МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: **«Розробка системи автоматизованого оцінювання програмного коду на основі юніт тестів»**

Виконала: студентка 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Чирко Валерія Ігорівна

Керівник: *старший викладач кафедри інформаційних технологій та аналітики даних*
Клебан Юрій Вікторович

Рецензент: *ФОП в галузі інформаційних технологій, викладач кафедри менеджменту та маркетингу НаУОА*
Шпак Андрій Григорович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____
проф., д.е.н. Кривицька О.Р.
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Розробка системи автоматизованого оцінювання програмного коду на основі юніт тестів.

Автор: Чирко Валерія Ігорівна.

Науковий керівник: старший викладач кафедри інформаційних технологій та аналітики даних Клебан Юрій Вікторович.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *70 с., 14 рис., 3 табл., 4 додатки, 40 джерел.*

Ключові слова: автоматизоване оцінювання, модульні тести, мікросервісна архітектура, Next.js, .NET, Docker, Redis, PostgreSQL, Keycloak, хмарні сервіси.

Короткий зміст праці:
У кваліфікаційній роботі окреслено створення інформаційної системи, призначеної для автоматичної оцінки програмного коду студентів за допомогою модульних тестів. Ця система призначена для використання у вищих навчальних закладах, пропонуючи можливість перевірки поданого студентами коду в узгодженому, ізольованому середовищі.
З погляду архітектури програмного забезпечення, модель детально описана з використанням мікросервісного підходу, що включає сучасні вебтехнології. Клієнтська функціональність розроблена з використанням фреймворку Next.js, що забезпечує інтерактивний інтерфейс та авторизацію на основі Keycloak. Серверний компонент побудовано на платформі .NET 9 з gRPC, що забезпечує міжсервісну комунікацію. Docker-контейнери використовуються для компіляції коду та виконання тестів, що забезпечує безпечне та контрольоване операційне середовище.
Система використовує Redis для кешування даних та PostgreSQL для зберігання результатів оцінювання, журналів спроб і метаданих. Доступні зручні інтерфейси для керування курсами, завданнями та тестовими рішеннями. Було створено гібридну хмарну інфраструктуру: фронтенд розміщено на Vercel, бекенд — на Azure, а бази даних та допоміжні сервіси — на Railway. Запропонована конфігурація призводить до створення надійної та розширюваної системи, яка спрощує процес перевірки коду, полегшує навантаження на викладачів та підвищує неупередженість оцінювання.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Title: Development of an Automated Code Assessment System Based on Unit Tests

Author: Valeriia Ihorivna Chyrko

Scientific advisor: Yurii Viktorovych Kleban.

Defended «……»…… 2025.

Explanatory note to the qualification thesis: 70 pages, 14 figures, 3 tables, 4 appendices, 40 references.

Keywords: automated assessment, unit testing, microservice architecture, Next.js, .NET, Docker, Redis, PostgreSQL, Keycloak, cloud services.

Abstract:
This bachelor's thesis presents the development of an automated system for assessing students' programming solutions based on unit tests. The system is designed for use in higher education institutions to provide efficient and objective evaluation of code submitted by students within a reproducible and secure environment.
The architecture of the software is based on a microservice approach using modern web technologies. The client-side is implemented with the Next.js framework and provides an interactive interface with authentication via Keycloak. The backend is built on the .NET 9 platform, using gRPC for service-to-service communication. Code compilation and test execution are handled within Docker containers, ensuring safety and isolation.
The system integrates Redis for caching and PostgreSQL for storing evaluation results, attempt logs, and metadata. Administrative interfaces allow managing courses, tasks, and evaluation structures. A hybrid cloud deployment strategy is implemented: the frontend is hosted on Vercel, the backend on Azure, and the database services on Railway.
As a result, the developed system is scalable and reliable, automating the code verification process, reducing instructors' workload, and improving the fairness and transparency of assessments.

_____

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ЦП	Центральний процесор — основний виконавчий компонент комп’ютерної системи, що виконує обчислення та керує операціями.
UI	User Interface — користувацький інтерфейс; сукупність засобів взаємодії користувача з програмним забезпеченням.
Docker	Платформа для створення та запуску ізольованих середовищ (контейнерів), що забезпечує стабільне виконання коду.
LMS	Learning Management System — система управління навчанням для адміністрування та доставки освітніх курсів.
Keycloak	Система керування ідентифікацією та доступом з підтримкою OpenID Connect, SAML і OAuth2.
JWT	JSON Web Token — формат передачі зашифрованих токенів для автентифікації та авторизації.
API	Application Programming Interface — інтерфейс прикладного програмування для взаємодії компонентів.
gRPC	gRPC Remote Procedure Call — фреймворк для виклику віддалених процедур з використанням HTTP/2 і Protocol Buffers.
REST	Representational State Transfer — архітектурний стиль вебсервісів, що працюють за HTTP.
CI/CD	Continuous Integration / Continuous Delivery — автоматизоване тестування та розгортання програмного забезпечення.

Мікросервісна архітектура	Організація системи як набір незалежних сервісів, що взаємодіють через визначені інтерфейси.
Redis	Структура в пам’яті для кешування, зберігання даних і передачі повідомлень.
PostgreSQL	Об'єктно-реляційна СУБД з відкритим кодом для зберігання структурованої інформації.
Vercel	Хмарна платформа для розгортання фронтенду, особливо з використанням Next.js.
Azure	Хмарна платформа Microsoft для хостингу, моніторингу та масштабування бекенд-сервісів.
Railway	Платформа для розгортання баз даних, Redis та інфраструктурних компонентів.
.NET	Кросплатформна платформа Microsoft для створення серверної логіки системи.
Next.js	Фреймворк для React-додатків із серверним рендерингом та статичною генерацією сторінок.
Unit-тестування	Перевірка коректності окремих функціональних модулів ПЗ.
Автоматизоване оцінювання коду	Автоматична перевірка коду за допомогою тестів без участі викладача.
Кешування	Тимчасове зберігання результатів запитів для підвищення швидкодії системи.

ВСТУП

Сучасна система освіти зазнає суттєвих змін через вплив цифрових технологій. Інтеграція автоматизованих інструментів в освіту не лише оптимізує методи навчання, але й покращує загальний досвід навчання. Особливо важливою є автоматизація оцінювання програмних завдань, яка скорочує час оцінювання, усуває суб'єктивні упередження та забезпечує негайний зворотний зв'язок зі студентами. У сфері навчання інформаційних технологій впровадження автоматизованого тестування коду є доречним та необхідним елементом освітньої програми.

