зі студентами, перевірці робіт студентів на схожість, а також перевірці студентів на доброчесність під час проходження тестів. Розроблений комплекс зі зручним інтерфейсом придатний до застосування у навчальних закладах в рамках взаємодії викладача зі студентами. Програма забезпечує можливість створення викладачем курсів та робіт, перевірку робіт студентів на схожість, перевірку доброчесності студентів під час проходження тестів.

Комплекс взаємодії викладача та студентів виконує основні необхідні функції, дозволяючи спростити процес перевірки студентських робіт та доброчесності студентів.

ABSTRACT

This qualification work considers the topic «Software and hardware complex for automating teacher-student interaction».

The goal of the work is to develop a complex in which teachers and students can interact. It is planned to automate some tasks to reduce the load on users and improve the interaction process.

The work describes the basics of the theory of code analysis, proctoring, as well as the application of these approaches in the educational process. A step-by-step analysis of the tasks set and methods for solving them is carried out. Modern and most popular programs for organizing training, code analysis tools and proctoring tools are considered. A program for teacher-student interaction is developed. The work of the program consists in automating teacher-student interaction, checking student works for similarity, and checking students for integrity when taking tests. The developed complex with a convenient interface is suitable for use in educational institutions within the framework of teacher-student interaction. The program provides the ability for the teacher to create courses and papers, check student works for similarity, and check students' integrity when taking tests.

The complex of interaction between the teacher and students performs the basic necessary functions, allowing to simplify the process of checking student work and the integrity of students.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ 7

ВСТУП 8

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 10

1.1 Опис предметної області 10

1.2 Огляд існуючих систем управління навчальною діяльністю 12

1.2.1 Система управління навчанням Moodle 12

1.2.2 Система управління навчанням Google Classroom 13

1.2.3 Система управління навчанням Canvas 14

1.3 Огляд існуючих систем аналізу коду 15

1.3.1 Система аналізу коду MOSS 15

1.3.2 Система аналізу коду JPlag 16

1.3.3 Система аналізу коду Codequiry 17

1.4 Огляд існуючих систем прокторингу 18

1.4.1 Система прокторингу ProctorU 18

1.4.2 Система прокторингу ProctorExam 19

1.4.3 Система прокторингу Honorlock 20

1.5 Постановка завдання 21

2 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ 22

2.1 Архітектура програмно-апаратного комплексу 22

2.2 Вибір засобів реалізації для створення комплексу 26

2.2.1 Мова програмування C# 26

2.2.2 Фреймворк для веб-розробки ASP.NET Core 26

2.2.3 Модель програмування Razor Pages 27

2.2.4 CSS-фреймворк Bootstrap 27

2.2.5 Мова програмування JavaScript 27
2.2.6 Entity Framework Core 28
2.2.7 Система управління базами даних PostgreSQL 28
2.2.8 Провайдер Npgsql 28
2.2.9 Компіляторна платформа Roslyn 29
2.2.10 Бібліотека face-api.js 29
2.2.11 Веб-сервер IIS Express 30
2.3 Проєктування функціональної структури комплексу 30
2.3.1 Модуль управління обліковими записами 30
2.3.2 Модуль обліку курсів та завдань 31
2.3.3 Модуль студентських робіт 31
2.3.4 Модуль аналізу схожості коду 32
2.3.5 Модуль тестування 33
2.3.6 Модуль прокторингу 34
2.4 Проєктування бази даних 35
2.5 Апаратне забезпечення робочого місця користувача 39
3 РЕАЛІЗАЦІЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ 40
3.1 Компоненти застосунку програмно-апаратного комплексу 40
3.2 Реалізація управління обліковими записами 40
3.3 Реалізація обліку курсів та завдань 42
3.4 Реалізація модулю студентських робіт 43
3.5 Реалізація аналізу схожості коду 45
3.6 Реалізація модулю тестування 50
3.7 Реалізація прокторингу 53
3.8 Реалізація бази даних 57

3.9 Висновки до третього розділу 61
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64
ДОДАТОК А Опис сутностей бази даних 69
ДОДАТОК Б Код реалізації управління обліковими записами 75
ДОДАТОК В Код реалізації обліку курсів та завдань 82
ДОДАТОК Г Код реалізації модулю студентських робіт 85
ДОДАТОК Д Код реалізації аналізу схожості студентських робіт 89
ДОДАТОК Е Код реалізації модулю тестування 101
ДОДАТОК Ж Код реалізації модулю прокторингу 107
ДОДАТОК К Запити на створення доменів та таблиць бази даних 110
ДОДАТОК Л Запити на створення функцій та процедур 115
ДОДАТОК М Запити на створення ролей та надання їм привілеїв 118

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

LMS – Learning Management System – програмна платформа, призначена для організації, супроводу та оцінювання навчального процесу в електронному середовищі.

AST – Abstract Syntax Tree – структуроване представлення коду у вигляді дерева, яке відображає синтаксичну організацію програми без урахування форматування.

MVC – Model-View-Controller – архітектурний шаблон проєктування, що розділяє логіку програми на модель даних, інтерфейс користувача та обробник подій.

.NET – Network Enabled Technologies – платформа розробки програмного забезпечення від Microsoft, що забезпечує роботу застосунків на різних мовах програмування.

DTO – Data Transfer Object – спеціальний об’єкт в програмуванні, який використовується для безпечного та структурованого передавання даних між різними рівнями застосунку.

ВСТУП

Сучасний освітній процес у сфері програмування вимагає ефективних механізмів взаємодії між викладачами та студентами. Традиційні методи перевірки робіт та проведення тестувань часто виявляються трудомісткими та суб’єктивними, особливо при зростаючій кількості студентів. Водночас, дистанційні форми навчання створюють додаткові виклики щодо забезпечення академічної доброчесності. Це обумовлює потребу в інтегрованих рішеннях, які б поєднували функції перевірки робіт із засобами моніторингу навчальної активності.

На сьогодні існують різноманітні програмні рішення для організації взаємодії у навчальному процесі. Частина з них орієнтована на управління курсами та розподіл завдань, інші пропонують інструменти для автоматизованої перевірки робіт або проведення тестувань. Однак комплексні рішення, які б поєднували всі ці функції в єдиному середовищі з високим рівнем автоматизації, залишаються актуальною потребою сучасної освіти, особливо у сфері програмування.

Автоматизація навчального процесу є одним з найбільш пріоритетних напрямів розвитку освітніх технологій. У зв’язку з постійним зростанням навантаження на викладача та необхідністю забезпечення прозорості та об’єктивності оцінювання, розробка програмно-апаратних рішень здатна суттєво підвищити ефективність взаємодії між викладачем та студентами. Актуальність даного проєкту зумовлена потребою у створенні єдиної платформи, в якій поєднано управління курсами, перевірка робіт і прокторинг під час онлайн-тестування.

Метою проєкту є розробка комплексної системи, яка забезпечить елементи автоматизації обробки студентських завдань, а також реалізацію модуля прокторингу з використанням веб-камери. Завдання проєкту включають створення інтуїтивно зрозумілого інтерфейсу для викладачів та студентів, побудову надійної архітектури застосунку з дотриманням

принципів безпеки даних, а також інтеграцію безкоштовних сервісів для розпізнавання обличчя й аналізу коду на співпадіння.

Практична значимість полягає в тому, що проєкт дозволяє зменшити навантаження на викладачів за рахунок автоматизації процесу перевірки студентських робіт, підвищити об’єктивність оцінювання та забезпечити контроль академічної доброчесності. Система дає змогу аналізувати подані студентами матеріали, виявляти їх подібність за структурою, а також фіксувати ключові параметри поведінки під час виконання завдань.

Запропонований комплекс відрізняється поєднанням функцій аналізу вмісту студентських робіт із засобами моніторингу процесу тестування. Аналіз програмного коду здійснюється через порівняння структурних характеристик, що дозволяє виявляти схожість незалежно від поверхневих змін. Моніторинг тестування базується на аналізі ключових поведінкових параметрів, що забезпечує контроль за дотриманням правил. Очікуваними результатами є створення програмно-апаратного комплексу з розвиненим функціоналом, що може бути масштабований та адаптований до потреб закладів освіти різного рівня.

Для досягнення мети необхідно виконати наступні завдання:

1) провести аналіз предметної області та вимог до системи;

2) дослідити існуючі рішення в галузі автоматизації освітнього процесу;

3) розробити методи аналізу програмного коду для виявлення структурних схожостей;

4) запропонувати алгоритми контролю академічної доброчесності під час тестувань;

5) спроєктувати архітектуру програмно-апаратного комплексу;

6) реалізувати інтуїтивний інтерфейс для користувачів;

7) розробити основні програмні модулі проєкту;

8) провести тестування розробленого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області

Сучасна освітня інфраструктура все частіше залежить від цифрових інструментів, які спрощують комунікацію між викладачами та студентами, автоматизують перевірку робіт і забезпечують контроль академічної доброчесності. У даній роботі предметна область охоплює три ключові напрями: системи управління навчанням (LMS), автоматизований аналіз програмного коду та технології прокторингу.

Системи управління навчанням (LMS) є спеціалізованими програмними платформами, призначеними для організації та керування навчальним процесом [1]. Вони надають інструменти для створення курсів, розподілу матеріалів, проведення оцінювання та забезпечення комунікації між учасниками освітнього процесу. Сучасні LMS також інтегрують функції для проведення онлайн-лекцій, тестувань та спільної роботи, що робить їх особливо корисними для дистанційного та змішаного навчання [1].

Аналіз програмного коду − це широкий спектр методів та інструментів, що застосовуються для оцінки якості, безпеки, продуктивності та надійності програмного забезпечення [2].

Аналіз коду можна розділити на наступні категорії [2]:

1) Статичний аналіз коду. Цей вид аналізу передбачає перевірку вихідного коду програми без його фактичного виконання. Інструменти статичного аналізу виявляють потенційні помилки, вразливості безпеки та порушення стандартів кодування, аналізуючи структуру, синтаксис і можливі потоки виконання коду;

2) Динамічний аналіз коду. На противагу статичному, динамічний аналіз здійснюється під час виконання програми. Він дозволяє виявити проблеми, які проявляються лише в реальних умовах взаємодії з системою, такі

як витоки пам’яті, помилки часу виконання або вразливості, що виникають при обробці зовнішніх даних;

3) Ручний аналіз коду. Це систематична перевірка вихідного коду людьми (розробниками або експертами). Ручний аналіз ефективний для виявлення складних логічних помилок, покращення архітектурних рішень, а також забезпечення читабельності та підтримки коду.

У сфері програмування важливим аспектом є перевірка коду на плагіат. На відміну від текстових документів, програмний код має структурований синтаксис, який можна аналізувати за допомогою спеціалізованих алгоритмів [3]. Перевірка коду на плагіат, як правило, відносять до статичного аналізу коду.

Прокторинг − це система моніторингу, що забезпечує контроль за дотриманням правил під час онлайн-тестувань [4]. Вона може включати відеоспостереження, аналіз поведінки користувача (рухи очей, положення голови) та виявлення підозрілих дій. Сучасні рішення використовують алгоритми комп’ютерного зору та машинного навчання для автоматизації цих процесів, зменшуючи необхідність у безпосередній участі людини [5].

Розглянемо сучасні технології прокторингу [5]:

1) відеомоніторинг – включає розпізнавання обличчя, аналіз очного контакту та положення голови;

2) акустичний аналіз – виявляє сторонні звуки та мовлення;

3) поведінковий аналіз – ідентифікує підозрілі дії (часті перемикання вкладок, використання сторонніх додатків).

Сучасні системи часто поєднують ці технології з алгоритмами штучного інтелекту для підвищення ефективності [5].

Розглянуті напрями становлять основу для створення сучасних освітніх інструментів. Їхня інтеграція дозволяє будувати комплексні рішення, які не лише автоматизують навчальний процес, але й забезпечують його прозорість та об’єктивність.

1.2 Огляд існуючих систем управління навчальною діяльністю

1.2.1 Система управління навчанням Moodle

Moodle (Modular Object-Oriented Dynamic Learning Environment) − це одна з провідних систем управління навчанням з відкритим вихідним кодом, розроблена у 2001–2002 роках. З тих пір проєкт постійно розвивається завдяки внескам міжнародної спільноти розробників [6].

Moodle розроблена на мові програмування PHP та використовує реляційні бази даних, такі як MySQL або PostgreSQL [7-9]. Вона підтримує міжнародні стандарти електронного навчання, що дозволяє інтегрувати різноманітний навчальний контент. Для взаємодії з іншими системами Moodle надає вбудовані веб-сервіси та програмні інтерфейси, які дозволяють зовнішнім застосункам обмінюватися даними з платформою (наприклад, реєструвати курс, створювати облікові записи або завантажувати результати).

Система пропонує широкий спектр інструментів для управління навчальним процесом. Вона дозволяє створювати структуровані курси з чіткою ієрархією модулів і тем, а також надає гнучку систему обмежень доступу. Серед інструментів навчання варто відзначити вікі-систему, глосарії, бази даних та інтерактивні уроки. Moodle також включає систему оцінювання з різними типами завдань, рубриками оцінювання, журналом оцінок та інструментами зворотного зв’язку. Для комунікації між учасниками навчального процесу система надає форуми різних типів, чати, систему повідомлень та коментарі.

Однією з ключових переваг є те, що Moodle можна завантажити та змінювати під свої потреби, оскільки вихідний код доступний відкрито. Платформа повністю локалізована українською мовою, що робить її зручною для використання в Україні. Велика кількість безкоштовних плагінів значно розширює функціонал системи. Moodle також відома своєю гнучкістю та

можливістю глибокої кастомізації під конкретні потреби навчального закладу. Активна спільнота користувачів забезпечує підтримку та обмін досвідом.

Незважаючи на численні переваги, ця система має певні недоліки. Вона вимагає технічних знань для адміністрування, що може ускладнити її впровадження в закладах без спеціалізованого IT-персоналу. Деякі важливі функції, такі як розширена аналітика чи спеціалізовані типи завдань, доступні лише через додаткові плагіни. Інтерфейс системи може здатися застарілим порівняно з сучасними аналогами. Moodle може бути ресурсомісткою, що вимагає потужного серверного обладнання.

1.2.2 Система управління навчанням Google Classroom

Google Classroom є сучасною хмарною платформою для організації навчального процесу, розробленою компанією Google у 2014 році [10]. Цей сервіс створено як частину екосистеми Google Workspace for Education [11]. Платформа швидко набула популярності завдяки своїй простоті та інтуїтивності.

Платформу побудовано на базі хмарних технологій Google. Вона тісно інтегрована з іншими сервісами компанії. Система не вимагає установки додаткового програмного забезпечення та повністю функціонує через веб-браузер.

Основним призначенням Google Classroom є спрощення процесу організації навчання, розподілу матеріалів та комунікації між викладачами та студентами. Система дозволяє легко створювати віртуальні класи, до яких студенти можуть приєднуватися за спеціальним кодом або запрошенням.

Серед ключових функцій варто відзначити простий механізм розподілу завдань, який дозволяє викладачам швидко створювати, розсилати та перевіряти роботи студентів. Система автоматично організовує робочі потоки, нагадує про терміни виконання та спрощує процес оцінювання. Для

комунікації передбачені оголошення, коментарі до завдань та можливість особистих повідомлень.

Головною перевагою Google Classroom є його простота та зручність використання. Ця платформа не вимагає спеціальних технічних знань і може бути освоєна викладачами та студентами за короткий період часу. Тісна інтеграція з іншими сервісами Google робить її хорошим вибором для закладів, які вже використовують ці інструменти [11]. Важливою перевагою є також безкоштовний доступ для освітніх установ.

Однак система має і певні обмеження. У порівнянні з повноцінними системами управління навчанням, Google Classroom пропонує значно менше функцій для структурування навчального контенту та аналітики навчання. Відсутність вбудованих інструментів для створення складних тестів або перевірки на плагіат обмежує її застосування для деяких видів навчальної діяльності. Крім того, робота з платформою повністю залежить від інтернет-з’єднання та серверів Google.

Google Classroom знайшла широке застосування в школах, на курсах додаткової освіти та для організації невеликих навчальних проєктів. Її простота та доступність роблять її популярним вибором для швидкого впровадження дистанційного навчання без значних технічних витрат.

1.2.3 Система управління навчанням Canvas

Canvas є сучасною системою управління навчанням, розробленою компанією Instructure у 2011 році [12]. Ця платформа отримала популярність серед навчальних закладів завдяки своєму інтуїтивному інтерфейсу та хорошим функціональним можливостям.

Canvas побудовано як хмарне рішення, але також воно доступно для локального встановлення. Система відрізняється гнучкою API-архітектурою, що дозволяє легко інтегрувати її з іншими освітніми сервісами та корпоративними системами [13].

Основною перевагою Canvas є сучасний, адаптивний інтерфейс. Система пропонує повноцінний набір інструментів для створення курсів, включаючи редактор контенту, засоби для створення тестів і завдань, а також інструменти для спільної роботи.

Canvas відзначається потужною системою оцінювання з підтримкою різноманітних рубрик і схем оцінювання. Система автоматизує багато процесів, таких як розподіл завдань, нагадування про дедлайни і навіть деякі елементи перевірки робіт. Для комунікації передбачені інтегровані інструменти: від дискусійних форумів до відеоконференцій. Важливою особливістю є також розвинена система аналітики, яка дозволяє відстежувати прогрес студентів і ефективність навчальних матеріалів.

Однак Canvas має і деякі недоліки. Хоча базовий функціонал досить простий у використанні, деякі розширені функції вимагають технічних знань для налаштування. Безкоштовна версія має обмеження, а повноцінне використання вимагає придбання ліцензії. Деякі користувачі відзначають, що система може бути надто складною для простих навчальних сценаріїв [14].

1.3 Огляд існуючих систем аналізу коду

1.3.1 Система аналізу коду MOSS

MOSS (Measure of Software Similarity) є інноваційною системою для виявлення схожості програмного коду, розробленою в Стенфордському університеті у 1994 році [15]. Технічні особливості MOSS ґрунтуються на передових алгоритмах аналізу коду. Система використовує метод «віконного відбитку», де код розбивається на перекриваючі сегменти, для кожного з яких генерується унікальний ідентифікатор. Цей підхід дозволяє виявляти схожість незалежно від поверхневих змін, таких як перейменування змінних, зміна форматування або перестановка операторів [16].

MOSS підтримує широкий спектр мов програмування. Система працює через веб-інтерфейс, де викладачі можуть завантажувати студентські роботи для аналізу. Після обробки MOSS генерує звіт з відсотком схожості між усіма парами робіт та виділяє конкретні схожі фрагменти.

Основною перевагою MOSS є її висока точність у виявленні структурної схожості коду. Вона здатна виявляти плагіат навіть у випадках, коли код був значно переписаний.

Однак MOSS має деякі обмеження. Система вимагає, щоб викладачі завантажували всі роботи для порівняння одночасно, що може бути незручним для великих груп. Також вона не інтегрована безпосередньо в системи управління навчанням і працює як окремий сервіс.

1.3.2 Система аналізу коду JPlag

JPlag є потужним інструментом для виявлення схожості програмного коду, розробленим у інституті Карлсруе (Німеччина) у 1997 році. Ця система стала одним із найбільш надійних рішень для академічного виявлення плагіату в програмних завданнях [17].

Технічна основа JPlag ґрунтується на алгоритмі токенізації коду, який перетворює вихідний текст на послідовність унікальних токенів, що відображають структурні елементи програми [18]. Система використовує модифікований алгоритм «найдовшої спільної підпослідовності» (LCS) для виявлення схожих фрагментів між різними роботами. Цей підхід дозволяє ігнорувати поверхневі відмінності, такі як коментарі, форматування чи імена змінних [19].

JPlag підтримує широкий спектр мов програмування, включаючи Java, C, C++, C#, Python, Scheme. Система працює у вигляді консольної програми або через веб-інтерфейс, де користувачі можуть завантажувати файли для аналізу. Після обробки JPlag генерує детальні звіти, які візуалізують знайдені збіги між роботами.

Основні переваги JPlag включають високу точність виявлення структурних збігів і здатність працювати з різними мовами програмування. Система особливо ефективна для виявлення «розподіленого» плагіату, коли фрагменти коду з різних джерел комбінуються в одній роботі. JPlag також відзначається швидкістю роботи та можливістю обробляти великі набори файлів. Серед обмежень JPlag варто відзначити відсутність повноцінної інтеграції з системами управління навчанням та дещо застарілий інтерфейс. Система також вимагає певних технічних знань для налаштування та інтерпретації результатів.

1.3.3 Система аналізу коду Codequiry

Codequiry є сучасною хмарною платформою для виявлення схожості програмного коду, розробленою у 2016 році як інноваційне рішення для академічних установ та IT-компаній [20]. Ця система поєднує традиційні методи аналізу коду з передовими технологіями штучного інтелекту, що дозволяє виявляти як прямі запозичення, так і складні випадки плагіату.

Технічна основа Codequiry ґрунтується на хмарній архітектурі з використанням розподілених обчислювальних систем. Система застосовує комбінований підхід, який включає лексичний, семантичний і стилістичний аналіз коду, доповнений алгоритмами машинного навчання для підвищення точності виявлення схожостей. Це дозволяє системі розпізнавати не лише дослівні збіги, а й структурну та логічну схожість програмних фрагментів.

Codequiry підтримує багато програмування, включаючи Java, Python, C++, C#, JavaScript та інші популярні мови. Система працює через зручний веб-інтерфейс, що не вимагає локального встановлення додаткового програмного забезпечення. Після аналізу Codequiry генерує детальні звіти з візуалізацією знайдених збігів та оцінкою рівня схожості.

Основною перевагою Codequiry є його комплексний підхід до аналізу коду, який дозволяє виявляти різні форми плагіату. Система також

відзначається зручним інтерфейсом, можливістю створення власних баз робіт для порівняння та інтеграцією з системами управління навчанням. Важливою особливістю є підтримка командної роботи, що дозволяє кільком викладачам спільно аналізувати результати. Серед обмежень системи варто відзначити відсутність локальної версії, що робить її залежною від якості інтернет-з'єднання. Безкоштовний функціонал обмежений, а повноцінне використання вимагає придбання підписки.

1.4 Огляд існуючих систем прокторингу

1.4.1 Система прокторингу ProctorU

ProctorU є провідною системою онлайн-прокторингу, заснованою у 2008 році [21]. Ця платформа спеціалізується на забезпеченні академічної доброчесності під час дистанційного тестування, поєднуючи автоматизовані технології з можливістю живого спостереження.

ProctorU базується на комплексному підході до моніторингу екзаменаційного процесу. Система використовує веб-камеру, мікрофон та технологію запису екрана для відстеження дій користувача. В ній інтегровано передові алгоритми комп'ютерного зору для автоматичного виявлення підозрілих дій, таких як відведення погляду або присутність сторонніх осіб.

Особливістю ProctorU є гібридна модель прокторингу, яка поєднує наступні технології [21]:

– автоматизований моніторинг за допомогою штучного інтелекту;
– живе спостереження сертифікованими прокторами;
– запис усієї сесії для подальшого аналізу.

Основні переваги ProctorU включають високу точність виявлення порушень, та гнучкість у виборі рівня контролю (від повністю автоматизованого до супроводу живого проктора). Система особливо ефективна для масштабних екзаменаційних сесій, де потрібно забезпечити

однакові умови для всіх учасників. Серед обмежень варто відзначити вимоги до технічного обладнання з боку користувачів та необхідність стабільного інтернет-з’єднання.

1.4.2 Система прокторингу ProctorExam

ProctorExam є сучасною системою онлайн-прокторингу, розробленою у Нідерландах. Ця платформа спеціалізується на забезпеченні академічної доброчесності під час дистанційного тестування, пропонуючи унікальне поєднання автоматизованих технологій та гнучких варіантів спостереження [22].

Система базується на комплексному моніторингу екзаменаційного процесу з використанням веб-камери, мікрофона та технології запису екрана. ProctorExam інтегрує передові алгоритми комп’ютерного зору для автоматичного виявлення підозрілих дій, включаючи відведення погляду, використання сторонніх пристроїв чи присутність інших осіб у кімнаті [23].

Можна виділити наступні особливості цієї системи:

– можливість повністю автоматизованого прокторингу;
– варіант змішаного режиму з участю живого проктора;
– функція офлайн-тестування з подальшою синхронізацією даних;
– розширені методи ідентифікації (біометрія обличчя та голосу).

Основною перевагою ProctorExam є його адаптивність до різних навчальних сценаріїв. Система підтримує широкий спектр типів екзаменів - від стандартних тестів до практичних завдань та усних відповідей. Важливою перевагою є повна інтеграція з популярними LMS-платформами та можливість використання на мобільних пристроях, що значно розширює сферу застосування.

Серед обмежень системи варто відзначити необхідність встановлення додаткового програмного забезпечення, що може ускладнювати процес підготовки до екзамену. Деякі користувачі також повідомляють про обмежену підтримку окремих операційних систем, що може бути суттєвим недоліком для певних навчальних закладів.

1.4.3 Система прокторингу Honorlock

Honorlock є системою онлайн-прокторингу, яка поєднує передові технології штучного інтелекту з можливістю втручання сертифікованих прокторів [24]. Цю платформу спеціально розроблено для забезпечення максимального рівня академічної доброчесності під час дистанційного тестування.

Система використовує комплексний підхід до моніторингу, що включає веб-камеру, запис екрана та детальний аналіз поведінки користувача. Honorlock відзначається особливо високою точністю у виявленні потенційних порушень, таких як перемикання вкладок, використання сторонніх додатків чи спроби спілкування з третіми особами.

Ключовою особливістю Honorlock є технологія «Search and Destroy», яка автоматично сканує інтернет у пошуку можливих джерел відповідей [25]. Головною перевагою Honorlock є його висока ефективність у запобіганні академічній недоброчесності при мінімальних технічних вимогах до користувачів. Серед інших характеристик системи можна виділити миттєві сповіщення для студентів при виявленні підозрілих дій, інтеграція з провідними LMS-платформами та автоматизований звіт про всі виявлені підозрілі дії.

Система відзначається зручним інтерфейсом та швидким налаштуванням, що робить її популярним вибором серед навчальних закладів. Однак деякі користувачі відзначають обмежену підтримку мов, окрім англійської, що може бути суттєвим недоліком для неангломовних установ. Також варто враховувати відносно високу вартість ліцензії, яка може бути перешкодою для невеликих навчальних закладів.

1.5 Постановка завдання

Проаналізувавши предметну область, існуючі рішення в рамках систем управління навчанням, аналізу коду, систем прокторингу та врахувавши основні вимоги до програмно-апаратного комплексу, необхідно вирішити наступні завдання:

1) забезпечити студентам можливість завантажувати свої роботи, а викладачам переглядати роботи та оцінювати їх;

2) створити інструмент пошуку структурних співпадінь у коді студентських робіт;

3) розробити модуль дистанційного тестування з контролем доброчесності, використовуючи камеру пристрою;

4) спроєктувати базу даних для надійного зберігання інформації про користувачів, курси, роботи, результати тестів та інших даних;

5) розробити зручний користувацький інтерфейс;

6) протестувати тестування функціоналу програмно-апаратного комплексу.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

2.1 Архітектура програмно-апаратного комплексу

Програмну архітектуру комплексу побудовано за принципом трирівневої клієнт-серверної моделі, яка забезпечує розділення відповідальності між представленням, логікою обробки даних та зберіганням інформації [26]. Такий підхід спрощує розгортання, підтримку та масштабування системи. Схему програмної архітектури комплексу зображено на рисунку 2.1.

Рисунок 2.1 − Програмна архітектура комплексу

На клієнтському рівні користувачі взаємодіють з програмним забезпеченням через інтерфейс веб-застосунку. Студенти мають змогу проходити тестування, під час якого активується пристрій відеоспостереження. На основі відеопотоку виконується аналіз орієнтації обличчя користувача. Під час проходження тесту клієнтська частина кожні декілька секунд формує звітні дані про поведінку користувача, які передаються на сервер для подальшої обробки.

На рівні сервера застосунку обробляються всі запити клієнта. Цей сервер відповідає за керування логікою взаємодії з базою даних, обробку результатів тестування, аналіз коду у студентських роботах, а також за фіксацію та інтерпретацію інформації з модулю прокторингу. Логіка сервера застосунку

чітко розділена за сферами відповідальності, зокрема на обробку тестових сесій, перевірку схожості фрагментів коду та загальну обробку завдань студентів.

На рівні зберігання даних здійснюються запити до серверу бази даних, а отримані результати повертаються на сервер застосунку. База даних містить відомості про користувачів, навчальні курси, студентські роботи, тестові завдання, результати перевірки знань та інформацію, пов’язану з академічною доброчесністю.

Обрана архітектура має наступні переваги:

1) чітке розділення функціоналу. Кожен рівень відповідає за виконання специфічних функцій, що забезпечує логічне розділення програми на компоненти. Це значно спрощує процеси розробки та подальшого супроводу коду;

2) гнучкість та автономність компонентів. Можливість модифікувати або замінювати окремі компоненти кожного рівня без впливу на функціональність інших рівнів гарантує високу гнучкість системи. Це дозволяє легко адаптувати її до нових вимог або інтегрувати нові технології;

3) підвищена безпека та надійність. Розділення системи на окремі рівні сприяє посиленню безпеки, оскільки кожен рівень може мати власні механізми захисту. Крім того, це підвищує загальну надійність системи, оскільки потенційні збої в одному рівні менше впливають на роботу інших.

У якості шаблону проєктування для рівня сервера застосунку обрано MVC (Model-View-Controller) [27]. Цей підхід розділяє веб-застосунок на три самостійні частини:

1) компонент Model (Модель) відповідає за внутрішнє подання даних. Тут відбувається обробка інформації та взаємодія з базою даних. Модель не містить жодних елементів відображення і нічого не знає про користувацький інтерфейс;

2) компонент View (Представлення) забезпечує візуалізацію даних для користувача та обробляє його взаємодії (натискання кнопок, заповнення форм). Його головний обов’язок відобразити інформацію, передану контролером;

3) компонент Controller (Контролер) виконує роль посередника. Він приймає запити від представлення, координує виклики методів моделі для обробки даних та повертає відповідне представлення для відображення результату.

Принцип роботи шаблону MVC зображено на рисунку 2.2.

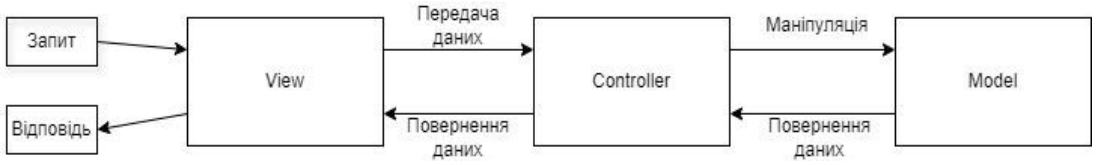

Рисунок 2.2 − Принцип роботи шаблону MVC

Шаблон MVC має наступні переваги:

1) чітке розділення відповідальності. Зміни в інтерфейсі (View) чи логіці обробки (Model) не порушують роботу інших компонентів;

2) масштабованість і гнучкість. Компоненти легко замінювати або розширювати без впливу на всю систему;

3) більшість веб-платформ мають вбудовану підтримку цього шаблону, що пришвидшує старт проєкту і зменшує час розробки.

Апаратна архітектуру комплексу побудовано на принципі взаємодії обчислювальних пристроїв у локальній обчислювальній мережі. Ця архітектура забезпечує необхідну інфраструктуру для функціонування трирівневої клієнт-серверної моделі, розділяючи відповідальність між представленням, логікою обробки та зберіганням даних.

Ключові компоненти апаратної архітектури включають:

1) клієнтські пристрої. Користувачі взаємодіють з програмним забезпеченням через інтерфейс веб-застосунку, використовуючи персональні комп’ютери або ноутбуки зі стандартними веб-браузерами. Для підтримки функціональності прокторингу, яка передбачає аналіз орієнтації обличчя, кожен клієнтський пристрій має бути оснащений вбудованою або зовнішньою камерою, здатною генерувати відеопотік для подальшої обробки. Під час

проходження тесту клієнтська частина періодично формує та передає на сервер звітні дані про поведінку користувача;

2) сервер комплексу. Це може бути персональний комп'ютер або ноутбук, на якому розміщується та виконується програмне забезпечення як сервера застосунку, так і сервера бази даних. Також можливо розміщення сервера застосунку і сервера бази даних у віртуальних машинах на сервері учбового закладу або ж у хмарному сервісі;

3) мережеве обладнання (маршрутизатор та комутатор). Маршрутизатор забезпечує з'єднання локальної обчислювальної мережі (LAN) комплексу із зовнішніми мережами, такими як Інтернет. Комутатор призначений для з'єднання численних дротових мережевих пристроїв. У сучасних маршрутизаторах інтегровано функціонал бездротової точки доступу, що дозволяє пристроям здійснювати бездротове підключення (Wi-Fi). У випадку відсутності вбудованої бездротової точки доступу, для забезпечення функціонування бездротової мережі необхідне встановлення окремого апаратного компонента – бездротової точки доступу.

Схему апаратної архітектури комплексу наведено на рисунку 2.3.

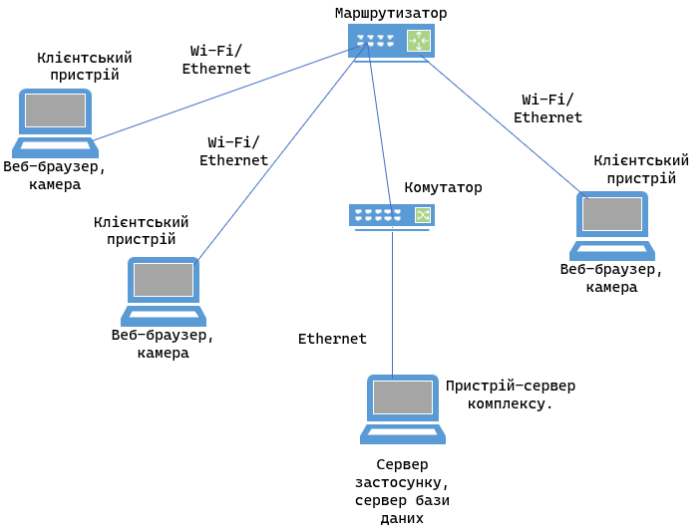

Рисунок 2.3 – Апаратна архітектура комплексу

2.2 Вибір засобів реалізації для створення комплексу

2.2.1 Мова програмування C#

Мова програмування C# − це сучасна, об’єктно-орієнтована мова програмування, розроблена Microsoft. Вона є частиною платформи .NET і має розвинений набір засобів, які добре підходять для розробки як серверних, так і десктопних застосунків [28].

Екосистема .NET, частиною якої є C#, надає доступ до численних бібліотек та фреймворків, що суттєво спрощує реалізацію складної бізнес-логіки та ефективну взаємодію з базами даних [29]. Важливою перевагою є також підтримка асинхронності, що дозволяє ефективно обробляти велику кількість одночасних запитів, підвищуючи чуйність та продуктивність застосунку.

2.2.2 Фреймворк для веб-розробки ASP.NET Core

ASP.NET Core − це сучасний, крос-платформний фреймворк з відкритим вихідним кодом від Microsoft, призначений для швидкої та ефективної побудови високопродуктивних веб-застосунків і сервісів. Він відрізняється легкістю, модульністю та можливістю розгортання на різних операційних системах [30].

Даний фреймворк оптимізований для обробки великої кількості одночасних запитів, що є критично важливим для масштабованих веб-систем. Його модульність і використання концепції middleware дозволяють легко додавати та конфігурувати функціональні блоки, такі як автентифікація або маршрутизація, роблячи архітектуру гнучкою [31]. Крім того, вбудовані механізми ін’єкції залежностей (Dependency injection) значно спрощують тестування коду та сприяють розширюваності застосунку, підтримуючи принципи чистої архітектури [32].

2.2.3 Модель програмування Razor Pages

Razor Pages − це спрощений підхід в ASP.NET Core, який дозволяє створювати веб-сторінки з вбудованим C# кодом безпосередньо у розмітці (HTML) [33]. Цей підхід орієнтовано на сторінки, що робить його хорошим варіантом для застосунків з чітко визначеною структурою. Цей засіб реалізації повністю сумісний з компонентами MVC.

2.2.4 CSS-фреймворк Bootstrap

Bootstrap −це один з найпопулярніших у світі відкритих CSS-фреймворків, розроблений для швидкої та ефективної розробки адаптивних веб-проектів [34]. Він надає великий набір готових CSS-стилів та JavaScript-компонентів, а також гнучку сіткову систему, що дозволяє створювати сучасний та функціональний користувацький інтерфейс. Наявність стандартних компонентів, таких як кнопки, навігація, форми та картки, значно прискорює процес розробки користувацького інтерфейсу та забезпечує єдиний візуальний стиль.

2.2.5 Мова програмування JavaScript

JavaScript − це високорівнева, інтерпретована мова програмування. Вона дозволяє створювати динамічні та інтерактивні елементи на веб-сторінках, які виконуються безпосередньо у веб-браузері користувача. JavaScript є головною клієнтською мовою для динамічних веб-сторінок, що робить її незамінною для створення інтерактивного користувацького досвіду, включаючи обробку подій та виконання асинхронних запитів до сервера без перезавантаження сторінки [35]. Її універсальність дозволяє реалізувати складну логіку, зокрема функціонал прокторингу.

2.2.6 Entity Framework Core

Entity Framework Core (EF Core) − це крос-платформний Object-Relational Mapper (ORM) від Microsoft [36]. ORM −це технологія, яка дозволяє розробникам взаємодіяти з базами даних використовуючи об’єкти та властивості мови програмування (наприклад C#) замість написання безпосередньо SQL-запитів. EF Core абстрагує деталі взаємодії з конкретною базою даних. Він підтримує різні підходи до роботи з даними, зокрема Code-First та Database-First, що робить його універсальним інструментом для різних сценаріїв розробки.

2.2.7 Система управління базами даних PostgreSQL

PostgreSQL − це потужна, відкрита, об’єктно-реляційна система управління базами даних (СУБД), відома своєю надійністю, розширюваністю та відповідністю стандартам SQL [37]. Вона є однією з найбільш просунутих СУБД і може обробляти великі обсяги даних та складні запити.

Вибір PostgreSQL обумовлено її доступністю, оскільки вона є безкоштовною. Система ролей і привілеїв забезпечує високий рівень безпеки, гарантуючи повний контроль доступу до даних. Гнучкість PostgreSQL дозволяє реалізовувати складні бізнес-правила завдяки підтримці користувацьких типів даних, функцій та тригерів, що є важливим для складних систем.

2.2.8 Провайдер Npgsql

Npgsql − це офіційний ADO.NET-провайдер для системи управління базами даних PostgreSQL [38]. Причиною вибору стала сумісність Npgsql з PostgreSQL, оскільки провайдер підтримує всі особливості PostgreSQL, включаючи користувацькі типи даних та складні транзакції.

2.2.9 Компіляторна платформа Roslyn

Roslyn (також відома як Microsoft.CodeAnalysis) − це компіляторна платформа, розроблена Microsoft, яка надає набір API для програмного аналізу, генерації та рефакторингу коду C# і Visual Basic. Вона дозволяє розробникам інтегрувати компілятори C# та VB.NET безпосередньо у свої програми, отримуючи доступ до синтаксичних дерев, семантичних моделей та іншої інформації про код [39].

Roslyn обрано завдяки можливості AST-аналізу (Abstract Syntax Tree), що дозволяє нормалізувати синтаксичні дерева коду та ефективно порівнювати фрагменти коду на структурному рівні [40]. Це є критично важливим для функціоналу перевірки схожості студентських робіт. Тісна інтеграція з .NET забезпечує просту інтеграцію Roslyn в екосистему C# та ASP.NET Core, полегшуючи розробку. Гнучкість платформи дозволяє реалізувати власні аналізатори коду та інструменти для рефакторингу, що розширює можливості системи.

2.2.10 Бібліотека face-api.js

Дана бібліотека призначена для виявлення облич, розпізнавання ключових точок обличчя та виявлення емоцій безпосередньо у веб-браузері. Вона використовує попередньо навчені моделі машинного навчання для виконання цих завдань [41]. Клієнтська обробка є ключовою перевагою бібліотеки, оскільки вона дозволяє виконувати аналіз відеопотоку безпосередньо в браузері користувача, зменшуючи навантаження на сервер. Це також означає, що на сервер передаються лише компактні дані про орієнтацію обличчя (наприклад, кути повороту), а не весь відеопотік, що оптимізує трафік. Висока якість детекції завдяки підтримці алгоритмів, таких як TinyFaceDetector і FaceLandmark68, дозволяє точно визначати орієнтацію голови користувача, що є важливим для реалізації прокторингу.

2.2.11 Веб-сервер IIS Express

IIS Express − це легкий, автономний веб-сервер, розроблений Microsoft. Він є мініатюрною версією повноцінного Internet Information Services (IIS) і дозволяє запускати веб-сайти без необхідності встановлення IIS на комп’ютері розробника [42]. Простота запуску та інтеграція з середовищем розробки Visual Studio дозволяють розгортати та відлагоджувати веб-застосунки без необхідності додаткових складних налаштувань. Крім того, підтримка HTTPS дозволяє тестувати захищені з’єднання без необхідності ручного налаштування складних сертифікатів, що є важливим для розробки безпечних веб-застосунків.

2.3 Проєктування функціональної структури комплексу

2.3.1 Модуль управління обліковими записами

Модуль управління обліковими записами забезпечує реєстрацію, автентифікацію та авторизацію користувачів системи. Його функціональність охоплює створення нових облікових записів, зберігання паролів у захищеному вигляді, перевірку автентичності, а також формування даних доступу відповідно до ролі користувача (студент або викладач). Здійснення входу до системи супроводжується створенням структури ідентифікаційних відомостей, що визначають повноваження користувача та використовуються в інших частинах комплексу.

З метою підвищення безпеки модуль реалізує криптографічну обробку паролів шляхом створення хешу з використанням сучасного адаптивного алгоритму. Перевірка коректності пароля при вході користувача в обліковий запис виконується шляхом повторного хешування введених даних і порівняння отриманого результату з раніше збереженим значенням.

У процесі реєстрації нових користувачів модуль забезпечує збереження основної облікової інформації у відповідних таблицях бази даних. Додатково відбувається створення запису у таблиці, що містить розширені дані про студента або викладача, із забезпеченням логічного зв’язку з основним обліковим записом. Залежно від ролі користувача формується індивідуальне підключення до бази даних.

2.3.2 Модуль обліку курсів та завдань

В цьому модулі реалізовано створення, редагування та управління навчальними курсами та завданнями в межах курсу. Викладач може створювати курси, задавати їхню назву, опис, період активності, а також додавати студентів. Для кожного курсу можна створювати завдання з описом, датою початку та дедлайном.

Модуль передбачає зручну навігацію по курсах, як для викладачів, (наприклад перегляд відправлених робіт), так і для студентів (перелік завдань, перелік тестів). Він є одним з основних елементів навчальної логіки комплексу, оскільки завдання напряму пов’язані з подачею рішень, перевіркою, оцінюванням і подальшим аналізом схожості коду.

2.3.3 Модуль студентських робіт

Модуль студентських робіт забезпечує можливість подання рішень на завдання з боку студентів, а також створює основу для подальшого їх перегляду та оцінювання викладачами. Кожна робота супроводжується інформацією про дату подання, поточний статус виконання (надіслано або перевірено), а також оцінкою, яка може бути призначена вручну викладачем. У рамках модуля реалізовано можливість редагування раніше поданого рішення до моменту його оцінювання.