Модульне тестування (юніт-тести) є важливим для автоматизованого тестування програмних рішень з метою об'єктивної оцінки знань студентів. Воно оцінює правильність логіки та відповідність реалізації очікуваним результатам. Розробка такої системи вимагає комплексного підходу, від проєктування архітектури до обробки виняткових ситуацій, таких як помилки компіляції, структурні невідповідності або порушення стандартів кодування. Отже, розробка ефективної автоматизованої системи оцінювання є нагальним завданням як на рівні окремих навчальних закладів, так і на ширших освітніх платформах.

Мета дослідження – створення програмного продукту, який забезпечить автоматизовану перевірку програмних рішень користувачів шляхом виконання юніт тестів і формування результатів оцінювання.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та дослідити сучасні методології автоматизованої перевірки коду.;
- розробити архітектуру інформаційної системи, яка відповідає критеріям модульності, масштабованості та безпеки;
- обрати відповідний інструментарій для реалізації та тестування програмного продукту;
- розробити прототип системи та перевірити її функціональність у контрольованих умовах.

Об’єктом дослідження є система автоматизованого оцінювання програмного коду в освітньому середовищі.

Предметом дослідження є інструментальні засоби, що застосовуються для розробки системи, зокрема мови програмування, фреймворки, моделі взаємодії компонентів, механізми кешування та юніт тестування.

У процесі роботи використовувалися такі методи дослідження: порівняльний аналіз сучасних технологій, методи об’єктно-орієнтованого проєктування, моделювання архітектури програмного забезпечення, прототипування та модульне тестування. Також використовувалися емпіричні методи перевірки працездатності системи в рамках тестових сценаріїв, максимально наближених до реального використання.

Актуальність теми дослідження зумовлена зростаючим попитом на цифрову трансформацію в освіті та необхідністю прозорих та неупереджених методів оцінювання знань з програмування. Результати цього дослідження мають практичну цінність і можуть бути застосовані у вищих навчальних закладах, онлайн-курсах та корпоративних навчальних програмах для оптимізації процесів оцінювання знань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ТЕСТУВАННЯ ПРОГРАМНОГО КОДУ В АВТОМАТИЗОВАНИХ СИСТЕМАХ

1.1. Особливості автоматизованого тестування в сучасному програмному забезпеченні

Автоматизація інструментів оцінювання коду стала поширеною в комп'ютерних курсах протягом останнього десятиліття. Ці інструменти значно зменшили необхідність ручного тестування, особливо у великих когортах студентів. Наприклад, у дослідженні Messer з колегами показано, що більшість таких систем надають повністю автоматизовану оцінку з миттєвим зворотним зв’язком, що підвищує задоволеність студентів і дає їм більше можливостей успішно виконати завдання [4]. Аналогічно Srinivasan і співавт. описують досвід використання платформ на кшталт CodeRunner або GitHub Classroom: у цьому випадку перевірка студентського коду відбувається автоматично без ручного втручання викладача, що звільняє його час для пояснення та налагодження складніших фрагментів програми [3]. Таким чином автоматизовані системи поступово долають обмеження традиційного підходу і замінюють ручне тестування, забезпечуючи об’єктивність і оперативність оцінювання.

Автоматизоване тестування відіграє вирішальну роль у розробці програмного забезпечення, підвищуючи ефективність, надійність та якість протягом усього процесу. Різні дослідження підкреслюють позитивний вплив автоматизованого тестування на результати студентів. Messer з колегами у дослідженні наголошують, що негайний зворотний зв'язок та необмежена кількість повторних спроб виконання завдань за допомогою автоматизованого тестування підвищують задоволеність студентів та мотивацію до виправлення помилок [4]. Панькевич та його колеги провели експериментальне дослідження, яке демонструє, що розбиття завдань на невеликі «мікрозавдання» за допомогою автоматизованого тестування зменшує розчарування та плутанину студентів, що призводить до швидшого виконання вправ. Група, яка використовувала мікротренінг,

продемонструвала значне покращення кінцевих результатів тестування порівняно з контрольною групою [8]. Ці результати підкреслюють цінність автоматизованого тестування у сприянні навчанню завдяки швидкому зворотному зв'язку та виправленню помилок у режимі реального часу.

Згідно з Міжнародною радою з кваліфікації тестування програмного забезпечення (ISTQB), тестування – це комплексний процес, який включає різні дії протягом життєвого циклу розробки програмного забезпечення. Це включає статичне та динамічне тестування, а також планування, підготовку та оцінку програмного забезпечення та пов'язаних з ним робочих продуктів. Мета полягає в тому, щоб підтвердити, що всі продукти відповідають заданим вимогам, придатні для використання та не мають дефектів [1].

Основні цілі тестування включають виявлення дефектів програмного забезпечення, перевірку відповідності системи специфікаціям та оцінку якості програмного забезпечення. ISTQB описує різні методи тестування, такі як «чорна скринька», «біла скринька» та інші, для забезпечення ефективного виконання тестів. Різні типи тестування, визначені ISTQB, включають модульне, інтеграційне, системне, регресійне та приймальне тестування. ISTQB робить акцент на документації в тестуванні, що охоплює плани тестування, тестові випадки, звіти про результати тестування та іншу відповідну документацію [1].

У цьому контексті можна логічно припустити, що забезпечення вищезгаданих цілей зазвичай передбачає багатоетапний процес Оцінювання студентських проєктів, кожен з яких відображає різні рівні складності. Спочатку проводиться перевірка для виявлення будь-яких синтаксичних помилок або помилок компіляції. Згодом оцінювання переходить до логічного рівня, де тестується функціональність програмного коду. Модульні тести є вирішальними на цьому етапі, оскільки вони пропонують об'єктивну оцінку продуктивності окремих функцій, методів і класів. Динамічне тестування, зокрема у формі модульних тестів, широко використовується автоматизованими інструментами оцінювання для перевірки виконання завдань. Як зазначають Messer з колегами, цей підхід включає тестування рішення з вхідними даними та порівняння результату з очікуваним результатом для

визначення правильності [4]. Остаточний висновок, зазвичай поданий у форматі "зараховано/не зараховано", дозволяє швидко та автоматизовано оцінювати рівні правильності рішення, тим самим утверджуючи модульні тести як традиційний підхід у сучасних системах автоматизованої оцінки програмного коду.