Даний модуль пов’язаний з модулем аналізу схожості коду, що дає змогу викладачеві перед виставленням оцінки виконати перевірку студентського розв’язку на предмет наявності фрагментів, які потенційно запозичені з інших робіт. Крім того, передбачено зручний інтерфейс для перегляду детальної інформації щодо надісланих рішень, включно з відображенням тексту коду та зазначенням ідентифікатора користувача. Таким чином, модуль виконує роль зв’язувальної ланки між процесами виконання завдань студентами та їх експертного аналізу викладачами.

2.3.4 Модуль аналізу схожості коду

Однією з ключових функцій комплексу є можливість виявлення структурної схожості між фрагментами коду у студентських роботах. Для реалізації такої перевірки використано абстрактне синтаксичне дерево (AST), що дає змогу проводити глибинний аналіз структури програми незалежно від її текстового представлення.

Абстрактне синтаксичне дерево (AST) – це представлення програми у вигляді ієрархічної структури, де кожен вузол відповідає певному синтаксичному елементу мови програмування. Така структура відображає логіку програми незалежно від форматування, стилістичних особливостей або послідовності символів у початковому коді. Завдяки цьому AST дозволяє виконувати точний і гнучкий аналіз програмного коду, зокрема виявляти повторювані конструкції або шаблони.

Робота модуля розпочинається з вибору викладачем фрагмента коду який він хоче знайти в усіх роботах студентів. Цей код проходить форматну нормалізацію, в результаті якої усуваються символи переносу рядка, зайві пробіли й табуляції, які можуть завадити коректному аналізу. Далі цей фрагмент перетворюється у відповідну синтаксичну структуру. Якщо обраний код є частковим (наприклад, цикл або умовний оператор), то він додатково обгортається в технічну оболонку у вигляді методу всередині класу. Це

необхідно для того, щоб його можливо було коректно обробити в рамках формального синтаксису мови програмування.

Після побудови синтаксичної структури виконується процес нормалізації ідентифікаторів. Це дає змогу виключити вплив назв змінних, методів або параметрів на результат порівняння. Зокрема, імена функцій та змінних уніфікуються до стандартного вигляду, а виклики сторонніх методів абстрагуються. Таким чином, дві логічно однакові конструкції, що використовують різні позначення, стають еквівалентними у процесі аналізу.

Після нормалізації побудованої структури виконується пошук відповідних фрагментів у роботах інших студентів, що належать до того самого завдання. Для цього кожна така робота також перетворюється на абстрактне синтаксичне дерево з якого виділяються фрагменти, які мають аналогічний тип – наприклад, методи, функції або класи. Ці фрагменти проходять такий самий процес нормалізації і надалі порівнюються з еталонним фрагментом.

У випадку виявлення збігу нормалізованих структур, система вважає, що знайдено потенційно схожий фрагмент. Для кожного такого фрагмента визначається його місце у вихідному коді студентської роботи. Ця інформація передається викладачу у зручному вигляді, з можливістю перегляду фрагмента коду та визначення його меж у програмному тексті.

2.3.5 Модуль тестування

Модуль тестування реалізує функціональність створення та проведення контрольних заходів у форматі електронного тестування. Викладач має можливість формувати тести, що складаються з різних типів запитань: з одним правильним варіантом відповіді, відкритих запитань із текстовою відповіддю, а також завдань, які потребують написання програмного коду. Кожне тестування пов’язується з конкретним навчальним курсом та має встановлений час на проходження, дату створення і відповідального викладача.

Після завершення тестування, система автоматично фіксує усі надані відповіді, виконує обробку запитань із закритим типом відповідей та обчислює часткову або повну підсумкову оцінку. Результати тестування зберігаються у базі даних і доступні для перегляду як студентам, так і викладачам. Усі питання з відкритими відповідями передаються на ручну перевірку.

Даний модуль тісно взаємодіє з модулем прокторингу. В процесі тестування система отримує доступ до відеопотоку з камери користувача, проводить аналіз орієнтації обличчя та накопичує статистику щодо поведінкових показників. Це дозволяє розширити оцінку результатів тестування додатковим індикатором доброчесності, який відображає загальний рівень уважності студента під час виконання завдань.

2.3.6 Модуль прокторингу

Модуль прокторингу відповідає за контроль доброчесності під час проходження студентом тесту. Основою цього модуля є використання камери на стороні клієнта, яка активується перед початком тестування. Після активації виконується аналіз обличчя користувача з використанням нейронної мережі, яка реалізує методи розпізнавання обличчя та його ключових точок. Це дозволяє визначати орієнтацію голови та фіксувати випадки, коли обличчя тимчасово зникає з кадру.

Поведінкові дані збираються періодично протягом усього тестування. Для кожного такого циклу система отримує зображення з камери, виконує розпізнавання обличчя та будує карту його ключових точок. Серед таких точок використано положення очей, носа, підборіддя та брів. Зокрема, на основі положення очей і перенісся визначається горизонтальне відхилення (тобто поворот голови вліво або вправо), а за співвідношенням висоти носа до відстані між бровами та підборіддям обчислюється вертикальне нахилення (вгору або вниз). У разі, якщо на зображенні не вдається зафіксувати обличчя, система вважає, що користувач відвернувся повністю або залишив місце тестування.

Кожне спостереження класифікується за однією з категорій: «погляд у центр», «погляд ліворуч», «погляд праворуч», «погляд угору», «погляд вниз» або «відсутнє обличчя». Під час тесту система накопичує кількість спостережень у кожній категорії та передає їх на сервер застосунку для подальшої обробки. Після завершення тестування ці значення використовуються для розрахунку показника доброчесності – як відсоткового співвідношення кількості фіксацій, коли студент дивився прямо на екран, до загальної кількості зафіксованих спостережень.

2.4 Проєктування бази даних

В структурі бази даних комплексу виділено наступні сутності:

1) account: сутність, що описує обліковий запис користувача. Містить у собі ідентифікатор, логін, хеш паролю та роль користувача (наприклад, студент або викладач);

2) teacher: сутність, що представляє викладача. Містить ідентифікатор, ім’я, прізвище та посилання на відповідний обліковий запис у таблиці Account;

3) student: сутність, що описує студента. Містить ідентифікатор, ім’я, прізвище, дату реєстрації та зовнішній ключ на обліковий запис;

4) course: сутність, що описує навчальний курс. Містить у собі ідентифікатор курсу, назву, опис, дату початку, дату завершення та посилання на викладача, який веде курс;

5) student_in_course: проміжна сутність, що реалізує реєстрацію студентів на курси. Складається з композитного ключа (ідентифікатор студента + ідентифікатор курсу) та дати зарахування. Реалізує зв’язок багато-до-багатьох між студентами та курсами;

6) coursetask: сутність, що описує завдання в межах курсу. Містить ідентифікатор, назву, опис, дату початку, дедлайн, а також зовнішній ключ на відповідний курс;

7) studentwork: сутність, що зберігає подані студентами роботи. Містить ідентифікатор роботи, дату подачі, статус роботи, оцінку (за шкалою 1–10), текст коду, а також зовнішні ключі на студента та завдання. Також існує обмеження унікальності: кожен студент може подати лише одну роботу на конкретне завдання;

8) notification: сутність, що описує повідомлення між користувачами. Містить ідентифікатор, заголовок повідомлення, текст, тип повідомлення, дату створення, статус прочитання та зовнішні ключі на відправника і одержувача;

9) test: сутність, що представляє тест для перевірки знань студентів. Містить ідентифікатор, назву, опис, викладача-автора, курс, дату створення та тривалість проходження в хвилинах;

10) testquestion: сутність, що описує конкретне запитання в тесті. Містить ідентифікатор, текст запитання, тип питання, тест до якого належить, та кількість балів за запитання;

11) questionoption: сутність, що описує варіанти відповіді до тестових питань з варіантами відповіді. Містить ідентифікатор, текст варіанта, правильність відповіді та належність до певного запитання;

12) studenttest: сутність, що фіксує факт проходження студентом певного тесту. Містить ідентифікатор тестування, студента який проходив тест, тест, дату початку, завершення, підсумкову оцінку та індикатор доброчесності;

13) studentanswer: сутність, що фіксує відповіді студента на конкретні питання. Містить ідентифікатор відповіді студента, ідентифікатор тестування, ідентифікатор запитання, обраний варіант (якщо є), текст відповіді (для текстових запитань або запитань з кодом) і набраний бал.

Між сутностями існують наступні типи зв’язків:

1) один-до-багатьох;

2) багато-до-багатьох;

3) один-до-багатьох умовний (умовність зі сторони N-арного зв’язку).

Далі детальніше розглянуто зв’язки типу один-до-багатьох:

1) teacher та course: один викладач може вести декілька курсів, але кожен курс належить лише одному викладачу;

2) course та coursetask: один курс може містити багато завдань, але кожне завдання прив’язане до одного курсу;

3) course та test: курс може містити кілька тестів, але кожен тест належить лише одному курсу;

4) test та testquestion: тест може включати багато запитань, але кожне запитання належить лише одному тесту;

5) testquestion та questionoption: запитання з варіантами відповіді мають кілька варіантів відповіді, але кожен варіант належить лише одному запитанню;

6) coursetask та studentwork: до одного завдання можуть бути подані роботи від багатьох студентів, але кожна робота належить лише одному завданню;

7) student та studentwork: один студент може подати багато робіт, але кожна робота належить лише одному студентові;

8) student та studenttest: один студент може пройти кілька тестів, але кожне проходження пов’язане лише з одним студентом;

9) test та studenttest: один тест може бути пройдений багатьма студентами, але кожне проходження відповідає одному тесту;

10) studenttest та studentanswer: кожне проходження тесту має багато відповідей, але кожна відповідь належить лише одному проходженню тесту;

11) testquestion та studentanswer: кожне запитання може мати багато відповідей (від різних студентів), але кожна відповідь прив’язана лише до одного запитання;

12) account та notification: один користувач може надіслати або отримати багато повідомлень, але кожне повідомлення має лише одного відправника та одного одержувача.

Далі описано зв’язок типу багато-до-багатьох:

1) student та course: даний зв’язок нормалізовано за допомогою додаткової проміжної таблиці Student_In_Course яка містить ідентифікатор студента, ідентифікатор курсу до якого належить студент та дату зарахування.

Далі розглянуто зв’язок один-до-багатьох умовний (умовність зі сторони N-арного зв’язку):

1) studentanswer та questionoption: зв’язок між сутностями виникає в тому випадку, якщо запитання має варіанти відповіді. Для текстових запитань або запитань з кодом зв’язку між екземплярами сутностей не буде.

Опис сутностей та їх атрибутів наведено в додатку А.

ER-діаграму бази даних наведено на рисунку 2.4.

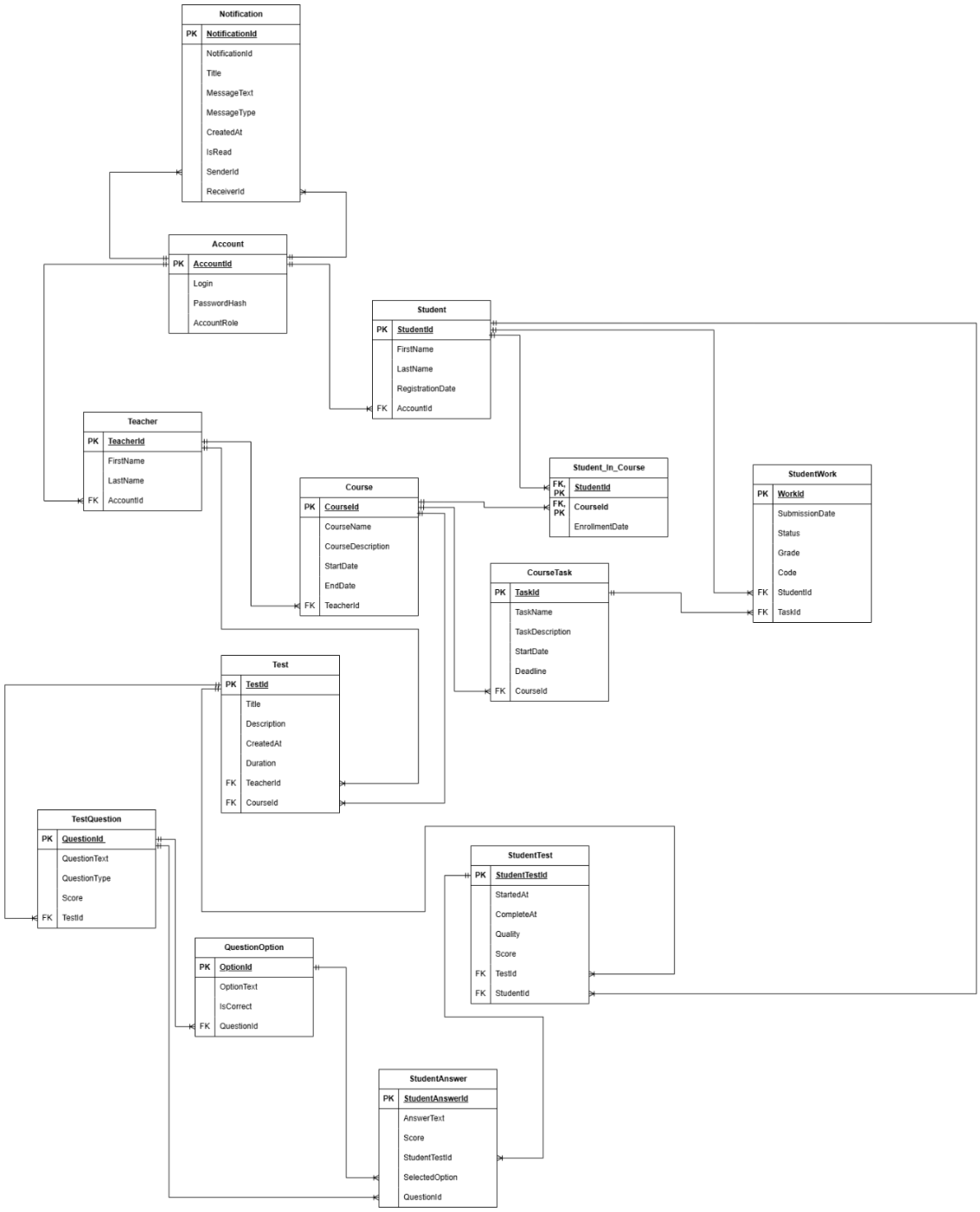

Рисунок 2.4 − ER-діаграма бази даних комплексу

2.5 Апаратне забезпечення робочого місця користувача

Для ефективної роботи з програмно-апаратним комплексом користувачеві необхідне базове робоче середовище. З боку студента передбачено використання персонального комп’ютера або ноутбука з веб-камерою (вбудованою або підключеною через USB), сучасного браузера, а також стабільного інтернет-з’єднання. Камеру в даному використано для реалізації функцій прокторингу під час проходження тестування.

Для викладачів достатньо комп’ютера з браузером для керування курсами, перевірки студентських робіт та запуску аналізу схожості коду.

З боку серверної інфраструктури необхідно забезпечити місце для розміщення програмного забезпечення веб-застосунку та бази даних. Це означає, що всі частини системи, які відповідають за обробку інформації та збереження даних, працюватимуть на спеціальному комп’ютері − сервері. Такий сервер може бути як фізичним (окремий комп’ютер, розташований у певному приміщенні), так і віртуальним, який працює у спеціальному онлайн-сервісі, що надає доступ до обчислювальних ресурсів.

3 РЕАЛІЗАЦІЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

3.1 Компоненти застосунку програмно-апаратного комплексу

Модель застосунку програмно-апаратного комплексу реалізовано у вигляді трирівневої структури, що складається з трьох основних компонентів:

1) бібліотека класів, яка реалізує рівень доступу до даних (Data Access Layer, DAL). Цей рівень відповідає за взаємодію з сервером бази даних та частково містить бізнес-логіку. Для кожної таблиці в базі даних створено відповідний клас-сутність, який відображає її структуру. На цьому рівні виконуються операції створення, зчитування, оновлення та видалення даних;

2) бібліотека класів, яка відповідає за бізнес-рівень (Business Logic Layer, BLL). Цей компонент виконує функцію посередника між рівнем даних та рівнем представлення. Тут здійснюється обробка даних, що надходять з рівня доступу до даних, їх трансформація у зручний для інтерфейсу вигляд, а також виклик логіки роботи з даними. У цьому рівні розміщено основні класи-сервіси, які реалізують логіку реєстрації, перевірки, фільтрації, обчислень та інші функції, пов’язані з основними сценаріями взаємодії користувачів з комплексом;

3) рівень представлення, реалізований у вигляді ASP.NET Core застосунку. Цей рівень відповідає за безпосередню взаємодію з користувачем. Запити, що надходять від клієнта, передаються до відповідних контролерів, де відбувається їх обробка за допомогою класів-сервісів бізнес-рівня. Контролери формують відповіді, які повертаються у вигляді представлень або результатів дій.

3.2 Реалізація управління обліковими записами

Модуль облікових записів реалізує функціональність автентифікації та реєстрації користувачів у системі. Його реалізовано на принципах трирівневої

архітектури, де кожен рівень відповідає за окрему сферу обробки даних. Усі запити користувача обробляються на рівні контролера AccountController, який виконує роль координатора між інтерфейсом та бізнес-логікою.

Процес реєстрації реалізовано у методі RegisterStudent (див. Додаток Б, лістинг Б.1). Після первинної перевірки введених даних у формі реєстрації, контролер ініціює виклики відповідних сервісів бізнес-рівня. Метод CreateStudent (див. Додаток Б, лістинг Б.2) формує об’єкти сутностей на основі DTO та здійснює виклик репозиторію. У свою чергу, метод CreateStudent (див. Додаток Б, лістинг Б.3) у рівні доступу до даних ініціює виклик збереженої процедури бази даних register_student (див. Додаток Л, лістинг Л.1), яка одночасно додає запис про користувача до таблиці users.Account та до таблиці Student.

Для забезпечення безпеки зберігання паролів у системі реалізовано механізм хешування, що ґрунтується на використанні алгоритму PBKDF2 з функцією SHA-256 [43]. Метод HashPassword (див. Додаток Б, лістинг Б.4) генерує випадкову сіль, створює хеш пароля та зберігає його разом із сіллю у вигляді єдиної послідовності байтів. Результат кодується у формат Base64 перед записом у базу даних. Такий підхід значно ускладнює можливість підбору пароля у разі витоку бази.

Під час автентифікації використовується метод Login (див. Додаток Б, лістинг Б.5), який приймає введені користувачем облікові дані, виконує пошук облікового запису за логіном, а потім викликає метод VerifyPassword (див. Додаток Б, лістинг Б.6). Цей метод відновлює сіль із збереженого хешу, обчислює хеш для наданого пароля та порівнює обидва результати побайтно. У разі успішної перевірки система ініціює створення користувацької сесії, де зберігається інформація про ідентифікатор користувача, його роль для взаємодії з базою даних та ідентифікатор пов’язаної сутності (викладача або студента). Методи для реєстрації вчителя мають аналогічну структуру, окрім того що зберігають інформацію в таблицях users.Account (Користувач) та Teacher (Викладач).

3.3 Реалізація обліку курсів та завдань

Модуль обліку курсів та завдань відповідає за створення, редагування та відображення навчальних курсів, а також управління завданнями, пов’язаними з цими курсами. Усі дії цього модуля реалізовано у вигляді окремих методів у контролері TeacherController, які забезпечують відповідну взаємодію користувача з бізнес-логікою та базою даних. Саме цей модуль забезпечує структурування навчального процесу, формування логіки навчальних одиниць, а також зв'язок між викладачем, курсом та студентами.

Створення нового курсу реалізовано за допомогою методу CreateCourse (див. Додаток В, лістинг В.1). Після отримання введених викладачем даних з веб-форми, виконується формування об’єкта CourseDTO, який містить інформацію про назву курсу, його опис, дату початку, дату завершення та ідентифікатор викладача. Цей об’єкт передається на рівень бізнес-логіки, де відбувається трансформація до відповідної сутності бази даних. У подальшому ця сутність передається до рівня доступу до даних, де зберігається у відповідній таблиці бази даних.

Для перегляду детальної інформації про курс реалізовано метод CourseFullInfo (див. Додаток В, лістинг В.2). Він викликає відповідний метод сервісу, отримує об’єкт CourseDTO, трансформує його у модель представлення, після чого передає у вигляді відповіді на сторону клієнта. Додатково, для оптимізації доступу, об’єкт курсу серіалізується у формат JSON і тимчасово зберігається у сесії користувача [44].

У межах цього модуля також реалізовано логіку створення та редагування завдань, що закріплюються за курсом. Метод CreateCourseTask (див. Додаток В, лістинг В.3) дозволяє викладачу додати нове завдання, вказавши його назву, опис, дату початку та дедлайн. Подібно до створення курсу, дані завдання проходять через рівень бізнес-логіки, де вони трансформуються у внутрішні об’єкти, що відповідають структурі таблиці

CourseTask (див. Додаток К, лістинг К.1). Згодом ці об’єкти передаються на збереження до бази даних.

Для відображення інформації про конкретне завдання використовується метод ShowTaskInfo (див. Додаток В, лістинг В.4), що дозволяє викладачу переглянути усі наявні атрибути завдання. У разі необхідності зміни інформації про завдання, викликається метод EditCourseTask (див. Додаток В, лістинг В.5), який забезпечує редагування даних. Після перевірки на валідність, об’єкт з новими значеннями передається до відповідного сервісу. Далі відбувається оновлення сутності на рівні зберігання даних, що гарантує актуальність інформації у системі.

3.4 Реалізація модулю студентських робіт

Функціональність, пов’язана з подачею та обробкою студентських рішень, реалізована як у контролері StudentController, так і у TeacherController. Ключовою задачею цього модуля є забезпечення можливості студенту завантажити своє рішення у вигляді програмного коду, а викладачу – переглядати, оцінювати або перевіряти ці роботи на наявність збігів.

Метод ShowStudentWork (див. Додаток Г, лістинг Г.1) призначений для відображення інформації про вже подане рішення конкретного студента на конкретне завдання. Завдання передається у вигляді серіалізованого об’єкта, що зберігає всі потрібні атрибути. Після десеріалізації даних та ідентифікації користувача, система отримує студентську роботу через бізнес-рівень і формує модель представлення, яка повертається у відповідь клієнту.

Надсилання нового рішення реалізовано у методі SubmitWork (див. Додаток Г, лістинг Г.2). Студент передає текст програми у вигляді рядка, який зберігається разом з поточними датою та часом відправлення, ідентифікатором завдання та статусом «відправлено». Усі ці дані інкапсулюються в об’єкт StudentworkDTO, який передається в бізнес-рівень, де обробляється, трансформується у сутність і зберігається в базі даних. У

якості реакції система формує повідомлення про подачу роботи та перенаправляє користувача на сторінку перегляду завдання.

У разі потреби студент може оновити раніше подане рішення за допомогою методу UpdateWork (див. Додаток Г, лістинг Г.3). Новий код надсилається разом з ідентифікатором відповідної роботи. Після обробки та валідації оновлені дані замінюють попередні у базі даних, а викладач отримує нове повідомлення про зміну.

Окрему частину модуля становить інтерфейс перегляду студентських робіт з боку викладача. Метод SubmittedWorks (див. Додаток Г, лістинг Г.4) дозволяє отримати повний перелік робіт, надісланих на певне завдання. Дані передаються з рівня бізнес-логіки, трансформуються у модель представлення і зберігаються у сесії для подальшого перегляду. Це дозволяє викладачу зручно оперувати інформацією та оцінювати подані рішення в інтерактивному режимі.

Для перегляду детальної інформації про окрему роботу реалізовано метод ShowSubmittedWorkDetails (див. Додаток Г, лістинг Г.5). Після десеріалізації об’єкта, що містить усі ключові атрибути роботи (дата подачі, вміст коду, статус, оцінка тощо), сторінка з детальною інформацією відкривається у вигляді окремого представлення.

Оцінювання студентських робіт викладачем реалізовано за допомогою методу SetGrade (див. Додаток Г, лістинг Г.6). Після отримання ідентифікатора конкретної роботи та виставленої оцінки, система здійснює пошук відповідного об’єкта на рівні бізнес-логіки. Після оновлення значення оцінки у DTO-об’єкті, інформація передається в репозиторій, де зміни зберігаються у базі даних. Завершення процесу супроводжується перенаправленням викладача на головну сторінку, що дозволяє оперативно переходити до оцінювання інших робіт. Реалізація методу підтримує перевірку достовірності запиту, що забезпечується атрибутом ValidateAntiForgeryToken, і гарантує безпечне оновлення критичних даних у системі.

3.5 Реалізація аналізу схожості коду

Однією з ключових функцій комплексу є виявлення структурної схожості між студентськими роботами. Для цього реалізовано окремий модуль, що дозволяє викладачеві обрати фрагмент коду з будь-якої студентської роботи та здійснити пошук подібних фрагментів у роботах інших студентів того самого завдання. На відміну від простого порівняння текстів, система аналізує синтаксичну структуру програмного коду, що дозволяє виявляти схожість навіть при зміні форматування чи перейменуванні змінних.

Механізм аналізу реалізовано на базі платформи Roslyn – компілятора .NET з відкритим кодом, що надає засоби для побудови, модифікації та аналізу синтаксичних дерев. У межах розробленого рішення використано клас CSharpSyntaxTree, який є частиною Roslyn. За допомогою цього компонента вхідний код компілюється у вигляді синтаксичного дерева, яке потім аналізується для визначення його типу: метод, локальна функція, клас, структура або фрагмент інструкцій (наприклад цикл чи умовний оператор).

Основну логіку порівняння реалізовано у методі FindSimilarities (див. Додаток Д, лістинг Д.1) класу TeacherController. Цей метод ініціює процес порівняння, передаючи код у метод CodeFormatNormalizer (див. Додаток Д, лістинг Д.2) для базової нормалізації форматування. Після цього метод FindSimilarities з класу CodeComparer (див. Додаток Д, лістинг Д.3) проводить глибокий аналіз шляхом порівняння абстрактних синтаксичних дерев.

Перш ніж здійснити порівняння, код нормалізується методом GetNormalizedMethod (див. Додаток Д, лістинг Д.4). Тут синтаксичне дерево створюється за допомогою CSharpSyntaxTree.ParseText(), після чого здійснюється обхід дерева для визначення типу конструкції та формування її уніфікованого представлення. При необхідності фрагмент коду автоматично «обгортається» у метод або клас, щоб забезпечити його коректну обробку.

Нормалізація імен змінних, методів та параметрів виконується за допомогою класу IdentifierNormalizer, який успадковується від базового класу

CSharpSyntaxRewriter (частина Roslyn). У цьому класі перевизначено методи, які відповідають за обробку різних вузлів синтаксичного дерева:

1) методи VisitMethodDeclaration та VisitLocalFunctionStatement нормалізують імена методів і локальних функцій (див. Додаток Д, лістинг Д.5, Д.6);

2) метод VisitParameter виконує уніфікацію параметрів методів (див. Додаток Д, лістинг Д.7);

3) метод VisitIdentifierName обробляє всі ідентифікатори, замінюючи їх на стандартні (див. Додаток Д, лістинг Д.8);

4) метод VisitVariableDeclarator уніфікує імена локальних змінних (див. Додаток Д, лістинг Д.9).

Після нормалізації всі роботи студентів проходять через такий самий процес синтаксичного аналізу. Система виконує обхід кожного вузла дерева в пошуку структур, які відповідають нормалізованому базовому коду. Якщо збіг знайдено, метод FindMatchingNodeInOriginalTree (див. Додаток Д, лістинг Д.10) дозволяє запам’ятати номера рядків знайденого фрагменту коду для його подальшого підсвічування в представленні.

Для демонстрації прикладу аналізу коду на схожість взято задачу обчислення числа Фібоначчі. Цю задачу можна вирішити декількома варіантами, але в даному прикладі фокус зосереджено на двох методах – рекурсивний та ітеративний. Нижче наведено програмну реалізацію знаходження числа Фібоначчі рекурсивним методом з трьома різними варіантами найменування об’єктів у коді:

```
public static long FibonacciFind(int a)
        {
            if (a <= 0) return 0;

            if (a == 1) return 1;

            return FibonacciFind(a - 1) + FibonacciFind(a - 2);

        }
```

Лістинг 3.1 – Рекурсивний метод знаходження числа Фібоначчі, версія 1

```
public static long FindRecursive(int number)
        {
            if (number <= 0) return 0;
            if (number == 1) return 1;
            return FindRecursive(number - 1) +
FindRecursive(number - 2);
        }
```

Лістинг 3.2 – Рекурсивний метод знаходження числа Фібоначчі, версія 2

```
public static long FibonacciRec(int n)
        {
            if (n <= 0) return 0;
            if (n == 1) return 1;
            return FibonacciRec(n - 1) + FibonacciRec(n - 2);
        }
```

Лістинг 3.3 – Рекурсивний метод знаходження числа Фібоначчі, версія 3

Кожна з цих трьох версій реалізації рекурсивного методу має різні ідентифікатори об’єктів, але структура коду (з якою як раз працює Roslyn) залишилась незмінною, через що реалізований алгоритм аналізу коду зможе знайти співпадіння у цих трьох варіантах реалізації. Також для демонстрації коректності роботи алгоритму в рамках завдання знаходження числа Фібоначчі створено реалізацію ітеративного методу двома версіями:

```
public static long FibonacciSearch(int num)
    {
        if (num <= 0) return 0;
        if (num == 1) return 1;
        long a = 0, b = 1;
        for (int i = 2; i <= num; i++)
        {
            long t = a + b;
            a = b;
            b = t;
        }
        return b;
    }
```

Лістинг 3.4 – Ітеративний метод знаходження числа Фібоначчі, версія 1

```
public static long FibonacciIterative(int n)
    {
        if (n <= 0) return 0;

        if (n == 1) return 1;

        long a = 0, b = 1;

        for (int i = 2; i <= n; i++)
        {
            long temp = a + b;

            a = b;

            b = temp;
        }

        return b;
    }
```

Лістинг 3.5 – Ітеративний метод знаходження числа Фібоначчі, версія 2

На сторінці з інформацією про роботу студента викладач може обрати фрагмент коду який він хоче знайти в усіх роботах в рамках завдання курсу та отримати результати збігу в завданнях (див. рис. 3.1).

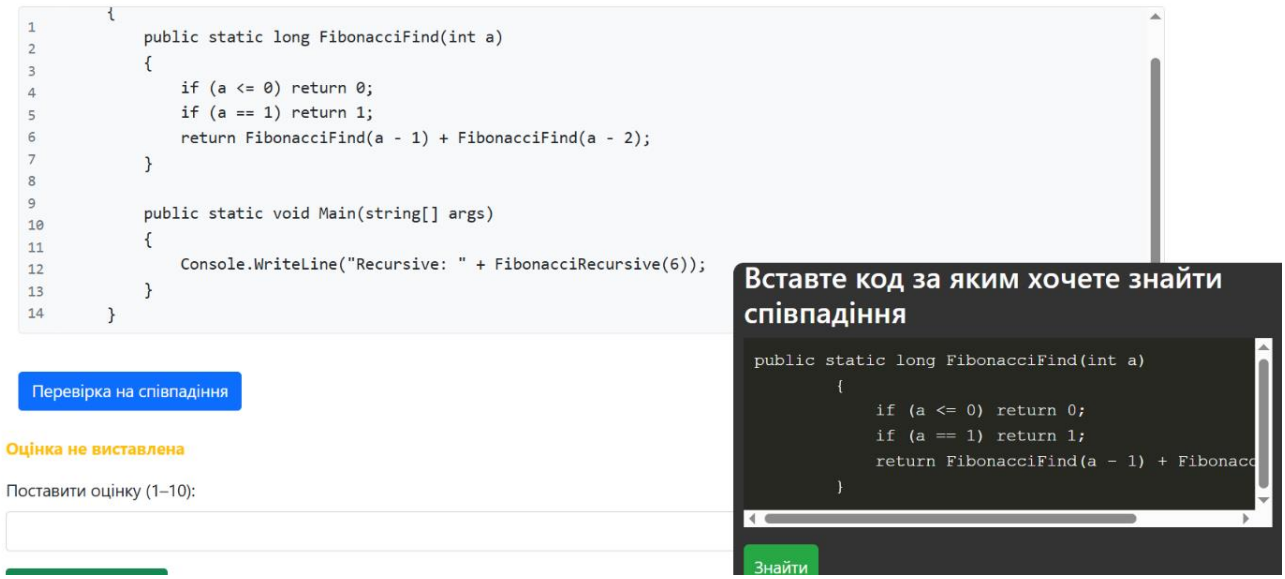

Рисунок 3.1 – Сторінка вибору фрагменту коду для виявлення збігів у роботах студентів

В даному випадку необхідно знайти роботи в яких реалізовано рекурсивний метод знаходження числа Фібоначчі. Результат роботи аналізу коду на співпадіння зображено на рисунках 3.2 та 3.3.

Рисунок 3.2 – Перелік робіт в яких виявлено схожість (частина 1)

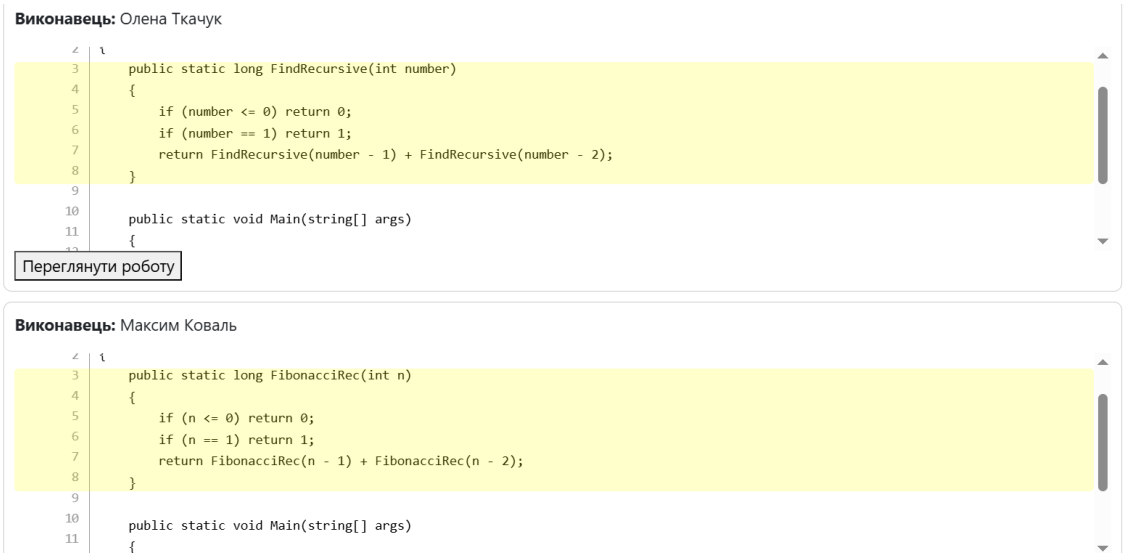

Рисунок 3.3 – Перелік робіт в яких виявлено схожість (частина 2)

В результаті виконання аналізу отримано три роботи в яких знайшовся обраний раніше фрагмент коду а сам знайдений фрагмент виділяється.

Результати демонструють принцип роботи реалізованого аналізу схожості коду. Аналіз схожості відбувається на основі саме структури коду, а не його тексту.

3.6 Реалізація модулю тестування

Модуль тестування реалізує функціональність створення, проходження та перевірки тестових завдань. Цей компонент дозволяє викладачам формувати тести з різними типами запитань (з варіантами відповідей, текстові питання або питання які пов’язані з кодом), які студенти можуть проходити у визначений час. Усі дії, пов’язані зі створенням тестів, надсиланням відповідей, збереженням результатів та оцінюванням відбуваються через взаємодію контролерів, бізнес-рівня та рівня доступу до даних.

Для створення нового тесту використано метод CreateTest (див. Додаток Е, Лістинг Е.1) контролера TeacherController. Отримані від викладача дані спочатку проходять перевірку на коректність, після чого перетворюються в об’єкт передачі даних (DTO), а згодом трансформуються у відповідну сутність бази даних. Збереження відбувається через відповідні сервіси, що забезпечують логіку взаємодії з репозиторіями та базою даних. Приклад створення тесту зображено на рисунках 3.4 та 3.5.

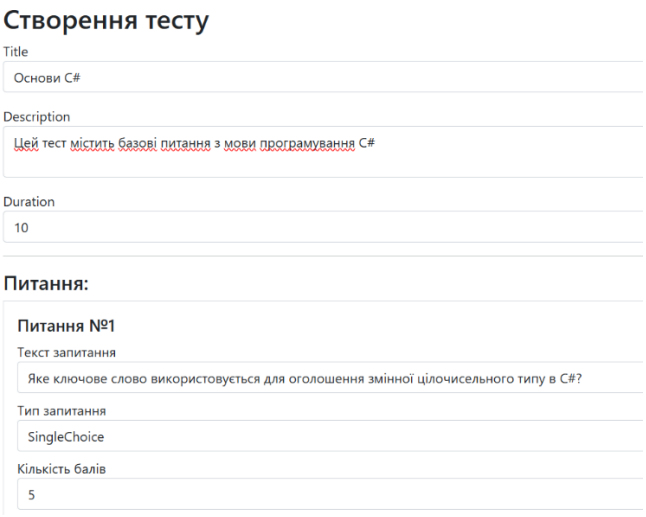

Рисунок 3.4 – Створення тесту викладачем

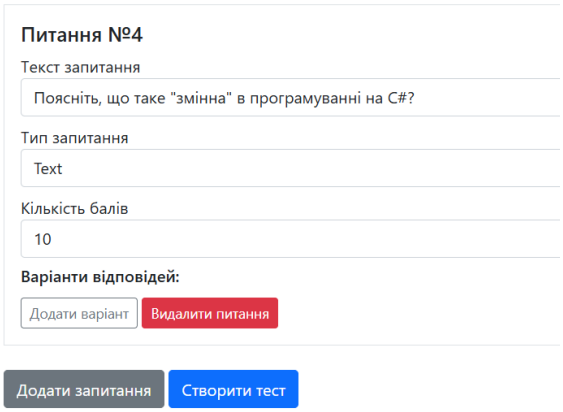

Рисунок 3.5 – Створення тесту викладачем (частина 2)

Викладачі мають змогу переглядати перелік усіх створених тестів для певного курсу через метод CourseTests (див. Додаток Е, Лістинг Е.2). Цей метод отримує тестові записи з бази даних, перетворює їх у модель представлення та передає у відповідне представлення для відображення. Аналогічно, за допомогою методу TestInfo (див. Додаток Е, Лістинг Е.3) студент може переглянути загальну інформацію про тест перед початком його проходження.

Запуск тесту реалізовано методом TakeTest (див. Додаток Е, Лістинг Е.4). Після запуску тесту створюється об’єкт StudenttestVM, який містить ідентифікатори тесту та студента, а також відмітку часу початку проходження. Дані цього об’єкта передаються у представлення, де студент взаємодіє із запитаннями (див. рис. 3.6).

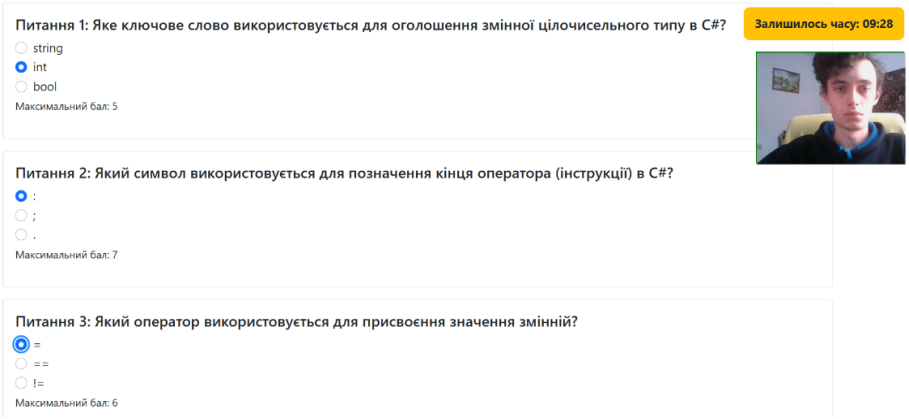

Рисунок 3.6 – Процес проходження студентом тесту

Після завершення тесту відповіді обробляються у методі SubmitTest (див. Додаток Е, Лістинг Е.5). Усі зібрані дані (включно з відповідями на запитання) формуються у відповідну DTO-модель і зберігаються в базі даних через метод SubmitStudentTestAsync (див. Додаток Е, Лістинг Е.6). Оцінювання відбувається автоматично для закритих запитань. Для цього використано окремий алгоритм, який аналізує правильність обраних варіантів, порівнюючи їх з правильними відповідями в тесті. Нараховані бали заносяться у відповідні сутності, а підсумкова оцінка встановлюється методом SetScoreToStudentTest (див. Додаток Е, Лістинг Е.7).

Результати проходження тесту виводяться за допомогою методу TestResults (див. Додаток Е, Лістинг Е.8). У цьому методі система підраховує кількість правильних відповідей на запитання з варіантами відповіді, формує підсумкову оцінку та передає дані в представлення (див. рис. 3.7). Якщо в тесті є запитання відкритого типу, система автоматично додає повідомлення про необхідність ручної перевірки з боку викладача.

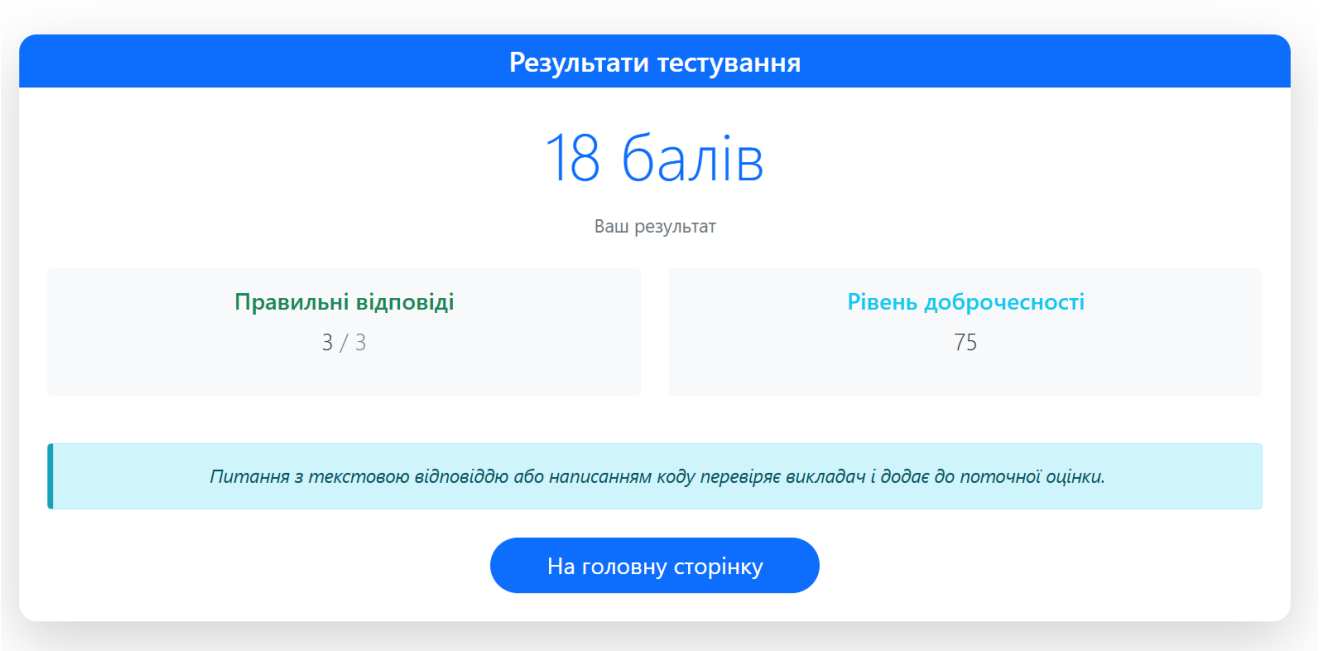

Рисунок 3.7 – Вивід результатів тестування

3.7 Реалізація прокторингу

Модуль прокторингу призначений для автоматичного нагляду за поведінкою студента під час проходження тесту з метою оцінювання доброчесності виконання. Його робота починається із запиту дозволу на доступ до камери користувача, що реалізовано у представленні сторінки тестування. Для цього використано інтерфейс navigator.mediaDevices.getUserMedia (див. Додаток Ж, Лістинг Ж.1), який надає змогу отримати відеопотік з камери через браузер за допомогою WebRTC API [46].