Окрім цього, оцінювання охоплює й інші важливі аспекти, такі як якість структури коду, покриття тестами, дотримання правил іменування, ефективність алгоритмів та реалізація шаблонів проєктування. Тому важливо, щоб система оцінювання могла враховувати всі ці фактори, проводячи статичний та динамічний аналіз програми. Динамічне оцінювання включає виконання програми для перевірки її функціональності та ефективності, порівнюючи вихідні дані програми з модельним рішенням. В інструментах, таких як Ceilidh, Assyst та Online Judge, результат виконання програми порівнюється з очікуваним, а інші вимірюють час ЦП або використовують динамічне профілювання для оцінки ефективності.

Цей підхід до оцінювання можна використовувати як для формувального, так і для підсумкового оцінювання. Формувальне оцінювання надає студентам своєчасний зворотний зв’язок для поліпшення їхніх знань, а підсумкове оцінює загальні досягнення студента. Проте важливо, щоб викладачі ретельно розробляли завдання та інтегрували автоматизоване оцінювання так, щоб уникнути недоліків, як от стимулювання поверхневих змін у коді або зменшення зусиль студентів щодо самостійного тестування. Збалансоване поєднання автоматизованого та ручного оцінювання може стати найбільш ефективним методом забезпечення вичерпного зворотного зв’язку і поліпшення якості навчання студентів.

1.2. Огляд існуючих рішень для оцінювання коду на основі юніт тестів

Останні тенденції в курсах програмування полягають у тому, що дедалі більше уваги приділяється відображенню реальних практик виробництва програмного забезпечення. Як результат, зростає впровадження систем контролю версій, таких як Git, та інструментів безперервної інтеграції та розгортання (CI/CD), які вже є галузевими стандартами. Такі платформи, як GitHub Classroom, активно

використовуються для розподілу навчальних завдань, тоді як Jenkins CI використовується для автоматизації процесів перевірки коду, включаючи модульне тестування та аналіз стилю, з кожним комітом [7]. Такий оптимізований підхід дозволяє викладачам зосередитися на наданні цінного зворотного зв'язку високого рівня щодо таких аспектів, як архітектура та дизайн коду, тоді як рутинні перевірки форматування та синтаксису виконуються автоматично. У роботі GitHub описано приклад успішного курсу, організованого John David N. Dionisio, де студенти отримали доступ до підготовлених репозиторіїв, що містять інструкції та тестові файли, при цьому система Jenkins виявляла стилістичні помилки без ручного втручання. Даний підхід надав викладачу можливість зосередитися на наданні якісного зворотного зв'язку щодо архітектури та дизайну коду, замість виправлення основних помилок форматування, таких як відступи або відсутні крапки з комою [7]. Українські ресурси також підкреслюють переваги підходів CI/CD, зазначаючи, що автоматизовані тести можна легко інтегрувати в процеси для раннього виявлення помилок та швидкого коригування [6]. Таке впровадження надає учням навичок роботи відповідно до галузевих стандартів.

Широке впровадження автоматизованих систем оцінки коду в глобальному секторі освіти демонструє різноманітний спектр технологічних рішень. Від ранніх платформ, таких як ASSYST та Ceilidh (CourseMarker) у 1980-х роках, які порівнювали результати виконання з контрольними значеннями, до сучасних систем на основі модульного тестування [3]. З часом з'явилися більш динамічні сервіси, такі як Online Judge та BOSS, що дозволяють автоматично перевіряти вивід програми за допомогою заздалегідь визначених тестів.

Сьогодні університети по всьому світу використовують поєднання міжнародних та місцевих платформ для автоматизованої оцінки, таких як e-olymp, Algotester, Contester, PC^2, NetOI (DOMjudge), ejudge та інші [5]. Українські дослідники та розробники також активно інтегрують платформи, на кшталт SPOJ, UVA Online Judge, TopCoder, POJ, USACO, в освітній ландшафт, узгоджуючи це зі світовою тенденцією використання систем онлайн-суддівства як у формальній освіті, так і в позакласній діяльності [9].

Сучасні рішення для оцінки коду за допомогою модульного тестування включають низку інструментів та методів, які автоматизують оцінку правильності, повноти та якості коду. Стандартною практикою є використання популярних фреймворків для модульного тестування, таких як JUnit для Java, PyTest для Python та CUnit для C, серед інших, адаптованих для різних мов програмування. Ці фреймворки дозволяють розробникам створювати та виконувати окремі тестові випадки для оцінки надійності програмних компонентів.

У сфері освіти сучасні системи управління навчанням (LMS), такі як Moodle, CascadeLMS та Sakai, розширюють свої можливості, включаючи такі функції, як CTPracticals, Automatic Grader та AutoGrader. Ці вдосконалення дозволяють автоматизувати оцінку робіт, поданих студентами. Примітним прикладом цього є Web-CAT, який не лише перевіряє точність вихідних даних, але й оцінює якість тестових випадків, створених студентами, забезпечуючи їхню ефективність та обсяг [11]. Такий цілісний підхід дозволяє викладачам оцінити глибину розуміння студентами матеріалу.

Деякі системи, такі як GitSEED, пропонують гнучкість, будучи незалежними від будь-якої конкретної мови програмування. Використовуючи GitLab CI/CD, GitSEED дозволяє користувачам переглядати код безпосередньо з кожним комітом, що дає змогу викладачам налаштовувати свої пайплайни за допомогою таких функцій, як статичний аналіз, перевірка пам'яті та виявлення плагіату за допомогою таких інструментів, як MOSS або JPlag [10, 15]. GitHub Classroom, з іншого боку, використовує дії GitHub для запуску тестів з кожним надсиланням до репозиторію, надаючи негайний зворотний зв'язок користувачам [12].

В окремій категорії можна виділити онлайн-платформи для оцінювання змагань з програмування, такі як Mooshak, DOMjudge, SPOJ, UVA Online Judge, e-olymp, PC^2 та ejudge, які функціонують як онлайн-судді. Ці платформи автоматично оцінюють рішення учасників на основі тестових даних, часто враховуючи обмеження часу та пам'яті [13, 17]. Подібні рішення набирають обертів в Україні, де такі платформи, як e-olymp, та новіші пропозиції, такі як JustClass або IT-studio, автоматизують оцінювання шкільних або олімпіадних завдань [14, 16].

У цьому контексті інтеграція світового досвіду та українських методологій була бездоганно впроваджена: ASSYST та CourseMarker відіграли ключову роль у розвитку систем онлайн-суддівства з 1980-х років, перетворившись на комплексні освітні платформи. Академічні дослідження підтвердили ефективність їх впровадження, а GitSEED та Web-CAT представляють сучасний підхід до автоматизованого оцінювання, який наголошує не лише на кінцевому результаті, а й на процесі навчання [17].

Для того, щоб надати вичерпний огляд найкращих автоматизованих систем оцінювання коду, було складено порівняльну таблицю А.1, яка розміщена в Додатку А. Таблиця містить інформацію про підтримку мов програмування, інтеграцію із системами управління навчанням (LMS), доступність інфраструктури CI/CD, тип зворотного зв'язку, вартість, доступність для навчальних закладів України та складність впровадження.

Аналіз зібраних даних дозволяє сформулювати кілька висновків. Web-CAT та GitSEED пропонують поглиблене автоматизоване тестування та аналіз коду, включаючи перевірку покриття, виявлення помилок та стилю. Однак налаштування та підтримка цих систем можуть бути складнішими. GitHub Classroom та GradeScope надають зручні вебсервіси для автоматизованого тестування, причому GitHub Classroom є безплатним для освітнього використання, а GradeScope вимагає комерційної ліцензії. Системи Mooshak та UVa Online Judge краще підходять для змагальних завдань з базовими вимогами до вводу/виводу, яким бракує розширених можливостей аналізу. JustClass – це безплатний український сервіс, призначений для створення тестових завдань з автоматичним оцінюванням. Вибір конкретного інструменту має ґрунтуватися на освітніх цілях, доступних мовах програмування, фінансових обмеженнях та технічних ресурсах навчального закладу.

Останнім часом у цю сферу починають проникати й новітні AI-рішення. Так, з’являються платформи, що застосовують штучний інтелект для автоматизації тестування, зокрема візуальної перевірки. Наприклад, Applitools пропонує сервіс на базі алгоритмів комп’ютерного зору, що зменшує кількість помилок, пов’язаних із ручним візуальним тестуванням [8]. Крім того, спостерігається дедалі більша

схильність до використання LLM та нейронних мереж для генерації тестів та аналізу коду (наприклад, GitHub Copilot, ChatGPT тощо), хоча їх інтеграція в освітні програми вимагає ретельної перевірки. Інструменти штучного інтелекту для тестування, хоча все ще перебувають на початковій стадії розробки, починають формувати практику як у професійному, так і в освітньому середовищі.

Автоматизовані інструменти перевірки коду не лише полегшують навантаження на викладачів, але й сприяють сучасному педагогічному підходу, який реагує на потреби студентів. Ці системи заохочують самостійне дослідження, сприяють глибшому розумінню концепцій та скорочують розрив між теорією та застосуванням – подібно до непомітної освітньої структури в академічній сфері.

1.3. Архітектурні засади побудови масштабованих автоматизованих систем оцінювання

У сучасних умовах цифровізації освіти розробка платформ для автоматизованої оцінки знань вимагає не лише операційної точності. Масштабованість, безпека, відмовостійкість та гнучкість також є важливими факторами. Коли справа доходить до побудови автоматизованої системи для оцінки програмного коду за допомогою модульних тестів, мікросервісний підхід виділяється як вартий уваги варіант. Ця архітектура дозволяє розробляти систему, що складається з окремих, автономних сервісів, кожен з яких має певну функцію, власне сховище даних та взаємодію з іншими компонентами, що забезпечується стандартизованим API.

Ключовою перевагою мікросервісного підходу є масштабованість, що демонструється горизонтальним масштабуванням та автономним управлінням кожним компонентом системи. Кожен сервіс працює у власному процесі, використовує окрему базу даних та може масштабуватися незалежно від інших, що забезпечує ефективний розподіл навантаження за змінних умов трафіку [24]. Ця архітектурна модель сприяє динамічному розподілу ресурсів між сильно навантаженими підсистемами без необхідності масштабування всієї системи.

Дослідження Payara підкреслює, що ізоляція сервісів та індивідуальне масштабування є фундаментальними перевагами, що підвищують гнучкість системи та адаптивність до різноманітних операційних сценаріїв [25].

Важливою характеристикою мікросервісної архітектури є ізоляція відмов. На відміну від монолітних систем, де відмова одного компонента може суттєво порушити роботу всієї системи, в мікросервісній конфігурації відмова одного сервісу не має критичного впливу на інші [28]. Це підвищує загальну надійність платформи. Наприклад, у проєктах, що використовують PostgreSQL, як зазначає Bruce Momjian, розподіл даних між окремими вузлами допомагає мінімізувати обсяг потенційних збоїв, тим самим підвищуючи відмовостійкість та оптимізуючи зусилля підтримки [23].

Мікросервіси пропонують додаткові переваги безпеки завдяки своїй моделі ізольованого доступу. Кожен сервіс має обмежену кількість зовнішніх API та застосовує власні політики автентифікації та авторизації, що ефективно знижує ризик несанкціонованого доступу до критичних компонентів системи [21]. Keycloak, програмне забезпечення для управління відкритим доступом, є популярним рішенням у сфері управління ідентифікацією. Red Hat рекламує сумісність Keycloak із сучасними протоколами безпеки OAuth2 та OpenID Connect, що спрощує його інтеграцію в архітектуру мікросервісів. Це забезпечує централізовану автентифікацію користувачів та можливість оптимізувати управління політиками доступу [26]. Altkom Software також підтримує ефективність цього підходу в забезпеченні безпеки корпоративних систем [18].

У системах з високою інтенсивністю запитів продуктивність міжсервісної взаємодії стає критично важливим фактором. Для досягнення оптимальної затримки та пропускної здатності рекомендується використовувати такі інструменти, як Redis та gRPC. Redis, потужне рішення для зберігання даних в пам'яті, забезпечує винятково низьку затримку в мілісекундному діапазоні, ефективно зменшуючи навантаження на основну базу даних [27]. І навпаки, gRPC, розроблений Google для ефективної взаємодії сервісів, використовує HTTP/2 та буфери протоколів для оптимізованої серіалізації та передачі даних [19]. Тому використання Redis для

кешування разом з gRPC для викликів сервісів є ідеальним підходом для платформ оцінки з високим навантаженням.