Збір даних починається після того, як відео починає відтворюватися. Розпізнавання обличчя відбувається кожні 3 секунди із використанням бібліотеки face-api.js, яка є обгорткою над TensorFlow.js для спрощення задач розпізнавання обличчя та виявлення його орієнтації [45]. Для забезпечення цієї функціональності використано попередньо навчені моделі, що зберігаються у директорії wwwroot/models проєкту ASP.NET Core. Моделі відповідають за виявлення обличчя та локалізацію ключових точок (очі, ніс, щелепа тощо).

Функція estimateFaceOrientation (див. Додаток Ж, Лістинг Ж.2) аналізує координати орієнтирів (обраних точок обличчя), щоб визначити положення голови користувача у просторі. Орієнтація класифікується за горизонтальним (RIGHT, LEFT, CENTER) та вертикальним (UP, DOWN, CENTER) напрямками. Якщо обличчя не виявлено на кадрі, подія фіксується як NO_FACE.

Далі розглянуто варіанти даних які надсилаються на сервер застосунку при різних положеннях голови. При фронтальному положенні обличчя (див. рис. 3.8) дані матимуть такий вигляд: { CENTER, CENTER }. Перший параметр вказує напрямок по горизонталі, а другий по вертикалі.

Рисунок 3.8 – Фронтальний напрямок обличчя

При відхиленні обличчя праворуч (див. рис. 3.9) дані матимуть наступний вигляд: { RIGHT, CENTER }.

Рисунок 3.9 – Відхилення обличчя праворуч

При відхиленні обличчя ліворуч (див. рис. 3.10) дані матимуть наступний вигляд: { LEFT, CENTER }.

Рисунок 3.10 – Відхилення обличчя ліворуч

При нахилі обличчя вниз (див. рис. 3.11) дані матимуть наступний вигляд: { CENTER, DOWN }.

Рисунок 3.11 – Нахил обличчя вниз

При нахилі обличчя вгору (див. рис. 3.12) дані матимуть наступний вигляд: { CENTER, UP }.

Рисунок 3.12 – Нахил обличчя вгору

Усі описані визначені стани накопичуються у словнику orientationCounts, що зберігається у сесії користувача. Для ідентифікації користувача формується унікальний ключ, який включає його AccountId і Studenttestid. Таким чином, навіть у випадку одночасного тестування кількох користувачів, сесійні дані із орієнтацією обличчя не змішуються, оскільки ключ чітко пов’язаний з ідентифікатором кожного тесту й користувача.

Збір даних реалізовано у методі RecordOrientation (див. Додаток Ж, Лістинг Ж.3), куди асинхронно надсилаються JSON-об’єкти з визначеним напрямком голови і нахилом. Метод розпаковує отриману інформацію, додає нові значення до відповідного словника та повторно зберігає його у сесію.

Після завершення тесту ці накопичені дані використовуються для обчислення індикатора доброчесності. У методі SubmitTest (див. Додаток Ж, Лістинг Ж.4) отримується словник counts, у якому зберігається кількість

кожного з зафіксованих станів. Рівень доброчесності визначається як відсоткове співвідношення кількості «чесних» розпізнавань до загальної кількості усіх спроб фіксації положення обличчя. Це значення множиться на 100, щоб отримати відсоток, і обмежується зверху значенням 100 для уникнення помилок.

Таким чином, модуль прокторингу забезпечує базову, але ефективну оцінку поведінкової доброчесності студентів під час тестування. Його перевагою є використання відкритих моделей розпізнавання, що дозволяє уникнути надмірного навантаження на сервер і не потребує додаткових драйверів чи програмного забезпечення на стороні користувача.

3.8 Реалізація бази даних

Реалізація бази даних є невід'ємною складовою проєкту, що забезпечує збереження, цілісність і доступність усіх необхідних даних для роботи комплексу. Реалізацію виконано за допомогою системи керування базами даних PostgreSQL, що надає широкі можливості для створення типів даних, таблиць, процедур, а також управління правами доступу. Далі розглянуто приклади створення таких елементів бази даних як домени, таблиці, функції а також створення ролі.

У межах реалізації створено спеціалізований домен, що уніфікує допустимі значення для поля статусу студентської роботи. Такий домен дозволяє централізовано контролювати перелік допустимих значень, підвищуючи узгодженість і стійкість схеми даних. Код створення домену наведено в лістингу 3.6.

```
create domain WorkStatus as varchar(30)
default 'Submitted'
check (value in ('Submitted', 'Evaluated'));
```

Лістинг 3.6 – Код створення домену WorkStatus

У наведеному прикладі створено домен WorkStatus (Статус роботи), який обмежує значення лише до двох допустимих: Submitted (відправлено) та Evaluated (оцінено). Це дозволяє у подальшому використовувати його як тип стовпця у відповідних таблицях, що описують статуси робіт.

Далі розглянуто приклад створення таблиці StudentWork, що зберігає дані про студентські роботи. Вона містить усі необхідні атрибути для ідентифікації роботи, її стану, пов’язаних студентів і завдань, а також сам код як результат виконання завдання. Код створення таблиці наведено у лістингу 3.7.

```
create table StudentWork
(
     WorkId serial primary key,
     SubmissionDate date not null,
     Status WorkStatus not null,
     Grade integer check (Grade between 1 and 10 or Grade is
null),
     Code text not null,
     StudentId integer not null,
     foreign key (StudentId) references Student(StudentId) on
delete restrict on update cascade,
     TaskId integer not null,
     foreign key (TaskId) references CourseTask(TaskId) on delete
restrict on update cascade,
     unique (StudentId, TaskId)
);
```

Лістинг 3.7 – Код створення таблиці StudentWork

Ця таблиця зберігає роботи студентів із зазначенням дати подання, статусу, оцінки (за наявності), а також посиланням на відповідного студента і завдання. Обмеження цілісності забезпечують коректність зв’язків між таблицями, а обмеження унікальності для комбінації зовнішніх ключів StudentId та TaskId не дозволяє одному студенту надіслати декілька робіт на одне й те саме завдання. Запити на створення доменів та таблиць бази даних наведено у додатку К.

Далі розглянуто приклад створення функції яка повертає всі вхідні повідомлення для конкретного користувача. Функції та процедури допомагають організувати складну логіку в рамках маніпуляції даними. Код створення процедури наведено в лістингу 3.8.

```
create or replace function GetReceivedMessages(p_accountId int)
returns table
(
     NotificationId int,
     Title text,
     MessageText text,
     MessageType MessageType,
     CreatedAt timestamp,
     IsRead boolean,
     SenderLogin text,
     SenderFirstName text,
     SenderLastName text,
     ReceiverLogin text,
     ReceiverFirstName text,
     ReceiverLastName text
)
language sql
security definer
as
$$   select n.NotificationId, n.Title, n.MessageText,
n.MessageType, n.CreatedAt, n.IsRead, sender_acc.Login as
SenderLogin, coalesce(sender_teacher.FirstName,
sender_student.FirstName), coalesce(sender_teacher.LastName,
sender_student.LastName), receiver_acc.Login,
coalesce(receiver_teacher.FirstName,
receiver_student.FirstName), coalesce(receiver_teacher.LastName,
receiver_student.LastName)
     from Notification n join users.Account sender_acc on
n.SenderId = sender_acc.AccountId
     left join Teacher sender_teacher on sender_acc.AccountId =
sender_teacher.AccountId
     left join Student sender_student on sender_acc.AccountId =
sender_student.AccountId
     join users.Account receiver_acc on receiver_acc.AccountId =
n.ReceiverId
     left join Teacher receiver_teacher on receiver_acc.AccountId
= receiver_teacher.AccountId
     left join Student receiver_student on receiver_acc.AccountId
= receiver_student.AccountId
     where n.ReceiverId = p_accountId
     order by n.CreatedAt desc;
$$;
```

Лістинг 3.8 – Код створення функції GetReceivedMessages

Ця функція дістає дані з таблиці сповіщень та облікових записів, формуючи повну інформацію про кожне повідомлення разом з відправником і одержувачем. Завдяки ключовому слову security definer, функція виконується з правами користувача, який її створив, що дозволяє забезпечити безпечне отримання інформації. Запити на створення функцій та процедур наведено у додатку Л.

Окрему увагу в реалізації бази даних приділено системі управління доступом. Ролі користувачів є одним із засобів забезпечення розмежування прав у PostgreSQL. Кожна роль визначає набір дозволів, які користувач має щодо окремих об’єктів бази даних. Це дає змогу не лише забезпечити доступ до даних, а й дотриматися принципу найменших привілеїв. Приклад створення роді викладача наведено в лістингу 3.9.

```
create role Teacher with login password 'teacher';
grant insert, select, update, delete on Student_In_Course to
Teacher;
grant insert, select, update, delete on Notification to Teacher;
grant insert, select, update, delete on Course to Teacher;
grant insert, select, update, delete on CourseTask to Teacher;
grant select, update on public.Teacher to Teacher;
grant select, update on StudentWork to Teacher;
grant select on Student to Teacher;
grant insert, select, update, delete on Test to Teacher;
grant insert, select, update, delete on TestQuestion to Teacher;
grant insert, select, update, delete on QuestionOption to
Teacher;
grant select, update on StudentTest to Teacher;
grant select, update on StudentAnswer to Teacher;

grant usage, select on sequence course_courseid_seq,
coursetask_taskid_seq, notification_notificationid_seq to
Teacher; -- для послідовностей первинного ключа
grant usage, select on sequence test_testid_seq to Teacher;
grant usage, select on sequence testquestion_questionid_seq to
Teacher;
grant usage, select on sequence questionoption_optionid_seq to
Teacher;
```

Лістинг 3.9 – Створення ролі teacher та надання ролі привілеїв

У даному скрипті створено роль teacher, якій призначаються права на виконання базових операцій над таблицями, що безпосередньо пов’язані з діяльністю викладача в системі. Також роль отримує доступ до послідовностей, які відповідають за автоматичну генерацію значень первинних ключів для потрібних таблиць. Така структура дозволяє гнучко керувати правами доступу та мінімізувати ризик несанкціонованих змін у базі даних. Запити на створення ролей та надання їм привілеїв наведено у додатку М.

3.9 Висновки до третього розділу

У даному розділі описано реалізацію функціональних компонентів програмно-апаратного комплексу: управління обліковими записами, облік курсів і завдань, обробка студентських робіт, тестування, аналіз схожості коду та прокторинг. В рамках реалізації використано трирівневу архітектуру застосунку, що дозволило ефективно розділити логіку взаємодії з користувачем, бізнес-процеси та роботу з базою даних.

Особливу увагу приділено обробці облікових даних, а також структурному порівнянню коду для виявлення можливих збігів між роботами студентів. У рамках модуля тестування впроваджено базові механізми прокторингу, що забезпечують додатковий контроль за поведінкою користувача під час проходження оцінювання. Всі компоненти інтегруються у єдину систему за рахунок бази даних, структура якої реалізована з урахуванням обмежень цілісності та розмежуванням прав доступу для окремих ролей.

ВИСНОВКИ

У ході виконання роботи проаналізовано особливості організації освітнього процесу в умовах цифровізації, зокрема ті аспекти, що стосуються автоматизації перевірки завдань, формування оцінювання, організації тестування, перевірка робіт на співпадіння та забезпечення академічної доброчесності. Визначено ключові функціональні компоненти системи, які необхідно реалізувати для підтримки ефективної взаємодії між студентами та викладачами у межах освітнього середовища.

Запропоновано архітектуру програмно-апаратного комплексу, що базується на трирівневій клієнт-серверній моделі з розподілом функціональності між рівнем представлення, бізнес-логіки та рівнем зберігання даних. Для реалізації серверної частини застосовано мову програмування C# і платформу ASP.NET Core, що забезпечує гнучке управління маршрутами, обробкою запитів і розширюваністю логіки. Візуальну частину побудовано за допомогою Razor Pages із застосуванням CSS-фреймворку Bootstrap, що дозволяє створювати адаптивні й зручні інтерфейси. Для доступу до бази даних використано ORM-рішення Entity Framework Core, що спрощує управління сутностями. Як систему управління базами даних обрано PostgreSQL. Для аналізу коду на структурну схожість використано компіляторну платформу Roslyn, яка дозволяє працювати з абстрактними синтаксичними деревами. Модуль прокторингу реалізовано за допомогою бібліотеки face-api.js, що виконує аналіз обличчя в режимі реального часу.

У межах розробленої функціональної структури реалізовано модулі для управління обліковими записами, реєстрації курсів і завдань, роботи з індивідуальними студентськими рішеннями, організації тестування, аналізу робіт на співпадіння та здійснення прокторингу. Забезпечено захищене зберігання облікових даних користувачів з використанням криптографічних алгоритмів хешування, реалізовано розмежування прав доступу до ресурсів

системи. Усі навчальні дії пов’язуються з відповідним обліковим записом, що дозволяє фіксувати активність, зберігати історію взаємодії з елементами курсу та контролювати процес навчання.

Система підтримує створення викладачами курсів та завдань, з можливістю додавання опису, термінів виконання та контролю виконання. Студенти можуть надсилати свої роботи, які зберігаються в системі, відображаються викладачам і можуть бути оцінені. Додатково передбачено можливість виявлення структурних збігів між фрагментами коду студентських рішень на основі побудови абстрактного синтаксичного дерева. Це дозволяє виявити плагіат навіть за умов зміни форматування чи найменувань.

Розроблено гнучку систему тестування з підтримкою різних типів запитань. Відповіді зберігаються й автоматично перевіряються. Особливу увагу приділено розробці системи прокторингу. Під час проходження тесту користувачу пропонується активувати камеру, після чого здійснюється регулярна фіксація орієнтації голови. Обробка поведінкових ознак відбувається на основі аналізу просторових координат ключових точок обличчя, що дозволяє визначити напрям погляду або зафіксувати його відсутність. Дані передаються на сервер, накопичуються в сеансі користувача та використовуються для обчислення коефіцієнта доброчесності на основі співвідношення кількості фіксацій з прямим поглядом до загальної кількості фіксацій.

Реалізовано структуровану базу даних з визначенням ролей користувачів і наданням необхідних привілеїв на рівні СУБД. Запропонована структура забезпечує цілісність даних та підтримує логіку взаємодії між об’єктами освітнього процесу.

Перспективи розвитку комплексу передбачають розширення системи тестування з підтримкою адаптивного оцінювання, вдосконалення модуля прокторингу через впровадження аналізу міміки та емоцій, підтримка аналізу робіт які написані на інших популярних мовах програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bradley V.M. Learning Management System (LMS) Use with Online Instruction // International Journal of Technology in Education. – 2021. – Vol. 4, No. 1, 68–92.
2. Sommerville I. Software Engineering. 10th ed. – Pearson Education, 2015. – 792 p.
3. Nielson F., Nielson H.R., Hankin C. Principles of Program Analysis. – Springer, 2005. – 299 p.
4. Кобзев І.В., Тимченко О.Л. Системи прокторінгу для проведення онлайн іспитів // Мультимедійні технології в освіті та інших сферах діяльності: матеріали Міжнародної науково-практичної конференції. – Київ: НАУ, 2024. – С. 245–248.
5. Chua S.S., Lee Y.H., Goh W.L., Tan J.K., Lim K.W., et al. AI-based Proctoring Systems: A Review // Journal of Educational Technology Systems. – 2021. – Vol. 49, №3. – P. 1–17.
6. MoodleDocs [Електронний ресурс]. – Режим доступу: https://docs.moodle.org/500/en/Main_page – Дата звернення: 17.03.2025
7. PHP [Електронний ресурс]. – Режим доступу: https://www.php.net/manual/en – Дата звернення: 17.03.2025
8. MySQL [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc – Дата звернення: 17.03.2025
9. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/17/index.html – Дата звернення: 17.03.2025
10. About Google Classroom [Електроний ресурс]. – Режим доступу: https://edu.google.com/workspace-for-education/products/classroom – Дата звернення: 17.03.2025
11. Google Workspace for Education [Електронний ресурс]. – Режим доступу: https://edu.google.com/workspace-for-education/editions/overview – Дата звернення: 17.03.2025

12. Canvas LMS − Instructure Community [Електроний ресурс]. – Режим доступу: https://community.canvaslms.com/t5/Canvas/ct-p/canvas – Дата звернення: 17.03.2025
13. Canvas LMS API [Електронний ресурс]. – Режим доступу: https://community.canvaslms.com/t5/Canvas-Change-Log/2025-API-and-CLI-Change-Log/ta-p/626858 – Дата звернення: 17.03.2025
14. Canvas Reviews 2025: Pros & Cons, Ratings & more [Електронний ресурс]. – Режим доступу: https://elearningindustry.com/directory/elearning-software/canvas/reviews – Дата звернення: 17.03.2025
15. MOSS. A System for Detecting Software Similarity [Електронний ресурс]. – Режим доступу: https://theory.stanford.edu/~aiken/moss – Дата звернення: 19.03.2025
16. Schleimer S., Wilkerson D.S., Aiken A. Winnowing: Local Algorithms for Document Fingerprinting [Електронний ресурс]. – Режим доступу: https://theory.stanford.edu/~aiken/publications/papers/sigmod03 – Дата звернення: 19.03.2025
17. JPlag [Електронний ресурс]. – Режим доступу: https://github.com/jplag/jplag – Дата звернення: 19.03.2025
18. Aho A.V., Lam M.S., Sethi R., Ullman J.D. Compilers: Principles, Techniques, and Tools. 2nd ed. – Pearson Education, 2007. – 1000 p.
19. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 3rd ed. – MIT Press, 2009. – 1312 p.
20. Codequiry – Usage Documentation [Електронний ресурс]. – Режим доступу: https://codequiry.com/usage/docs – Дата звернення: 19.03.2025
21. ProctorU [Електронний ресурс]. – Режим доступу: https://www.proctoru.com – Дата звернення: 19.03.2025
22. ProctorExam [Електронний ресурс]. – Режим доступу: https://www.proctorexam.com – Дата звернення: 19.03.2025

23. What is computer vision? – IBM [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/computer-vision – Дата звернення: 19.03.2025
24. Honorlock Online Proctoring [Електронний ресурс]. – Режим доступу: https://honorlock.com – Дата звернення: 19.03.2025
25. Honorlock Search and Destroy [Електронний ресурс]. – Режим доступу: https://honorlock.com/search-and-destroy-gppc – Дата звернення: 19.03.2025
26. What is a 3-tier application architecture? Definition and Examples – vFunction [Електронний ресурс]. – Режим доступу: https://vfunction.com/blog/3-tier-application – Дата звернення: 22.03.2025
27. ASP.NET MVC Pattern [Електронний ресурс]. – Режим доступу: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc – Дата звернення: 22.03.2025
28. C# language documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp – Дата звернення: 23.03.2025
29. What is .NET? [Електронний ресурс]. – Режим доступу: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet – Дата звернення: 23.03.2025
30. ASP.NET Core [Електронний ресурс]. – Режим доступу: https://dotnet.microsoft.com/en-us/apps/aspnet – Дата звернення: 23.03.2025
31. ASP.NET Core Middleware [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-9.0 – Дата звернення: 23.03.2025
32. Dependency injection in ASP.NET Core [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-9.0 – Дата звернення: 23.03.2025

33. Introduction to Razor Pages in ASP.NET Core [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio – Дата звернення: 23.03.2025
34. Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com – Дата звернення: 23.03.2025
35. MDN Web Docs - JavaScript [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript – Дата звернення: 23.03.2025
36. Entity Framework Core [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core – Дата звернення: 24.03.2025
37. PostgreSQL 17.5 Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/current/index.html – Дата звернення: 24.03.2025
38. Npgsql Documentation [Електронний ресурс]. – Режим доступу: https://www.npgsql.org/doc – Дата звернення: 24.03.2025
39. The .NET Compiler Platform SDK [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/csharp/roslyn-sdk – Дата звернення: 27.03.2025
40. Parr T. Language implementation patterns: Techniques, strategies, and idioms for building domain-specific languages. Raleigh: Pragmatic Bookshelf, 2010. – 364 p.
41. face-api.js Documentation [Електронний ресурс]. – Режим доступу: https://justadudewhohacks.github.io/face-api.js/docs/index.html – Дата звернення: 27.03.2025
42. IIS Express Overview [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview – Дата звернення: 27.03.2025
43. PBKDF2 - Practical Cryptography for Developers [Електронний ресурс]. – Режим доступу: https://cryptobook.nakov.com/mac-and-key-derivation/pbkdf2 – Дата звернення: 11.04.2025