Ще одним важливим аспектом, який слід враховувати, є сприяння гнучким процесам CI/CD. Архітектура мікросервісів дозволяє кожному сервісу мати власний автономний життєвий цикл, де розробка, тестування, розгортання та оновлення можуть відбуватися незалежно. Такий підхід дозволяє командам працювати паралельно, прискорює процес оновлення окремих компонентів та мінімізує ризики під час впровадження змін [20]. Згідно з Центром архітектури Microsoft Azure, у таких системах команди мають можливість розгортати та підтримувати сервіси незалежно, без необхідності координації з іншими, що призводить до значного підвищення швидкості впровадження нових функцій [22].

Варто зазначити здатність мікросервісів ефективно розподіляти робочі навантаження. Завдяки конструкції, яка включає дублювання окремих сервісів, система може адаптуватися до коливань кількості користувачів та підтримувати стабільну продуктивність навіть у пікові періоди [29]. Сервіси, що зазнають більших навантажень, можуть масштабуватися незалежно, що призводить до максимального використання ресурсів. Наприклад, кластери Redis використовують архітектуру без спільного доступу з автоматичним перемиканням на резервний комп'ютер, що дозволяє горизонтальне масштабування вузлів та забезпечує надійну відмовостійкість [27].

Використовуючи передові технології розподілених систем, такі як .NET, React/Next.js, PostgreSQL, Redis, gRPC та Keycloak, інтегровані в архітектуру мікросервісів, можна розробляти високомасштабовані, безпечні та ефективні платформи для автоматизованої оцінки. Впровадження гнучких методологій DevOps, надійних механізмів кешування, асинхронних протоколів передачі даних та централізованого управління автентифікацією підвищує надійність та розширюваність системи.

Архітектурні принципи моделі мікросервісів підтверджують її доцільність у створенні автоматизованої системи оцінки коду. Ключові переваги включають масштабованість, розподіленість, гнучкість у впровадженні змін, підвищену безпеку

та відмовостійкість. Доречність цієї архітектури підтверджена аналітичними оглядами, галузевими стандартами та практиками провідних компаній у розробці хмарних сервісів. Такий підхід дозволяє створити комплексну платформу, здатну обробляти численні одночасні запити, динамічно адаптуватися до змін середовища та забезпечувати об'єктивну оцінку знань користувачів у режимі реального часу.

1.4. Методологічні засади оцінювання якості програмного коду в автоматизованих системах

Якість програмного забезпечення в сучасному інженерному підході визначається моделлю ISO/IEC 25010:2011, яка охоплює ключові характеристики, такі як функціональна придатність, надійність, продуктивність, зручність використання, безпека, сумісність [34]. Цей стандарт слугує основою для встановлення вимог до програмних систем, зокрема в галузі освітніх рішень, де якість рішень для студентів оцінюється автоматично. Українські джерела підкреслюють функціональність, зручність використання, надійність, продуктивність та здатності до супроводу як основні критерії оцінки якості коду в освітніх умовах [31]. Ці фактори є вирішальними в автоматизованих системах оцінки програмного коду, забезпечуючи об'єктивність та відтворюваність результатів тестування.

Основна функція цих систем полягає в оцінці не лише точності виконання програми, але і якості її реалізації. Це досягається за допомогою поєднання статичного та динамічного аналізу. Статичний аналіз допомагає виявити відхилення від стилістичних рекомендацій, потенційні помилки та слабкі місця в структурі коду без фактичного його виконання. Прикладами таких інструментів є ESLint для JavaScript та Pylint для Python, які забезпечують відповідність коду стандартам, чистоту реалізації та логічну узгодженість [32, 37]. SonarQube пропонує комплексне рішення для кількох мов програмування, забезпечуючи всебічну оцінку якості коду на основі метрик, пов'язаних з безпекою, складністю, повторним використанням та іншими факторами [36]. Динамічний аналіз передбачає виконання програмного коду

з тестовими вхідними даними для оцінки функціональної придатності, зазвичай за допомогою модульного тестування, при цьому результати автоматично аналізуються системою верифікації. Принципи, викладені у стандарті IEEE 829-2008, слугують основою для організації цих процедур, охоплюючи створення тестових сценаріїв, розробку планів тестування та документування результатів [33].

Покриття тестами є ключовим аспектом оцінювання, оскільки воно вимірює ступінь покриття коду тестами. Цей показник допомагає визначити відсоток логіки програми, яка була протестована, слугуючи показником надійності тестів [30]. В освітніх програмах покриття тестами може відігравати певну роль у підсумкових оцінюваннях або використовуватися як передумова для вступу. Системи оцінювання в таких контекстах можуть базуватися на коефіцієнтах завершення тестів або можуть призначати різні ваги помилкам залежно від їхньої серйозності. Стандарт ISO/IEC/IEEE 29119 пропонує категоризувати дефекти за серйозністю, щоб розрізнити їхній вплив на загальні результати тестування [35]. У поєднанні з результатами статичного аналізу цей підхід пропонує комплексний метод оцінки, який враховує як технічну точність коду, так і його дотримання стилістичних та архітектурних рекомендацій.

Дослідження автоматизованого перегляду коду показали, що ці системи зазвичай використовують комплексну модель оцінювання, що включає результати тестування, дані статичного аналізу, покриття коду та інші неупереджені показники [8]. Такий підхід забезпечує ретельну та справедливу оцінку, а також звільняє викладача від надмірного навантаження та підвищує прозорість в освітньому середовищі. Тому дотримання стандартів якості програмного забезпечення та використання багаторівневих методів перегляду коду є важливими для розробки ефективних та масштабованих освітніх платформ.

Висновки до розділу 1

У першому розділі було проаналізовано сучасні методи модульного тестування та його інтеграцію в життєвий цикл розробки. Основна увага приділяється окресленню ключових переваг використання модульних тестів для об'єктивної перевірки функціональної коректності програм. Шляхом огляду різних платформ та фреймворків, призначених для оцінки коду, визначаються спільні риси цих рішень та обговорюються ключові технічні аспекти, що впливають на вибір інструментів.

Вивчення сучасної літератури вказує на зростаючу тенденцію в освіті з програмування, де автоматизоване тестування замінює традиційне ручне тестування [4, 3]. Модульні тести відіграють центральну роль у цих системах, тоді як впровадження концепцій CI/CD та аналізу якості коду створює більш реалістичне середовище розробки [6, 7]. Українські університети також почали впроваджувати подібні рішення, використовуючи такі платформи, як E-Olymp, SPOJ та інші онлайн-журнали для виконання завдань, та ретельно відбираючи сервіси оцінювання завдань [5, 9].

У галузі автоматизація тестування стала стандартною практикою, що призводить до покращення якості програмного забезпечення та швидших циклів розробки [6]. Дослідження показали, що студенти отримують користь від такої практики, оскільки вони цінують швидкий зворотний зв'язок, що підвищує їхню залученість та успіх [4, 8]. Інтеграція інструментів, орієнтованих на штучний інтелект, у тестування відкриває нові можливості, хоча його вплив на освіту потребує подальшого дослідження.

Теоретичний огляд включав вивчення архітектурних методологій, що забезпечують масштабованість, ефективність та надійність автоматизованих систем. Акцент був зроблений на перевагах мікросервісної архітектури, таких як автономні сервіси, централізована автентифікація, кешування, високопродуктивні протоколи зв'язку та інтеграція з CI/CD. Ці рішення використовують сучасні технології та методології для побудови розподілених систем, зокрема, зосереджених на

автоматизованих освітніх платформах. Автоматизоване тестування відіграє вирішальну роль у сучасному життєвому циклі розробки програмного забезпечення (SDLC) в галузі розробки програмного забезпечення.

Впровадження CI/CD передбачає запуск автоматизованих тестів з кожною інтеграцією коду, що дозволяє раннє виявлення дефектів та підвищення стабільності релізів [6]. Згідно зі звітами GitHub, система контролю версій Git є переважним вибором серед розробників, з коефіцієнтом використання приблизно 93%, а її впровадження у вищих навчальних закладах продовжує зростати [3]. Додаткові джерела підкреслюють, що автоматизоване тестування в рамках конвеєрів CI/CD значно скорочує час виходу на ринок, забезпечуючи при цьому високу якість, оскільки тестування проводиться швидко, послідовно та відповідно до стандартизованих критеріїв. Отже, автоматизація тестування стала стандартною практикою в галузі модульного та інтеграційного тестування, а також в оцінці зовнішнього вигляду та функціональності користувацьких інтерфейсів у сучасних промислових проєктах.

Таким чином, розділ сформував цілісне теоретичне підґрунтя для подальшої розробки та реалізації системи автоматизованого оцінювання програмного коду у межах програмної платформи навчального закладу.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Вимоги до систем автоматизованого оцінювання програмного коду

Під час розробки автоматизованої системи оцінки коду чітке визначення функціональних та нефункціональних вимог є надзвичайно важливим. Ці вимоги визначають обсяг розробки, архітектуру рішення, критерії оцінки та ризики протягом життєвого циклу програмного забезпечення. Для платформи дистанційного навчання у вищому навчальному закладі, яка включає автоматичну перевірку рішень за допомогою модульних тестів, вимоги класифікуються на функціональні, нефункціональні, організаційні та інструментальні аспекти.

Система повинна забезпечувати безперебійний операційний процес для всіх учасників освітнього процесу, включаючи студентів, викладачів та адміністраторів. Студенти повинні мати можливість отримувати доступ до курсів, реєструватися, переглядати програми, виконувати завдання та автоматично отримувати результати оцінювання. Безпечний механізм авторизації з використанням протоколів OAuth2.0, є важливим для захисту персональних даних та полегшення інтеграції з іншими сервісами платформи. Крім того, система повинна зберігати прогрес студента та дозволяти йому відновити навчання з того місця, де він зупинився. Це вимагає впровадження механізмів відстеження активності та персоналізованого доступу до різних матеріалів, таких як відео, презентації, текстові лекції та практичні завдання.

Частина платформи, з погляду ролі викладача, пропонує такі функції, як створення та редагування курсів, завантаження матеріалів, постановка завдань з тестуванням та перевірка результатів. Оцінювання виконаних завдань дозволяє використовувати гібридну модель перевірки з автоматичним оцінюванням на основі тестів та ручним контролем. Інтерфейс адміністратора дозволяє централізовано керувати користувачами, курсами, категоріями та ролями доступу, забезпечуючи безперебійну роботу у великих навчальних закладах.

Основою нефункціональних характеристик системи є задоволення вимог до її продуктивності та стабільності. Ключовою вимогою є забезпечення швидкого доступу до освітніх ресурсів, з метою обмеження часу відповіді на пошуковий запит курсу до однієї секунди та загального часу завантаження сторінки до трьох секунд. Це стає особливо важливим у конкурентному середовищі онлайн-платформ, де користувацький досвід відіграє значну роль в ефективності системи у сприянні освітньому процесу. Щобільше, система повинна бути здатною масштабуватися та підтримувати щонайменше 1000 одночасно активних користувачів. Для досягнення цієї мети необхідні сучасні методології, такі як побудова мікросервісної архітектури, реалізація балансування навантаження та оптимізація кешування.

З технічного погляду, адаптивний дизайн інтерфейсу відіграє вагому роль у забезпеченні коректного відображення системи на різних пристроях, включаючи смартфони, планшети та персональні комп'ютери. Важливо дотримуватися правил захисту даних, зокрема GDPR, шляхом впровадження шифрування, резервного копіювання, контролю доступу та безпечної передачі інформації через HTTPS. Інтеграція зовнішніх залежностей, таких як Keycloak, повинна виконуватися з максимальною надійністю, щоб гарантувати завершення авторизації користувача протягом 2 секунд.

На організаційному рівні система повинна гарантувати стабільну доступність (мінімум 99,5% часу безвідмовної роботи) та мати резервні резерви для відновлення даних у разі технічних збоїв. Крім того, ключовими міркуваннями є впровадження механізмів CI/CD та підтримка незалежного оновлення компонентів системи без порушення загальної роботи.

Визначаючи системні вимоги, важливо враховувати не лише функціональні та технічні аспекти. Не менш важливо оцінити потенційні ризики, які можуть вплинути на надійність, масштабованість та безпеку системи. Ці ризики можуть включати проблеми з продуктивністю через розширення бази користувачів, нестабільність сервера або збої в зовнішніх сервісах. Крім того, існує ризик вразливостей, що виникають через застарілі бібліотеки або непомічені помилки коду в результаті неадекватних процедур тестування. Ризики, пов'язані з інструментами, пов'язані з

вибором інструментів розробки, їх конфігурацією та ефективністю. Такі проблеми, як недостатня підтримка або проблеми з інтеграцією інструментів CI/CD, ведення журналу або моніторингу, можуть перешкоджати швидкості виявлення проблем та цілодобовій підтримці системи.