44. JSON [Електронний ресурс]. – Режим доступу: https://www.json.org/json-en.html – Дата звернення: 18.04.2025
45. TensorFlow.js [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/js?hl=en – Дата звернення: 24.04.2025
46. WebRTC [Електронний ресурс]. – Режим доступу: https://webrtc.org/getting-started/overview – Дата звернення: 02.05.2025

ДОДАТОК А

Опис сутностей бази даних

Таблиця А.1 – Опис сутностей бази даних

Ім’я атрибуту	Призначення атрибуту	Обмеження
Teacher (Викладач)		
TeacherId	Ідентифікатор викладача	Первинний ключ
FirstName	Ім’я викладача	Не порожнє
LastName	Прізвище викладача	Не порожнє
AccountId	Обліковий запис викладача	Зовнішній ключ для зв’язку з таблицею Account(AccountId)
Course (Курс)		
CourseId	Ідентифікатор курсу	Первинний ключ
CourseName	Назва курсу	Не порожнє
CourseDescription	Опис курсу	Не порожнє
StartDate	Дата початку курсу	Не порожнє
EndDate	Дата завершення курсу	Не порожнє, не може бути меншим (раніше) за StartDate
TeacherId	Викладач курсу	Зовнішній ключ для зв’язку з таблицею Teacher(TeacherId)
CourseTask (Завдання курсу)		
TaskId	Ідентифікатор завдання	Первинний ключ
TaskName	Назва завдання	Не порожнє
TaskDescription	Опис завдання	Не порожнє
StartDate	Дата початку виконання завдання	Не порожнє

Продовження таблиці А.1

Ім’я атрибуту	Призначення атрибуту	Обмеження
Deadline	Кінцевий термін виконання	Не порожнє, не може бути меншим (раніше) за StartDate
CourseId	Курс до якого належить завдання	Зовнішній ключ для зв’язку з таблицею Course(CourseId)
Student (Студент)		
StudentId	Ідентифікатор студента	Первинний ключ
FirstName	Ім’я студента	Не порожнє
LastName	Прізвище студента	Не порожнє
RegistrationDate	Дата реєстрації	Не порожнє
AccountId	Обліковий запис студента	Зовнішній ключ для зв’язку з таблицею Account(AccountId)
StudentWork (Робота студента)		
WorkId	Ідентифікатор роботи	Первинний ключ
SubmissionDate	Дата завантаження роботи	Не порожнє
Status	Статус роботи	Не порожнє, належить до множини (‘Submitted’, ‘Evaluated’)
Grade	Оцінка	Ціле число від 1 до 10, або null
Code	Код роботи	Не порожнє
StudentId	Студент який здав роботу	Зовнішній ключ для зв’язку з таблицею Student(StudentId), разом з TaskId утворюють унікальну комбінацію

Продовження таблиці А.1

Ім’я атрибуту	Призначення атрибуту	Обмеження
TaskId	Завдання до якого належить робота	Зовнішній ключ для зв’язку з таблицею CourseTask(TaskId), разом з StudentId утворюють унікальну комбінацію
Student_In_Course (Студент в певному курсі)		
StudentId	Ідентифікатор студента	Зовнішній ключ для зв’язку з таблицею Student(StudentId), разом з CourseId утворюють первинний ключ
CourseId	Ідентифікатор курсу	Зовнішній ключ для зв’язку з таблицею Course(CourseId), разом з StudentId утворюють первинний ключ
EnrollmentDate	Дата зарахування на курс	Не порожнє
Account (Обліковий запис)		
AccountId	Ідентифікатор облікового запису	Первинний ключ
Login	Логін користувача	Не порожнє, унікальне
PasswordHash	Хеш пароля	Не порожнє
AccountRole	Роль облікового запису	Не порожнє, належить до множини (‘student’, ‘teacher’)
Notification (Сповіщення)		
NotificationId	Ідентифікатор повідомлення	Первинний ключ
Title	Заголовок	Не порожнє
MessageText	Текст повідомлення	Відсутні

Продовження таблиці А.1

Ім’я атрибуту	Призначення атрибуту	Обмеження
MessageType	Тип повідомлення	Не порожнє, належить до множини (‘Default message’, ‘Work submission’, ‘Work update’, ‘Work evaluated’, ‘Course invitation’)
CreatedAt	Дата та час створення повідомлення	Не порожнє, за замовчуванням відповідає поточній даті та часу
IsRead	Атрибут для перевірки сповіщення на прочитання	Не порожнє, за замовчуванням false
SenderId	Відправник повідомлення	Зовнішній ключ для зв’язку з таблицею Account(AccountId)
ReceiverId	Отримувач повідомлення	Зовнішній ключ для зв’язку з таблицею Account(AccountId)
Test (Тест)		
TestId	Ідентифікатор тесту	Первинний ключ
Title	Заголовок тесту	Не порожнє
Description	Опис тесту	Відсутні
CreatedAt	Дата створення тесту	Не порожнє, за замовчуванням відповідає поточній даті та часу
Duration	Тривалість тесту в хвилинах	Не порожнє, має бути більше за 0

Продовження таблиці А.1

Ім’я атрибуту	Призначення атрибуту	Обмеження
TeacherId	Викладач який створив тест	Зовнішній ключ для зв’язку з таблицею Teacher(TeacherId)
CourseId	Курс до якого відноситься тест	Зовнішній ключ для зв’язку з таблицею Course(CourseId)
TestQuestion (Питання тесту)		
QuestionId	Ідентифікатор питання	Первинний ключ
QuestionText	Текст питання	Не порожнє
QuestionType	Тип питання	Не порожнє, належить до множини (‘SingleChoice’, ‘Text’, ‘Code’)
Score	Бал питання	Не порожнє, за замовчуванням 1
TestId	Тест до якого відноситься питання	Зовнішній ключ для зв’язку з таблицею Test(TestId)
QuestionOption (Варіант відповіді на питання)		
OptionId	Ідентифікатор варіанту відповіді	Первинний ключ
OptionText	Текст варіанту відповіді	Не порожнє
IsCorrect	Позначення правильної відповіді	Не порожнє, за замовчуванням false
QuestionId	Питання до якого відноситься варіант відповіді	Зовнішній ключ для зв’язку з таблицею TestQuestion(QuestionId)

Продовження таблиці А.1

Ім’я атрибуту	Призначення атрибуту	Обмеження
StudentTest (Проходження студентом тесту)		
StudentTestId	Ідентифікатор проходження тесту	Первинний ключ
StartedAt	Дата та час початку проходження тесту	Не порожнє, за замовчуванням відповідає поточній даті та часу
CompleteAt	Дата та час завершення тесту	Не порожнє
Quality	Якість виконання тесту (рівень доброчесності)	Не порожнє, за замовчуванням 0
Score	Оцінка за тест	Не порожнє
StudentId	Студент який пройшов тест	Зовнішній ключ для зв’язку з таблицею Student(StudentId)
TestId	Тест який склав студент	Зовнішній ключ для зв’язку з таблицею Test(TestId)
StudentAnswer (Відповідь на питання)		
StudentAnswerId	Ідентифікатор відповіді	Первинний ключ
AnswerTest	Текст відповіді (Для текстових питань чи питань з кодом)	Відсутні
Score	Оцінка за відповідь	Не порожнє
StudentTestId	Тест до якого відноситься відповідь	Зовнішній ключ для зв’язку з таблицею StudentTest(StudentTestId)
QuestionId	Питання до якого відноситься відповідь	Зовнішній ключ для зв’язку з таблицею TestQuestion(QuestionId)
SelectedOptionId	Обраний варіант відповіді (Для питань з варіантами відповіді)	Зовнішній ключ для зв’язку з таблицею QuestionOption(OptionId)

ДОДАТОК Б

Код реалізації управління обліковими записами

```
[HttpPost]
public async Task<IActionResult>
RegisterStudent(StudentViewModel studentViewData)
{
    string connectionString =
configuration.GetConnectionString("DefaultConnection");
    using (var connection = new
NpgsqlConnection(connectionString))
    {
        connection.Open();

        if (!ModelState.IsValid)
        {
            return View();
        }

        UserDbContextInfo serviceRole = new UserDbContextInfo
        {
            UserId = 0,
            ConnectionString =
configuration.GetConnectionString("DefaultConnection")
        };

        int serviceId = 0;
        connectionService.AddInfo(serviceRole);

        AccountDTO accountCheck =
accountService.GetByLogin(studentViewData.Email,
serviceId.ToString());

        if (accountCheck != null)
        {
            throw new ValidationException("User with this email
is already exist");
        }

        AccountDTO accountDTO = new AccountDTO
        {
            Login = studentViewData.Email,
            Passwordhash = studentViewData.Password,
            Accountrole = "student"
        };

        StudentDTO studentDTO = new StudentDTO
        {
```

Лістинг Б.1 – Код методу для реєстрації студента

```
            Firstname = studentViewData.Firstname,
            Lastname = studentViewData.Lastname,
            Registrationdate =
DateOnly.FromDateTime(DateTime.Now)
        };

        accountService.CreateStudent(accountDTO, studentDTO,
serviceId.ToString());

        UserDbContextInfo studentInfo = new UserDbContextInfo
        {
            UserId = 1,
            ConnectionString =
configuration.GetConnectionString("StudentConnection")
        };

        connectionService.AddInfo(studentInfo);

        studentViewData.Accountrole = "student";

        accountCheck =
accountService.GetByLogin(studentViewData.Email,
serviceId.ToString());

        studentDTO =
accountService.GetStudentByAccountId(accountCheck.Accountid,
studentInfo.UserId.ToString());

        var claims = new List<Claim>
        {

            new Claim(ClaimTypes.Name, studentViewData.Email),
            new Claim(ClaimTypes.Role, "student"),
            new Claim("AccountId",
accountCheck.Accountid.ToString()),
            new Claim("RoleId", studentDTO.Studentid.ToString())
        };
        var identity = new ClaimsIdentity(claims, "CookieAuth");

ClaimsPrincipal claimsPrincipal = new ClaimsPrincipal(identity);

        await HttpContext.SignInAsync("CookieAuth",
claimsPrincipal);
        connection.Close();
        NpgsqlConnection.ClearPool(connection);

        return RedirectToAction("StudentMainPage", "Student");
    }
}
```

Лістинг Б.1, аркуш 2

```
public void CreateStudent(AccountDTO accountDTO, StudentDTO
studentDTO, string connectionId)
{
    Account account = new Account
    {
        Login = accountDTO.Login,
        Passwordhash =
passwordService.HashPassword(accountDTO.Passwordhash),
        Accountrole = "student"
    };

    Student student = new Student
    {
        Firstname = studentDTO.Firstname,
        Lastname = studentDTO.Lastname,
        Registrationdate = studentDTO.Registrationdate
    };

    accountRepository.CreateStudent(account, student,
connectionId);
}
```

Лістинг Б.2 – Код методу для створення класів сутностей студента та користувача

```
public void CreateStudent(Account newAccount, Student
newStudent, string connectionId)
{
    UserDbContextInfo userDbContextInfo =
connectionService.GetInfo(connectionId.ToString()); // service
or repository ?
    string connectionString =
userDbContextInfo.ConnectionString; // service or repository ?
    var Dbbuilder = new
DbContextOptionsBuilder<ApplicationDbContext>(); // service or
repository ?
    Dbbuilder.UseNpgsql(connectionString); // service or
repository ?
    ApplicationDbContext dbContext = new
ApplicationDbContext(Dbbuilder.Options, connectionString);

    try
    {
        using (NpgsqlConnection connection = new
NpgsqlConnection(connectionString))
        {
            connection.Open();
```

Лістинг Б.3 – Код виклику процедури для внесення нового студента в потрібні таблиці

```
using (NpgsqlCommand command = new
NpgsqlCommand("register_student", connection))
            {
                command.CommandType =
System.Data.CommandType.StoredProcedure;

command.Parameters.AddWithValue("student_first_name",
newStudent.Firstname);

command.Parameters.AddWithValue("student_last_name",
newStudent.Lastname);

command.Parameters.AddWithValue("hashed_password",
newAccount.Passwordhash);
                command.Parameters.AddWithValue("student_login",
newAccount.Login);

command.Parameters.AddWithValue("registration_date",
newStudent.Registrationdate);
                command.ExecuteNonQuery();
            }
        }
    }
    catch
    {
        throw new ArgumentException("Something went wrong");
    }
}
```

Лістинг Б.3, аркуш 2

```
public string HashPassword(string password)
{
    byte[] salt = RandomNumberGenerator.GetBytes(16);

    var hash = new Rfc2898DeriveBytes(password, salt, 100_000,
HashAlgorithmName.SHA256);

    byte[] hashBytes = hash.GetBytes(32);

    byte[] hashWithSaltBytes = new byte[48];
    Array.Copy(salt, 0, hashWithSaltBytes, 0, 16);

    Array.Copy(hashBytes, 0, hashWithSaltBytes, 16, 32);

    return Convert.ToBase64String(hashWithSaltBytes);
}
```

Лістинг Б.4 – Код хешування паролю користувача

```
[HttpPost]
public async Task<IActionResult> Login(AccountViewModel
loginData)
{
    string connectionString =
configuration.GetConnectionString("DefaultConnection");
    using (var connection = new
NpgsqlConnection(connectionString))
    {
        connection.Open();
        if (!ModelState.IsValid)
        {
            return View();
        }
        UserDbContextInfo serviceRole = new UserDbContextInfo
        {
            UserId = 0,
            ConnectionString =
configuration.GetConnectionString("DefaultConnection")
        };
        int serviceId = 0;
        connectionService.AddInfo(serviceRole);
        var email = loginData.Login;
        var user = accountService.GetByLogin(email,
serviceId.ToString());
        if (user == null)
        {
            string userNotFound = "Користувача з такими даними
не існує.";
            return View();
        }
        bool passwordValidation =
accountService.CheckPassword(loginData.Password,
user.Passwordhash);
        if (!passwordValidation)
        {
            string wrongPassword = "Невірний пароль";
            return View();
        }

        int domainId = -1;

        switch(user.Accountrole)
        {
            case "teacher":

                UserDbContextInfo teacherInfo = new
UserDbContextInfo
                {
                    UserId = user.Accountid,
```

Лістинг Б.5 – Код методу для автентифікації користувача

```
                    ConnectionString =
configuration.GetConnectionString("TeacherConnection")
                };
                connectionService.AddInfo(teacherInfo);

                TeacherDTO teacherDTO =
accountService.GetTeacherByAccountId(user.Accountid,
user.Accountid.ToString());
                domainId = teacherDTO.Teacherid;

                break;
            case "student":

                UserDbContextInfo adminInfo = new
UserDbContextInfo
                {
                    UserId = user.Accountid,
                    ConnectionString =
configuration.GetConnectionString("StudentConnection")
                };
                connectionService.AddInfo(adminInfo);

                StudentDTO studentDTO =
accountService.GetStudentByAccountId(user.Accountid,
user.Accountid.ToString());
                domainId = studentDTO.Studentid;

                break;
        }

        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, user.Login),
            new Claim(ClaimTypes.Role, user.Accountrole),
            new Claim("AccountId", user.Accountid.ToString()),
            new Claim("RoleId", domainId.ToString())
        };
        var identity = new ClaimsIdentity(claims, "CookieAuth");
        ClaimsPrincipal claimsPrincipal = new
ClaimsPrincipal(identity);
        await HttpContext.SignInAsync("CookieAuth",
claimsPrincipal);
        connection.Close();
        NpgsqlConnection.ClearPool(connection);

        if (user.Accountrole == "teacher")
        {
            return RedirectToAction("TeacherMainPage",
"Teacher");
        }
```

Лістинг Б.5, аркуш 2

```
        else if (user.Accountrole == "student")
        {
            return RedirectToAction("StudentMainPage",
"Student");
        }
        else
        {
            throw new Exception("Undefined role");
        }
    }
}
```

Лістинг Б.5, аркуш 3

```
public bool VerifyPassword(string password, string
hashedPassword)
{
    byte[] hashWithSaltBytes =
Convert.FromBase64String(hashedPassword);
    byte[] salt = new byte[16];
    Array.Copy(hashWithSaltBytes, 0, salt, 0, 16);

    var hash = new Rfc2898DeriveBytes(password, salt, 100_000,
HashAlgorithmName.SHA256);
    byte[] hashBytes = hash.GetBytes(32);

    for (int i = 0; i < 32; i++)
    {
        if (hashWithSaltBytes[i + 16] != hashBytes[i])
            return false;
    }
    return true;
}
```

Лістинг Б.6 – Код методу перевірки паролю на співпадіння

ДОДАТОК В

Код реалізації обліку курсів та завдань

```
[HttpPost]
public IActionResult CreateCourse(CourseViewModel
courseViewModel)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    CourseDTO courseDTO = new CourseDTO
    {
        Coursename = courseViewModel.Coursename,
        Coursedescription = courseViewModel.Coursedescription,
        Startdate = courseViewModel.Startdate,
        Enddate = courseViewModel.Enddate,
        Teacherid = Int32.Parse(roleId)
    };

    courseService.CreateCourse(courseDTO, connectionId);

    return View();
}
```

Лістинг В.1 – Код методу створення курсу

```
public IActionResult CourseFullInfo(string courseIdString)
{

    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    int courseId = int.Parse(courseIdString);

    CourseDTO courseInfoDTO =
courseService.GetCourseFullInfo(courseId, connectionId);

    CourseFullInfoVM courseFullInfo =
mapper.Map<CourseFullInfoVM>(courseInfoDTO);
```

Лістинг В.2 – Код методу для отримання повної інформації про курс

```
    var options = new JsonSerializerOptions
    {
        Encoder =
System.Text.Encodings.Web.JavaScriptEncoder.UnsafeRelaxedJsonEsc
aping
    };

    HttpContext.Session.SetString("CourseInfo",
System.Text.Json.JsonSerializer.Serialize(courseFullInfo,
options));

    return View(courseFullInfo);
}
```

Лістинг В.2, аркуш 2

```
[HttpPost]
public IActionResult CreateCourseTask(CoursetaskViewModel
taskVM)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;
    CoursetaskDTO taskDTO = new CoursetaskDTO
    {
        Taskname = taskVM.Taskname,
        Taskdescription = taskVM.Taskdescription,
        Startdate = taskVM.Startdate,
        Deadline = taskVM.Deadline,
        Courseid = taskVM.Courseid
    };
    courseTaskService.CreateCourseTask(taskDTO, connectionId);
    return View();
}
```

Лістинг В.3 – Код методу створення завдання курсу

```
[HttpGet]
public IActionResult ShowTaskInfo(string taskIdString)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
```

Лістинг В.4 – Код виводу інформації про завдання

```
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;
    CoursetaskDTO taskDTO =
courseTaskService.GetCourseTask(int.Parse(taskIdString),
connectionId);
    CoursetaskFullInfoVM taskVM =
mapper.Map<CoursetaskFullInfoVM>(taskDTO);
    return View(taskVM);
}
```

Лістинг В.4, аркуш 2

```
[HttpPost]
public IActionResult EditCourseTask(CoursetaskFullInfoVM
editedTask)
{
    if(!ModelState.IsValid)
    {
        foreach (var state in ModelState)
        {
            Console.WriteLine($"{state.Key} => {string.Join(",
", state.Value.Errors.Select(e => e.ErrorMessage))}");
        }
        return View(editedTask);
    }
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;
    CoursetaskDTO existingTask =
courseTaskService.GetCourseTask(editedTask.Taskid,
connectionId);
    if (existingTask == null)
    {
        throw new ValidationException("The existing object of
course task is not found. Failed to update course task");
    }
    existingTask.Taskname = editedTask.Taskname;
    existingTask.Taskdescription = editedTask.Taskdescription;
    existingTask.Startdate = editedTask.Startdate;
    existingTask.Deadline = editedTask.Deadline;
    courseTaskService.UpdateCourseTask(existingTask,
connectionId);
    return RedirectToAction("ShowTaskInfo", new { taskIdString =
existingTask.Taskid });
}
```

Лістинг В.5 – Код методу редагування завдання

ДОДАТОК Г

Код реалізації модулю студентських робіт

```
[HttpGet]
public IActionResult ShowStudentWork(string taskJson)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    var model =
Newtonsoft.Json.JsonConvert.DeserializeObject<CoursetaskFullInfo
VM>(taskJson);

    TempData["TaskId"] = model.Taskid;

    var studentWorkDTO =
studentWorkService.Get(Int32.Parse(roleId), model.Taskid,
connectionId);

    var studentWorkVM =
mapper.Map<StudentworkViewModel>(studentWorkDTO);

    return View(studentWorkVM);
}
```

Лістинг Г.1 – Код методу знаходження роботи студента

```
[HttpPost]
public IActionResult SubmitWork(string code)
{

    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    if (!TempData.TryGetValue("TaskId", out var taskIdObj) ||
taskIdObj == null)
    {
        return BadRequest("TaskId not found.");
    }
```

Лістинг Г.2 – Код публікації роботи студентом до завдання

```
    int taskId = Convert.ToInt32(taskIdObj);

    StudentworkDTO createdWorkDTO = new StudentworkDTO()
    {
        Submissiondate = DateOnly.FromDateTime(DateTime.Now),
        Status = "Submitted",
        Code = code,
        Studentid = Int32.Parse(roleId),
        Taskid = taskId
    };

    studentWorkService.Create(createdWorkDTO, connectionId);

    SubmittedWorkMessage(taskId);

    return RedirectToAction("ShowTaskInfo", new { taskIdString =
taskId.ToString() });
}
```

Лістинг Г.2, аркуш 2

```
[HttpPost]
public IActionResult UpdateWork(string code, string taskJson)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    if (!TempData.TryGetValue("TaskId", out var taskIdObj) ||
taskIdObj == null)
    {
        return BadRequest("TaskId not found.");
    }
    int taskId = Convert.ToInt32(taskIdObj);
    var taskDTO =
Newtonsoft.Json.JsonConvert.DeserializeObject<StudentworkDTO>(ta
skJson);
    taskDTO.Code = code;
    studentWorkService.Update(taskDTO, connectionId);

    UpdatedWorkMessage(taskDTO.Taskid);

    return RedirectToAction("ShowTaskInfo", new { taskIdString =
taskId.ToString() });
}
```

Лістинг Г.3 – Код методу оновлення роботи студентом

```
[HttpGet]
public IActionResult SubmittedWorks(string taskJson)
{
    var user = HttpContext.User;

    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    var model =
Newtonsoft.Json.JsonConvert.DeserializeObject<CoursetaskFullInfo
VM>(taskJson);

    var worksDTO =
courseTaskService.SubmittedWorks(model.Taskid, connectionId);

    List<StudentworkViewModel> worksVM =
mapper.Map<List<StudentworkViewModel>>(worksDTO);

    model.Studentworks = worksVM;

    HttpContext.Session.SetString("CourseTaskFullInfoVM",
JsonConvert.SerializeObject(model));

    return View(model);
}
```

Лістинг Г.4 – Код методу для отримання завантажених робіт певного завдання

```
public IActionResult ShowSubmittedWorkDetails(string
studentWorkJson)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;


    var studentWork =
Newtonsoft.Json.JsonConvert.DeserializeObject<StudentworkViewMod
el>(studentWorkJson);


    return View(studentWork);
}
```

Лістинг Г.5 – Код для отримання детальної інформації про роботу студента

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult SetGrade(int workId, int grade)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    var studentWorkDTO = studentWorkService.Get(workId,
connectionId);

    studentWorkDTO.Grade = grade;

    studentWorkService.Update(studentWorkDTO, connectionId);

    return RedirectToAction("TeacherMainPage");
}
```

Лістинг Г.6 – Код методу оцінювання роботи студента

ДОДАТОК Д

Код реалізації аналізу схожості студентських робіт

```
[HttpPost]
public IActionResult FindSimilarities(string baseCode, string
tasksJson)
{
    var modelJson =
HttpContext.Session.GetString("CourseTaskFullInfoVM");
    var model =
JsonConvert.DeserializeObject<CoursetaskFullInfoVM>(modelJson);

    var studentsWorksDTO =
mapper.Map<List<StudentworkDTO>>(model.Studentworks);

    ViewData["TaskModel"] = model;

    string normalizedBaseCode =
codeComparer.CodeFormatNormalizer(baseCode);

    List<StudentworkDTO> similarities =
codeComparer.FindSimilarities(normalizedBaseCode,
studentsWorksDTO);

    var similaritiesVM =
mapper.Map<List<StudentworkViewModel>>(similarities);

    return View("ShowMatchedTasks" ,similaritiesVM);
}
```

Лістинг Д.1 – Код основного методу аналізу коду на співпадіння та виводу результатів

```
public string CodeFormatNormalizer(string code)
{
    code = code.Replace("\r\n", "\n");
    code  =  string.Join("\n",  code.Split('\n').Select(line  =>
line.Trim()));
    code   =   System.Text.RegularExpressions.Regex.Replace(code,
@"\s+", " ");

    code = code.Replace("\u00A0", "");
    code = code.Trim();
    return code;
}
```

Лістинг Д.2 – Код нормалізації форматування коду

```
public List<StudentworkDTO> FindSimilarities(string baseCode,
List<StudentworkDTO> tasks)
{

    NormalizedCodeResult basicCodeNormalized =
GetNormalizedMethod(baseCode);
    string normalizedCode = basicCodeNormalized.NormalizedCode;
    string basicCodeType = basicCodeNormalized.CodeType;

    List<StudentworkDTO> matchingTasks = new
List<StudentworkDTO>();

    foreach (StudentworkDTO task in tasks)
    {

        SyntaxTree altTree =
CSharpSyntaxTree.ParseText(task.Code);
        SyntaxNode altRoot = altTree.GetRoot();

        IEnumerable<SyntaxNode> candidateNodes = basicCodeType
switch
        {

            "method" => altRoot.DescendantNodes()
               .OfType<MethodDeclarationSyntax>()
               .Cast<SyntaxNode>()
               .Concat(altRoot.DescendantNodes()

.OfType<LocalFunctionStatementSyntax>()
                              .Cast<SyntaxNode>()),

            "localFunction" => altRoot.DescendantNodes()

.OfType<MethodDeclarationSyntax>()
                                      .Cast<SyntaxNode>()

.Concat(altRoot.DescendantNodes()

.OfType<LocalFunctionStatementSyntax>()

.Cast<SyntaxNode>()),

            "class" =>
altRoot.DescendantNodes().OfType<ClassDeclarationSyntax>().Cast<
SyntaxNode>(),
            "struct" =>
altRoot.DescendantNodes().OfType<StructDeclarationSyntax>().Cast
<SyntaxNode>(),
```

Лістинг Д.3 – Код методу для порівняння AST-дерев коду студентських робіт

```
            "fragment" =>
altRoot.DescendantNodes().OfType<StatementSyntax>().Cast<SyntaxN
ode>(),
            _ => Enumerable.Empty<SyntaxNode>()
        };

        foreach (var node in candidateNodes)
        {
            if (node is MethodDeclarationSyntax methodNode)
            {
                string methodName = methodNode.Identifier.Text;
                List<string> parameters =
methodNode.ParameterList.Parameters.Select(p =>
p.Identifier.Text).ToList();

                var normalizer = new
IdentifierNormalizer(methodName, parameters);
                var normalizedTaskCode =
normalizer.Visit(node).NormalizeWhitespace().ToFullString();

                if (normalizedTaskCode.Equals(normalizedCode))
                {
                    SyntaxNode origMatch =
FindMatchingNodeInOriginalTree(altRoot, node, normalizer,
normalizedCode);

                    var lineSpan =
altTree.GetLineSpan(origMatch.Span);
                    int startLine =
lineSpan.StartLinePosition.Line + 1;
                    int endLine = lineSpan.EndLinePosition.Line
+ 1;

                    task.MatchedCode = node.ToFullString();
                    task.StartLine = startLine;
                    task.EndLine = endLine;

                    matchingTasks.Add(task);
                }
            }
            else if (node is LocalFunctionStatementSyntax
localFunc)
            {
                string funcName = localFunc.Identifier.Text;
                List<string> parameters =
localFunc.ParameterList.Parameters.Select(p =>
p.Identifier.Text).ToList();

                var normalizer = new
IdentifierNormalizer(funcName, parameters);
```

Лістинг Д.3, аркуш 2

```
                var normalizedTaskCode =
normalizer.Visit(node).NormalizeWhitespace().ToFullString();


                if (normalizedTaskCode.Equals(normalizedCode))
                {
                    SyntaxNode origMatch =
FindMatchingNodeInOriginalTree(altRoot, node, normalizer,
normalizedCode);

                    var lineSpan =
altTree.GetLineSpan(origMatch.Span);
                    int startLine =
lineSpan.StartLinePosition.Line + 1;
                    int endLine = lineSpan.EndLinePosition.Line
+ 1;

                    task.MatchedCode = node.ToFullString();
                    task.StartLine = startLine;
                    task.EndLine = endLine;

                    matchingTasks.Add(task);
                }

            }
            else if (node is ClassDeclarationSyntax classNode)
            {
                var normalizer = new IdentifierNormalizer();
                var normalizedTaskCode =
normalizer.Visit(node).NormalizeWhitespace().ToFullString();


                if (normalizedTaskCode.Equals(normalizedCode))
                {
                    SyntaxNode origMatch =
FindMatchingNodeInOriginalTree(altRoot, node, normalizer,
normalizedCode);

                    var lineSpan =
altTree.GetLineSpan(origMatch.Span);
                    int startLine =
lineSpan.StartLinePosition.Line + 1;
                    int endLine = lineSpan.EndLinePosition.Line
+ 1;

                    task.MatchedCode = node.ToFullString();
                    task.StartLine = startLine;
                    task.EndLine = endLine;

                    matchingTasks.Add(task);
                }
```

Лістинг Д.3, аркуш 3

```
            }
            else if (node is StructDeclarationSyntax structNode)
            {
                var normalizer = new IdentifierNormalizer();

                var normalizedTaskCode =
normalizer.Visit(node).NormalizeWhitespace().ToFullString();


                if (normalizedTaskCode.Equals(normalizedCode))
                {
                    SyntaxNode origMatch =
FindMatchingNodeInOriginalTree(altRoot, node, normalizer,
normalizedCode);

                    var lineSpan =
altTree.GetLineSpan(origMatch.Span);

                    int startLine =
lineSpan.StartLinePosition.Line + 1;
                    int endLine = lineSpan.EndLinePosition.Line
+ 1;

                    task.MatchedCode = node.ToFullString();
                    task.StartLine = startLine;
                    task.EndLine = endLine;

                    matchingTasks.Add(task);
                }

            }
            else
            {
                var normalizer = new IdentifierNormalizer();
                var normalizedTaskCode =
normalizer.Visit(node).NormalizeWhitespace().ToFullString();


                if (NormalizeWhitespace(normalizedTaskCode) ==
NormalizeWhitespace(normalizedCode))
                {
                    SyntaxNode origMatch =
FindMatchingNodeInOriginalTree(altRoot, node, normalizer,
normalizedCode);

                    var lineSpan =
altTree.GetLineSpan(origMatch.Span);
                    int startLine =
lineSpan.StartLinePosition.Line + 1;
                    int endLine = lineSpan.EndLinePosition.Line
+ 1;
```

Лістинг Д.3, аркуш 4

```
                    task.MatchedCode = node.ToFullString();
                    task.StartLine = startLine;
                    task.EndLine = endLine;

                    matchingTasks.Add(task);
                }

            }
        }
    }
    return matchingTasks;
}
```

Лістинг Д.3, аркуш 5

```
public NormalizedCodeResult GetNormalizedMethod(string code)
{

    SyntaxTree tree = CSharpSyntaxTree.ParseText(code);
    SyntaxNode root = tree.GetRoot();


    SyntaxNode targetNode =
root.DescendantNodes().OfType<ClassDeclarationSyntax>().Cast<Syn
taxNode>()

.Concat(root.DescendantNodes().OfType<StructDeclarationSyntax>()
.Cast<SyntaxNode>())

.Concat(root.DescendantNodes().OfType<MethodDeclarationSyntax>()
.Cast<SyntaxNode>())

.Concat(root.DescendantNodes().OfType<LocalFunctionStatementSynt
ax>().Cast<SyntaxNode>()).FirstOrDefault();

    string codeType = "";
    string wrappedCode = "";
    if (targetNode == null)
    {
        wrappedCode = CodeWrapper(code);
    }
    string methodName = "";

    List<string> parameters = new List<string>();
    SyntaxNode normalizedCode;
    IdentifierNormalizer normalizer;
```

Лістинг Д.4 – Код нормалізації фрагменту коду(обраного для аналізу) схожості

```
switch (targetNode)
    {
        case ClassDeclarationSyntax classNode:

            codeType = "class";
            normalizer = new IdentifierNormalizer();
            normalizedCode =
normalizer.Visit(targetNode).NormalizeWhitespace();
            break;
        case StructDeclarationSyntax structNode:

            codeType = "struct";
            normalizer = new IdentifierNormalizer();
            normalizedCode =
normalizer.Visit(targetNode).NormalizeWhitespace();
            break;
        case MethodDeclarationSyntax methodNode:

            codeType = "method";
            methodName = methodNode.Identifier.Text;
            parameters = methodNode.ParameterList.Parameters
                    .Select(p => p.Identifier.Text).ToList();
            normalizer = new IdentifierNormalizer(methodName,
parameters);
            normalizedCode =
normalizer.Visit(targetNode).NormalizeWhitespace();

            break;
        case LocalFunctionStatementSyntax functionNode:

            codeType = "localFunction";
            methodName = functionNode.Identifier.Text;
            parameters = functionNode.ParameterList.Parameters
                    .Select(p => p.Identifier.Text).ToList();
            normalizer = new IdentifierNormalizer(methodName,
parameters);
            normalizedCode =
normalizer.Visit(targetNode).NormalizeWhitespace();

            break;
        default:

            codeType = "fragment";
            SyntaxTree wrappedTree =
CSharpSyntaxTree.ParseText(wrappedCode);
            SyntaxNode wrappedRoot = wrappedTree.GetRoot();

            targetNode =
wrappedRoot.DescendantNodes().OfType<MethodDeclarationSyntax>().
FirstOrDefault();
            normalizer = new IdentifierNormalizer();
```

Лістинг Д.4, аркуш 2

```
            SyntaxNode normalizedWrappedCode =
normalizer.Visit(targetNode).NormalizeWhitespace();


            SyntaxNode extractedCode = normalizedWrappedCode
                .DescendantNodes()
                .OfType<StatementSyntax>()
                .FirstOrDefault();


            if (extractedCode is BlockSyntax block)
            {
                var statements = block.Statements;
                extractedCode =
block.Statements.FirstOrDefault();
            }


            if (extractedCode == null)
            {
                extractedCode = normalizedWrappedCode;
            }

            normalizedCode = extractedCode;

            break;
    }

    return new
NormalizedCodeResult(normalizedCode.ToFullString(), codeType);
}
```

Лістинг Д.4, аркуш 3

```
public override SyntaxNode
VisitMethodDeclaration(MethodDeclarationSyntax node)
{

    if (node.Identifier.Text == _methodName)
    {
        var newIdentifier =
SyntaxFactory.Identifier(_methodNormalizedName)
            .WithLeadingTrivia(node.Identifier.LeadingTrivia)
            .WithTrailingTrivia(node.Identifier.TrailingTrivia);
        node = node.WithIdentifier(newIdentifier);
    }
    else
    {
```

Лістинг Д.5 – Код нормалізації методу

```
        var newIdentifier =
SyntaxFactory.Identifier(_invokeNormalizedName)
            .WithLeadingTrivia(node.Identifier.LeadingTrivia)
            .WithTrailingTrivia(node.Identifier.TrailingTrivia);
        node = node.WithIdentifier(newIdentifier);
    }
    return base.VisitMethodDeclaration(node);
                                }
```

Лістинг Д.5, аркуш 2

```
public override SyntaxNode
VisitLocalFunctionStatement(LocalFunctionStatementSyntax node)
{
    if (node.Identifier.Text == _methodName)
    {
        var newIdentifier =
SyntaxFactory.Identifier(_methodNormalizedName)
            .WithLeadingTrivia(node.Identifier.LeadingTrivia)
            .WithTrailingTrivia(node.Identifier.TrailingTrivia);
        node = node.WithIdentifier(newIdentifier);
    }
    else
    {
        var newIdentifier =
SyntaxFactory.Identifier(_invokeNormalizedName)
            .WithLeadingTrivia(node.Identifier.LeadingTrivia)
            .WithTrailingTrivia(node.Identifier.TrailingTrivia);
        node = node.WithIdentifier(newIdentifier);
    }
    return base.VisitLocalFunctionStatement(node);
}
```

Лістинг Д.6 – Код нормалізації функції

```
public override SyntaxNode VisitParameter(ParameterSyntax node)
{if(_parametersMapping.TryGetValue(node.Identifier.Text, out
string normalized))
    {
        var newIdentifier = SyntaxFactory.Identifier(normalized)
            .WithLeadingTrivia(node.Identifier.LeadingTrivia)
            .WithTrailingTrivia(node.Identifier.TrailingTrivia);
        return node.WithIdentifier(newIdentifier);
    }
    return base.VisitParameter(node);
}
```

Лістинг Д.7 – Код нормалізації параметрів методу

```
public override SyntaxNode
VisitIdentifierName(IdentifierNameSyntax node)
{

    if (node.Parent is InvocationExpressionSyntax)
    {

        if (node.Identifier.Text == _methodName)
        {
            var newIdentifier =
SyntaxFactory.Identifier(_methodNormalizedName)

.WithLeadingTrivia(node.Identifier.LeadingTrivia)

.WithTrailingTrivia(node.Identifier.TrailingTrivia);
            return node.WithIdentifier(newIdentifier);
        }
        else
        {

            var newIdentifier =
SyntaxFactory.Identifier("invokedStatement")

.WithLeadingTrivia(node.Identifier.LeadingTrivia)

.WithTrailingTrivia(node.Identifier.TrailingTrivia);
            return node.WithIdentifier(newIdentifier);
        }
    }
    else
    {

        if
(_localVariablesMapping.TryGetValue(node.Identifier.Text, out
var normalizedLocal))
        {

            var newIdentifier =
SyntaxFactory.Identifier(normalizedLocal)

.WithLeadingTrivia(node.Identifier.LeadingTrivia)

.WithTrailingTrivia(node.Identifier.TrailingTrivia);
            return node.WithIdentifier(newIdentifier);
        }
        else if
(_parametersMapping.TryGetValue(node.Identifier.Text, out string
normalizedParam))
        {
```

Лістинг Д.8 – Код нормалізації ідентифікаторів

```
            var newIdentifier =
SyntaxFactory.Identifier(normalizedParam)

.WithLeadingTrivia(node.Identifier.LeadingTrivia)

.WithTrailingTrivia(node.Identifier.TrailingTrivia);
            return node.WithIdentifier(newIdentifier);
        }
        else
        {

            var newIdentifier =
SyntaxFactory.Identifier(_commonVariableName)

.WithLeadingTrivia(node.Identifier.LeadingTrivia)

.WithTrailingTrivia(node.Identifier.TrailingTrivia);
            return node.WithIdentifier(newIdentifier);
        }
    }
}
```

Лістинг Д.8, аркуш 2

```
public override SyntaxNode
VisitVariableDeclarator(VariableDeclaratorSyntax node)
{
    string originalName = node.Identifier.Text;
    _localVariablesMapping[originalName] = _commonVariableName;

    var newIdentifier =
SyntaxFactory.Identifier(_commonVariableName)
        .WithLeadingTrivia(node.Identifier.LeadingTrivia)
        .WithTrailingTrivia(node.Identifier.TrailingTrivia);
    return
base.VisitVariableDeclarator(node.WithIdentifier(newIdentifier))
;
}
```

Лістинг Д.9 – Код нормалізації локальних змінних

```
private SyntaxNode FindMatchingNodeInOriginalTree(SyntaxNode
origRoot, SyntaxNode nodeFromFormattedTree, IdentifierNormalizer
normalizer, string normalizedCode)
{
```

Лістинг Д.10 – Код виявлення позиції початку та кінця знайденого коду при виявлені схожості для подальшого виділення на сторінці

```
    IEnumerable<SyntaxNode> candidates = origRoot
        .DescendantNodesAndSelf()
        .Where(n => n.GetType() ==
nodeFromFormattedTree.GetType());

    foreach (var candidate in candidates)
    {
        string normalizedCandidate = normalizer
            .Visit(candidate)
            .NormalizeWhitespace()
            .ToFullString();

        if (normalizedCandidate == normalizedCode)
        {
            return candidate;
        }
    }
    return nodeFromFormattedTree;
}
```

Лістинг Д.10, аркуш 2

ДОДАТОК Е

Код реалізації модулю тестування

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> CreateTest(TestVM model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    model.Createdat = DateTime.Now;
    model.Teacherid = Int32.Parse(roleId);
    TestDTO testDTO = mapper.Map<TestDTO>(model);

    await testService.CreateTestAsync(testDTO, connectionId);

    return RedirectToAction("TeacherMainPage");
}
```

Лістинг Е.1 – Код створення нового тесту

```
[HttpGet]
public IActionResult CourseTests(int courseId)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    var testsDTO = testService.CourseTests(courseId,
connectionId);

    var testsVM = mapper.Map<List<TestVM>>(testsDTO);
    ViewBag.CourseId = courseId;
    return View(testsVM);
}
```

Лістинг Е.2 – Код отримання створених тестів

```
[HttpGet]
public IActionResult TestInfo(int testId)
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    var testDTO = testService.GetTest(testId, connectionId);

    var testVM = mapper.Map<TestVM>(testDTO);

    TempData["TestJson"] = JsonConvert.SerializeObject(testVM);

    return View(testVM);
}
```

Лістинг Е.3 – Код перегляду інформації про тест

```
[HttpGet]
public IActionResult TakeTest()
{
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;
    if (TempData.TryGetValue("TestJson", out var testJsonObj))
    {
        var testVM =
JsonConvert.DeserializeObject<TestVM>(testJsonObj.ToString());
        StudenttestVM studentTestVM = new StudenttestVM()
        {
            Test = testVM,
            Studentid = Int32.Parse(roleId),
            Testid = testVM.Testid,
            Startedat = DateTime.Now,
        };

        return View(studentTestVM);

    }
    string errorMessage = "Помилка. Не вдалося завантажити
тест";
    return View("ErrorMessage", errorMessage);
}
```

Лістинг Е.4 – Код запуску тесту

```
[HttpPost("SubmitTest")]
[ValidateAntiForgeryToken]
public async  Task<IActionResult> SubmitTest(StudenttestVM
model)
{
    if (!ModelState.IsValid)
    {
        return View("TakeTest", model);
    }
    model.Completeat = DateTime.Now;
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;
    StudenttestDTO studentTestDTO =
mapper.Map<StudenttestDTO>(model);

    var sessionKey =
$"OrientationCounts_{connectionId}_{model.Studenttestid}";
    var counts =
HttpContext.Session.GetObjectFromJson<Dictionary<string,
int>>(sessionKey)
                ?? new Dictionary<string, int> {
              { "RIGHT",0}, { "LEFT",0}, { "CENTER",0},
              { "UP",0}, { "DOWN",0}, { "NO_FACE",0}
                  };


    int totalDetections = counts["RIGHT"] + counts["LEFT"] +
counts["CENTER"] +
                          counts["UP"] + counts["DOWN"] +
counts["NO_FACE"];
    int honestCount = counts["CENTER"];
    int quality = Math.Min(100, totalDetections == 0 ? 0 :
(honestCount * 100 / totalDetections));

    studentTestDTO.Quality = quality;

    int newStudentTestId = await
testService.SubmitStudentTestAsync(studentTestDTO,
connectionId);



    return RedirectToAction("TestResults", new {
studentTestIdString = newStudentTestId, testIdString =
model.Testid });
}
```