Для глибшого вивчення впливу різних категорій факторів на ефективність системи було побудовано діаграму Ісікави(див. Додаток Б, Рисунок Б.1). Ця діаграма сприяла організованій категоризації ризиків та оцінці їхньої значущості в ширшій моделі. Отримані дані слугують основою для прийняття обґрунтованих рішень щодо архітектурного та функціонального проєктування платформи.

Аналіз вагових коефіцієнтів дозволив нам кількісно оцінити значущість кожної категорії та її відповідних підкатегорій для загальної ефективності системи. Особливої уваги заслуговує категорія «Технології», яка отримала найвищу питому вагу 0,25, що підкреслює вирішальну залежність якості онлайн-навчання від стабільності та продуктивності платформи. Помітний вплив мав фактор «Проблеми платформи» (0,13), який охоплює такі проблеми, як повільна робота, збої та технічна нестабільність.

Педагогічні підходи (0,20) та людський фактор (0,20) також відіграють значну роль, що підкреслює важливість коригування освітніх стратегій для онлайн-навчання та підвищення цифрових навичок освітян. Серед окремих факторів особливо варто відзначити «Неадекватні методології» (0,10) та «Обмежений рівень володіння цифровими технологіями серед вчителів» (0,07).

Хоча організаційні процедури (0,15) мають дещо менш виражений вплив, вони все ж вимагають стандартизації критеріїв та покращення допомоги студентам. Зовнішні впливи середовища (0,05) мають найменший вплив на ефективність, проте вони все ще можуть мати ситуативний вплив, попри те, що вони знаходяться поза контролем платформи.

Розроблена модель не лише визначила ключові сфери для втручання, але й окреслила пріоритетні дії для розвитку: визначення пріоритетів таких завдань, як стабілізація платформи, адаптація методів та підвищення цифрової компетентності.

Отже, ретельна стратегія визначення вимог та оцінки ризиків слугує основою для розробки надійної, адаптивної та безпечної автоматизованої системи оцінки коду. Ця система повинна не лише відповідати технічним потребам сучасного освітнього середовища, але й мати адаптивність до модифікацій, стійкість до потенційних ризиків та безперешкодну інтеграцію з іншими цифровими навчальними ресурсами.

2.2. Архітектура та функціональна модель системи

У сфері побудови сучасних веборієнтованих систем, наявність чітко визначеного архітектурного плану, який гарантує масштабованість, гнучкість та дотримання як функціональних, так і нефункціональних критеріїв, має важливе значення. Для створення платформи онлайн-курсів університету було прийнято рішення використовувати мікросервісну архітектуру, організовану в клієнтські та серверні компоненти. Цей стратегічний вибір відображає необхідність забезпечення одночасного використання кількома користувачами, сприяння модульності та полегшення майбутньої підтримки системи.

Для узгодження функціональних очікувань користувачів із запланованими архітектурними рішеннями було використано метод розгортання функцій якості (QFD). Цей систематичний підхід включав аналіз взаємозв'язків між вимогами основних груп користувачів (студентів, викладачів, адміністрації) та технічними характеристиками розробленої системи.

На початковому етапі основні потреби користувачів були визначені та впорядковані на основі їхньої значущості (див. Додаток В, Таблиця В.1). До пріоритетів належать: зручний інтерфейс, надійна система тестування, сумісність з мобільними пристроями та безпека даних.

Після цього було складено перелік суттєвих технічних характеристик системи для задоволення заданих вимог. А також створено матрицю залежностей, щоб показати вплив кожної технічної характеристики на задоволення відповідної потреби

користувача. Для оцінки міцності зв'язку використовувалася шкала від 0 (відсутній) до 9 (сильний). Результати можна знайти в Додатку В, Таблиця В.2.

Крім того, було проведено дослідження зв'язків між технічними атрибутами для визначення синергії або конфліктів, як показано на «даху» Будинку якості (див. Додаток В, Таблиця В.3). Згодом, на основі наданих даних було розраховано остаточну вагу для кожної технічної характеристики, що дозволило визначити пріоритети їх впровадження з погляду функціональності (див. Таблицю 2.1).

Таблиця 2.1

Підсумкова важливість технічних характеристик,
визначена за результатами моделі QFD

Технічна характеристика	**5**	**4**	**5**	**4**	**3**	**5**	**4**	**Сума ваги**
Архітектура клієнт-сервер	3	9	9	3	3	3	0	132
Модулі аутентифікації/авторизації	3	3	0	1	3	9	0	85
Адаптивна верстка	9	3	0	0	0	1	9	98
Інтеграція з системою тестування	1	3	9	3	3	3	0	98
Аналітичний модуль	0	0	3	1	9	3	0	61
Хмарне зберігання	0	1	3	1	3	3	0	47
Резервне копіювання та шифрування	0	0	1	0	0	9	0	50

Джерело: зроблено автором

Результати аналізу підкреслюють значний вплив архітектури «клієнт-сервер» на технічну структуру системи. Вона демонструє сильний зв'язок з такими важливими функціями, як модулі автентифікації, автоматизовані системи тестування, аналітичні модулі, хмарне сховище даних, резервне копіювання та механізми шифрування. Ці результати свідчать про те, що обрана архітектурна модель сприяє безперешкодній інтеграції функціональних компонентів, централізованому управлінню та покращеному контролю процесів.

Кореляція між підсистемами автоматизованого тестування, хмарного сховища результатів та аналітичних модулів є помітно сильною. Це зумовлено прямим технологічним взаємозв'язком: результати модульних тестів можуть бути використані як вхідні дані для формування звітів, аналізу навчального процесу та

встановлення зворотного зв'язку зі студентами. Крім того, інтеграція надійних механізмів резервного копіювання та шифрування позитивно впливає майже на всі компоненти, що обробляють персональні або освітні дані.

З іншого боку, спостерігалися конфліктні зв'язки, зокрема між модулями автентифікації/авторизації та адаптивністю інтерфейсу. Цей тип конфлікту поширений в інформаційних системах із суворими вимогами безпеки, оскільки використання захисних заходів (таких як CAPTCHA або багатофакторна автентифікація) може перешкоджати зручності використання, особливо на мобільних пристроях або в адаптивному режимі.