Лістинг Е.5 – Код обробки відповідей до тесту

```
public async Task<int> SubmitStudentTestAsync(Studenttest
studentTest, string connectionId)
{
    UserDbContextInfo userDbContextInfo =
connectionService.GetInfo(connectionId.ToString());

    string connectionString =
userDbContextInfo.ConnectionString;

    var Dbbuilder = new
DbContextOptionsBuilder<ApplicationDbContext>();
    Dbbuilder.UseNpgsql(connectionString);
    ApplicationDbContext dbContext = new
ApplicationDbContext(Dbbuilder.Options, connectionString);

    using (var transaction = await
dbContext.Database.BeginTransactionAsync())
    {
        try
        {
            dbContext.Studenttests.AddAsync(studentTest);

            await dbContext.SaveChangesAsync();

            await transaction.CommitAsync();

            return studentTest.Studenttestid;
        }
        catch (Exception ex)
        {
            await transaction.RollbackAsync();
            Console.WriteLine(ex);
            throw;
        }
    }
}
```

Лістинг Е.6 – Код збереження результатів тесту в базу даних

```
public void SetScoreToStudentTest(int studentTestId, int
newScore, string connectionId)
{
    UserDbContextInfo userDbContextInfo =
connectionService.GetInfo(connectionId.ToString());
    string connectionString =
userDbContextInfo.ConnectionString;
```

Лістинг Е.7 – Код оновлення оцінки тесту

```
    var Dbbuilder = new
DbContextOptionsBuilder<ApplicationDbContext>();
    Dbbuilder.UseNpgsql(connectionString);
    ApplicationDbContext dbContext = new
ApplicationDbContext(Dbbuilder.Options, connectionString);

    var studentTest =
dbContext.Studenttests.Find(studentTestId);
    studentTest.Score = newScore;
    dbContext.SaveChanges();
}
```

Лістинг Е.7, аркуш 2

```
[HttpGet]
public IActionResult TestResults(string studentTestIdString,
string testIdString)
{
    if (!int.TryParse(studentTestIdString, out var
studentTestId) ||
        !int.TryParse(testIdString, out var testId))
    {
        return BadRequest("Неправильні параметри запиту.");
    }

    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId")?.Value;
    if (connectionId == null)
        return Unauthorized();


    var testDTO = testService.GetTest(testId, connectionId);
    var studentTestDTO =
testService.GetStudentTest(studentTestId, connectionId);

    int totalAutoQuestions = 0;
    int correctAnswers = 0;
    int calculatedScore = 0;

    if (studentTestDTO == null)
        return NotFound("Не вдалося знайти проходження тесту.");

    foreach (var answer in studentTestDTO.Studentanswers)
    {
        var question = testDTO.Testquestions
            .FirstOrDefault(q => q.Questionid ==
answer.Questionid);
```

Лістинг Е.8 – Код оформлення результатів проходження тесту

```
        if (question == null)
            continue;

        if (question.Questiontype == "SingleChoice")
        {
            totalAutoQuestions++;

            var correctOption = question.Questionoptions
                .FirstOrDefault(o => o.Iscorrect);

            if (correctOption != null &&
                answer.Selectedoptionid ==
correctOption.Optionid)
            {
                correctAnswers++;
                answer.Score = question.Score;
                calculatedScore += question.Score;
            }
            else
            {
                answer.Score = 0;
            }
        }
        else
        {

            calculatedScore += answer.Score;
        }
    }

testService.UpdateStudentAnswerScores(studentTestDTO.Studentansw
ers, connectionId);

    testService.SetScoreToStudentTest(studentTestId,
calculatedScore, connectionId);


    var testResults = new TestResultsVM
    {
        Score = calculatedScore,
        CorrectAnswers = correctAnswers,
        Quality = studentTestDTO.Quality,
        TotalAutoQuestions = totalAutoQuestions,
        Message = "Питання з текстовою відповіддю або написанням
коду перевіряє викладач і додає до поточної оцінки."
    };
    return View(testResults);
}
```

Лістинг Е.8, аркуш 2

ДОДАТОК Ж

Код реалізації модулю прокторингу

```
enableCameraBtn.addEventListener("click", async () => {

    cameraPrompt.style.display = "none";


    cameraContainer.style.display = "block";
    await loadModels();


    try {
        const stream = await
navigator.mediaDevices.getUserMedia({ video: true });
        proctorVideo.srcObject = stream;
    } catch (err) {
        console.error("Не вдалося отримати доступ до камери:",
err);
        return;
    }


    proctorVideo.addEventListener('playing', () => {
        monitorFace();
    });


    testForm.style.display = "block";
});
```

Лістинг Ж.1 – Код підключення камери для проходження тесту

```
function estimateFaceOrientation(landmarks) {
    const leftEyePts = landmarks.getLeftEye();
    const rightEyePts = landmarks.getRightEye();
    const nosePts = landmarks.getNose();
    const jawPts = landmarks.getJawOutline();
    const browLeft = landmarks.getLeftEyeBrow()[4];
    const browRight = landmarks.getRightEyeBrow()[4];
    const B = leftEyePts[3];
    const C = rightEyePts[0];
    const E = nosePts[3];
    const G = jawPts[8];
    const H = { x: (browLeft.x + browRight.x) / 2,
```

Лістинг Ж.2 – Код аналізу положення обличчя

```
        y: (browLeft.y + browRight.y) / 2
    };
    const midEyeX = (B.x + C.x) / 2;
    const deltaX = E.x - midEyeX;
    let yawState;
    if (deltaX < -7) yawState = "RIGHT";
    else if (deltaX > 7) yawState = "LEFT";
    else yawState = "CENTER";
    const pitchRatio = (E.y - H.y) / (G.y - H.y);
    let pitchState;
    if (pitchRatio < 0.40) pitchState = "UP";
    else if (pitchRatio > 0.55) pitchState = "DOWN";
    else pitchState = "CENTER";
    return { yawState, pitchState };
}
```

Лістинг Ж.2, аркуш 2

```
[HttpPost]
public IActionResult RecordOrientation([FromBody]
OrientationData data)
{
    if (data == null)
    {
        return BadRequest();
    }
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId")?.Value;
    if (connectionId == null)
    {
        return Unauthorized();
    }
    var sessionKey =
$"OrientationCounts_{connectionId}_{data.Studenttestid}";
    var counts =
HttpContext.Session.GetObjectFromJson<Dictionary<string,
int>>(sessionKey)
                ?? new Dictionary<string, int> {
             { "RIGHT",0}, { "LEFT",0}, { "CENTER",0},
             { "UP",0}, { "DOWN",0}, { "NO_FACE",0}
                };
    if (counts.ContainsKey(data.Yaw)) counts[data.Yaw]++;

    if (counts.ContainsKey(data.Pitch)) counts[data.Pitch]++;
    HttpContext.Session.SetObjectAsJson(sessionKey, counts);
    return Ok();
}
```

Лістинг Ж.3 – Код отримання та накопичення даних про положення обличчя

```
[HttpPost("SubmitTest")]
[ValidateAntiForgeryToken]
public async  Task<IActionResult> SubmitTest(StudenttestVM
model)
{
    if (!ModelState.IsValid)
    {
        return View("TakeTest", model);
    }

    model.Completeat = DateTime.Now;
    var user = HttpContext.User;
    var connectionId = user.Claims.FirstOrDefault(c => c.Type ==
"AccountId").Value;
    var roleId = user.Claims.FirstOrDefault(c => c.Type ==
"RoleId")?.Value;

    StudenttestDTO studentTestDTO =
mapper.Map<StudenttestDTO>(model);

    var sessionKey =
$"OrientationCounts_{connectionId}_{model.Studenttestid}";
    var counts =
HttpContext.Session.GetObjectFromJson<Dictionary<string,
int>>(sessionKey)
                ?? new Dictionary<string, int> {
              { "RIGHT",0}, { "LEFT",0}, { "CENTER",0},
              { "UP",0}, { "DOWN",0}, { "NO_FACE",0}
                  };


    int totalDetections = counts["RIGHT"] + counts["LEFT"] +
counts["CENTER"] +
                          counts["UP"] + counts["DOWN"] +
counts["NO_FACE"];
    int honestCount = counts["CENTER"];
    int quality = Math.Min(100, totalDetections == 0 ? 0 :
(honestCount * 100 / totalDetections));

    studentTestDTO.Quality = quality;

    int newStudentTestId = await
testService.SubmitStudentTestAsync(studentTestDTO,
connectionId);
    return RedirectToAction("TestResults", new {
studentTestIdString = newStudentTestId, testIdString =
model.Testid });
}
```

Лістинг Ж.4 – Код обробки результатів тестування та обчислення рівня доброчесності

ДОДАТОК К

Запити на створення доменів та таблиць бази даних

```
create table Teacher
(
     TeacherId serial primary key,
     FirstName varchar(30) not null,
     LastName varchar(30) not null,
     AccountId integer not null,
     foreign key (AccountId) references users.Account(AccountId)
on delete restrict on update cascade
);
```

Лістинг К.1 – Створення таблиці «Викладач»

```
create table Course
(
     CourseId serial primary key,
     CourseName varchar(100) not null,
     CourseDescription text not null,
     StartDate date not null,
     EndDate date not null,
     check(EndDate >= StartDate),
     TeacherId integer not null,
     foreign key (TeacherId) references Teacher(TeacherId) on
delete restrict on update cascade
);
```

Лістинг К.2 – Створення таблиці «Курс»

```
create table CourseTask
(

     TaskId serial primary key,
     TaskName varchar(50) not null,
     TaskDescription text not null,
     StartDate date not null,
     Deadline date not null,
     check(Deadline >= StartDate),
     CourseId integer not null,
     foreign key (CourseId) references Course(CourseId) on delete
restrict on update cascade
);
```

Лістинг К.3 – Створення таблиці «Завдання курсу»

```
create table Student
(
     StudentId serial primary key,
     FirstName varchar(50) not null,
     LastName varchar(50) not null,
     RegistrationDate date not null,
     AccountId integer not null,
     foreign key (AccountId) references users.Account(AccountId)
on delete restrict on update cascade
);
```

Лістинг К.4 – Створення таблиці «Студент»

```
create domain WorkStatus as varchar(30)
default 'Submitted'
check (value in ('Submitted', 'Evaluated'));
```

Лістинг К.5 – Створення домену «Статус роботи»

```
create table StudentWork
(
     WorkId serial primary key,
     SubmissionDate date not null,
     Status WorkStatus not null,
     Grade integer check (Grade between 1 and 10 or Grade is
null),
     Code text not null,
     StudentId integer not null,
     foreign key (StudentId) references Student(StudentId) on
delete restrict on update cascade,
     TaskId integer not null,
     foreign key (TaskId) references CourseTask(TaskId) on delete
restrict on update cascade,
     unique (StudentId, TaskId)
);
```

Лістинг К.6 – Створення таблиці «Робота студента»

```
create table Student_In_Course
(
     primary key (StudentId, CourseId),
     EnrollmentDate date not null,
     StudentId integer not null,
     CourseId integer not null,
```

Лістинг К.7 – Створення таблиці «Студент в курсі»

```
     foreign key (StudentId) references Student(StudentId) on
delete restrict on update cascade,
     foreign key (CourseId) references Course(CourseId) on delete
restrict on update cascade,
     unique(StudentId, CourseId)
);
```

Лістинг К.7, аркуш 2

```
create table users.Account
(
     AccountId serial primary key,
     Login text unique not null,
     PasswordHash text not null,
     AccountRole varchar(20) not null check (AccountRole in
('teacher', 'student'))
);
```

Лістинг К.8 – Створення таблиці «Обліковий запис»

```
create domain MessageType as varchar(100)
default 'Default message'
check (value in('Default message', 'Work submission', 'Work
update', 'Work evaluated', 'Course invitation'));
```

Лістинг К.9 – Створення домену «Тип повідомлення»

```
create table Notification
(

     NotificationId serial primary key,
     Title varchar(200) not null,
     MessageText text,
     MessageType MessageType not null,
     CreatedAt timestamp not null default now(),
     IsRead boolean not null default false,
     SenderId integer not null,
     foreign key (SenderId) references users.Account(AccountId)
on delete cascade on update cascade,
     ReceiverId integer not null,
     foreign key (ReceiverId) references users.Account(AccountId)
on delete cascade on update cascade
);
```

Лістинг К.10 – Створення таблиці «Повідомлення»

```
create domain QuestionType as varchar(50)
check (value in ('SingleChoice', 'Text', 'Code'));
```

Лістинг К.11 – Створення домену «Тип запитання»

```
create table Test
(
  TestId       serial primary key,
  Title        varchar(200) not null,
  Description  text,
  TeacherId    integer not null,
  foreign key (TeacherId) references Teacher(TeacherId) on
delete cascade on update cascade,
  CreatedAt    timestamp not null default now(),
  Duration integer not null check (Duration > 0),
  CourseId integer not null,
  foreign key (CourseId) references Course(CourseId) on delete
cascade on update cascade
);
```

Лістинг К.12 – Створення таблиці «Тест»

```
create table TestQuestion
(
  QuestionId   serial primary key,
  TestId       integer not null,
  foreign key (TestId) references Test(TestId) on delete cascade
on update cascade,
  QuestionText text not null,
  QuestionType QuestionType not null,
  Score integer not null default 1
);
```

Лістинг К.13 – Створення таблиці «Запитання тесту»

```
create table QuestionOption
(
  OptionId serial primary key,
  QuestionId integer not null,
  foreign key (QuestionId) references TestQuestion(QuestionId)
on delete cascade on update cascade,
  OptionText text not null,
  IsCorrect boolean not null default false
);
```

Лістинг К.14 – Створення таблиці «Варіант відповіді»

```
create table StudentTest
(
  StudentTestId serial primary key,
  StudentId     integer not null,
  foreign key (StudentId) references Student(StudentId) on
delete cascade on update cascade,
  TestId integer not null,
  foreign key (TestId) references Test(TestId) on delete cascade
on update cascade,
  StartedAt     timestamp not null default now(),
  CompleteAt timestamp not null,
  Quality integer not null default 0,
  Score integer not null
);
```

Лістинг К.15 – Створення таблиці «Проходження студентом тесту»

```
create table StudentAnswer
(
  StudentAnswerId serial primary key,
  StudentTestId integer not null,
  foreign key (StudentTestId) references
StudentTest(StudentTestId) on delete cascade on update cascade,
  QuestionId integer not null,
  foreign key (QuestionId) references TestQuestion(QuestionId)
on delete cascade on update cascade,
  SelectedOptionId integer,
  foreign key (SelectedOptionId) references
QuestionOption(OptionId) on delete set null,
  AnswerText text,
  Score integer not null
);
```

Лістинг К.16 – Створення таблиці «Відповідь студента на питання»

ДОДАТОК Л

Запити на створення функцій та процедур

```
create or replace procedure register_teacher(teacher_first_name
text, teacher_last_name text, hashed_password text,
teacher_login text)
as $$
declare
     new_account_id integer;
begin

     perform set_config('search_path', 'users, public', false);

     insert into users.Account(Login, PasswordHash, AccountRole)
     values
     (teacher_login, hashed_password, 'teacher')
     returning AccountId into new_account_id;

     insert into public.Teacher(FirstName, LastName, AccountId)
     values
     (teacher_first_name, teacher_last_name, new_account_id);

end;
$$ language plpgsql
security definer;
```

Лістинг Л.1 – Процедура для внесення даних нового викладача

```
create or replace procedure register_student(student_first_name
text, student_last_name text, hashed_password text,
student_login text, registration_date date)
as $$
declare
     new_account_id integer;
begin
     perform set_config('search_path', 'users, public', false);

     insert into users.Account(Login, PasswordHash, AccountRole)
     values
     (student_login, hashed_password, 'student')
     returning AccountId into new_account_id;
     insert into public.Student(FirstName, LastName,
RegistrationDate, AccountId)
     values
     (student_first_name, student_last_name, registration_date,
new_account_id);

end;
```

```
$$ language plpgsql
security definer;
```

Лістинг Л.2 – Процедура для внесення даних нового студента

```
create or replace function GetReceivedMessages(p_accountId int)
returns table
(
     NotificationId int,
     Title text,
     MessageText text,
     MessageType MessageType,
     CreatedAt timestamp,
     IsRead boolean,
     SenderLogin text,
     SenderFirstName text,
     SenderLastName text,
     ReceiverLogin text,
     ReceiverFirstName text,
     ReceiverLastName text
)
language sql
security definer
as
$$
     select n.NotificationId, n.Title, n.MessageText,
n.MessageType, n.CreatedAt, n.IsRead, sender_acc.Login as
SenderLogin, coalesce(sender_teacher.FirstName,
sender_student.FirstName), coalesce(sender_teacher.LastName,
sender_student.LastName), receiver_acc.Login,
coalesce(receiver_teacher.FirstName,
receiver_student.FirstName), coalesce(receiver_teacher.LastName,
receiver_student.LastName)
     from Notification n join users.Account sender_acc on
n.SenderId = sender_acc.AccountId
     left join Teacher sender_teacher on sender_acc.AccountId =
sender_teacher.AccountId
     left join Student sender_student on sender_acc.AccountId =
sender_student.AccountId
     join users.Account receiver_acc on receiver_acc.AccountId =
n.ReceiverId
     left join Teacher receiver_teacher on receiver_acc.AccountId
= receiver_teacher.AccountId
     left join Student receiver_student on receiver_acc.AccountId
= receiver_student.AccountId
     where n.ReceiverId = p_accountId
     order by n.CreatedAt desc;
$$;
```

Лістинг Л.3 – Функція для отримання вхідних повідомлень користувача

```
create or replace function GetSentMessages(p_accountId int)
returns table
(
     NotificationId int,
     Title text,
     MessageText text,
     MessageType MessageType,
     CreatedAt timestamp,
     IsRead boolean,
     ReceiverLogin text,
     ReceiverFirstName text,
     ReceiverLastName text,
     SenderLogin text,
     SenderFirstName text,
     SenderLastName text
)
language sql
security definer
as
$$
     select n.NotificationId, n.Title, n.MessageText,
n.MessageType, n.CreatedAt, n.IsRead, receiver_acc.Login as
SenderLogin, coalesce(receiver_teacher.FirstName,
receiver_student.FirstName), coalesce(receiver_teacher.LastName,
receiver_student.LastName), sender_acc.Login as SenderLogin,
coalesce(sender_teacher.FirstName, sender_student.FirstName),
coalesce(sender_teacher.LastName, sender_student.LastName)
     from Notification n join users.Account receiver_acc on
n.ReceiverId = receiver_acc.AccountId
     left join Teacher receiver_teacher on receiver_acc.AccountId
= receiver_teacher.AccountId
     left join Student receiver_student on receiver_acc.AccountId
= receiver_student.AccountId
     join users.Account sender_acc on sender_acc.AccountId =
n.SenderId
     left join Teacher sender_teacher on sender_acc.AccountId =
sender_teacher.AccountId
     left join Student sender_student on sender_acc.AccountId =
sender_student.AccountId

     where n.SenderId = p_accountId
     order by n.CreatedAt desc;
$$;
```

Лістинг Л.4 – Функція для отримання відправлених повідомлень користувача

ДОДАТОК М

Запити на створення ролей та надання їм привілеїв

```
create role complex_connect_user with login password 'c0nnect'

grant usage on schema users to complex_connect_user;
grant select on users.Account to complex_connect_user;
grant execute on procedure register_teacher to
complex_connect_user;
grant execute on procedure register_student to
complex_connect_user;
```

Лістинг М.1 – Створення службової ролі «complex_connect_user» з привілеями

```
create role Teacher with login password 'teacher';
grant insert, select, update, delete on Student_In_Course to
Teacher;
grant insert, select, update, delete on Notification to Teacher;
grant insert, select, update, delete on Course to Teacher;
grant insert, select, update, delete on CourseTask to Teacher;
grant select, update on public.Teacher to Teacher;
grant select, update on StudentWork to Teacher;
grant select on Student to Teacher;
grant insert, select, update, delete on Test to Teacher;
grant insert, select, update, delete on TestQuestion to Teacher;
grant insert, select, update, delete on QuestionOption to
Teacher;
grant select, update on StudentTest to Teacher;
grant select, update on StudentAnswer to Teacher;

grant usage, select on sequence course_courseid_seq,
coursetask_taskid_seq, notification_notificationid_seq to
Teacher; -- для послідовностей первинного ключа
grant usage, select on sequence test_testid_seq to Teacher;
grant usage, select on sequence testquestion_questionid_seq to
Teacher;
grant usage, select on sequence questionoption_optionid_seq to
Teacher;
```

Лістинг М.2 – Створення ролі «Teacher» (Викладач) з привілеями

```
create role Student with login password 'student';
grant insert, select, update, delete on StudentWork to Student;
grant insert, select, update, delete on Notification to Student;
```

```
grant select, update on Student to Student;
grant select on Teacher, Course, CourseTask, Student_In_Course
to Student;
grant select on Test to Student;
grant select on TestQuestion to Student;
grant select on QuestionOption to Student;
grant select, insert, update on StudentTest to Student;
grant select, insert, update on StudentAnswer to Student;

grant usage, select on sequence studentwork_workid_seq,
notification_notificationid_seq to Student; -- для
послідовностей первинного ключа
grant usage, select on sequence studenttest_studenttestid_seq to
Student;
grant usage, select on sequence
studentanswer_studentanswerid_seq to Student;
```

Лістинг М.3 – Створення ролі «Student» (Студент) з привілеями