Графічне представлення моделі «Будинок якості» можна знайти в Додатку В, Рисунок В.1. Ця візуалізація допомагає користувачам ефективно контролювати взаємозв'язки між елементами системи та підтримувати прозорість прийняття рішень. Результати побудови та аналізу діаграми Будівлі якості підкреслюють важливість цілісного підходу до архітектурного проєктування системи. Встановлені зв'язки дозволяють нам ефективно визначати пріоритети технічних функцій, підтримувати внутрішню узгодженість платформи та оптимізувати розподіл ресурсів під час її еволюції та вдосконалення.

Для реалізації визначених зв'язків між основними технічними характеристиками, оптимальною стратегією є використання мікросервісної архітектури. Такий вибір пропонує необхідну гнучкість, масштабованість та автономність у розгортанні елементів системи. Дослідження в науковій та інженерній літературі показали, що мікросервіси можуть підвищити адаптивність програмного забезпечення, дозволяючи кожному сервісу зосередитися на певному наборі функцій та забезпечуючи незалежне розгортання, масштабування та оновлення [39, 40].

Перевагою обраної архітектури є незалежність окремих компонентів, що призводить до підвищеної стійкості системи до збоїв. Отже, якщо в одному сервісі виникає критична помилка, решта системи продовжує працювати без перебоїв. Ця функція особливо важлива для освітніх платформ, які потребують безперебійної доступності та стабільності в періоди пікового навантаження, такі як дедлайни або

велика кількість студентських робіт. Такий підхід відповідає принципам проєктування, орієнтованого на предметну область, де кожен мікросервіс представляє окрему предметну область, таку як управління курсами, перевірка коду та авторизація.

Запропонована архітектурна реалізація базується на концепції взаємодії клієнт-сервер, яка далі сегментована на мікросервіси. Інтерфейс користувача розроблено на платформі Next.js і служить фронтенд додатком, відповідальним за демонстрацію освітнього контенту, обробку введених користувачем даних та зв'язок із сервером через REST API. Централізована авторизація та автентифікація забезпечуються за допомогою Keycloak, спеціалізованого сервісу, що включає протокол OpenID Connect, пропонуючи єдину точку доступу для студентів, викладачів та адміністративного персоналу в системі.

Бекенд платформи складається з двох основних мікросервісів, розроблених з використанням технологій .NET. Перший сервіс, названий як HomeAssignment, служить центральним координатором платформи. Він займається управлінням освітнім контентом, таким як курси, розділи та завдання, а також записом спроб користувачів. HomeAssignment також взаємодіє з базою даних PostgreSQL та системою кешування Redis для підвищення ефективності обробки запитів.

Другий мікросервіс, RepoAnalysis, відповідає за конкретне завдання перевірки користувацьких рішень, поданих у вигляді репозиторіїв GitHub. Цей сервіс клонує репозиторії, компілює код, виконує модульні тести, генерує результати тестів і надсилає ці дані до HomeAssignment. Зв'язок між HomeAssignment та RepoAnalysis забезпечується gRPC, що забезпечує швидку та ефективну передачу даних у серіалізованому форматі.

Описаний архітектурний дизайн пропонує кілька суттєвих переваг. Однією з ключових переваг є можливість незалежного масштабування сервісів на основі попиту, що підвищує продуктивність системи. Розділення функцій авторизації (Keycloak) та перевірки рішень (RepoAnalysis) підвищує безпеку та надійність у разі збоїв. Централізоване зберігання даних у PostgreSQL та кешування за допомогою Redis допомагають зменшити затримки та пришвидшити реакцію клієнтів під час

взаємодії з API. Цей підхід має спільні риси з освітніми платформами з відкритим кодом, такими як Moodle та Open edX, але наша система виділяється своєю спрямованістю на автоматизовану обробку рішень завдяки інтеграції з GitHub, розширюючи свій потенціал для інженерних та програмних дисциплін.

На рисунку 2.2 зображено архітектурну модель системи, що демонструє основні компоненти та взаємозв'язки між ними.

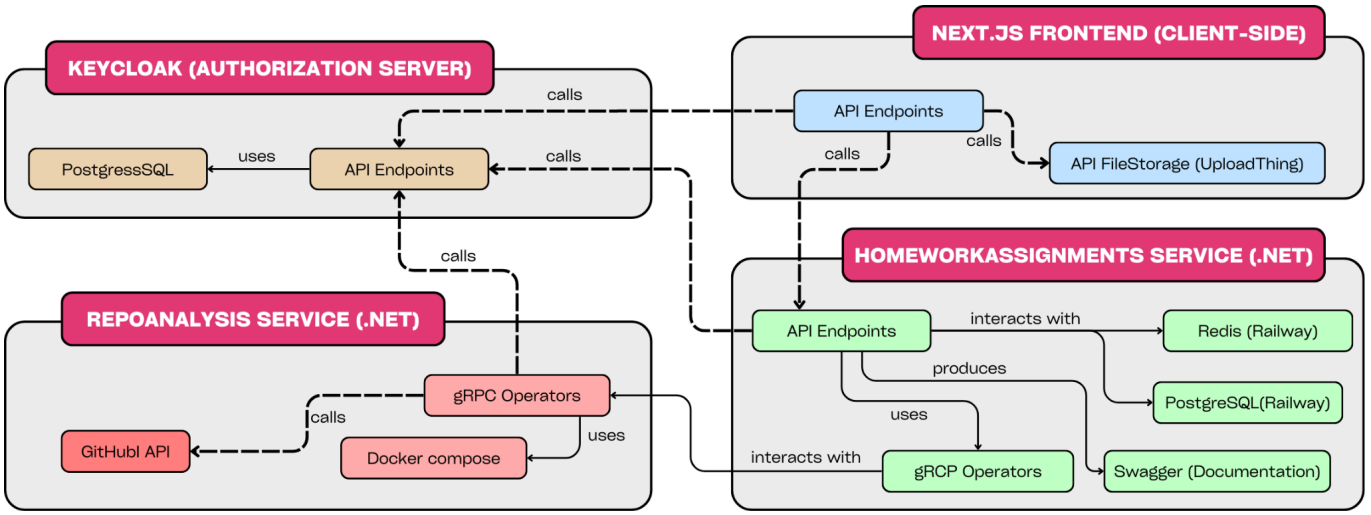

Рис. 2.2. Архітектура системи взаємодії сервісів.

Джерело: зроблено автором

Таким чином, обрана архітектура ефективно відповідає потребам сучасної освітньої платформи, яка робить акцент на автоматизованих процесах оцінювання. Впровадження мікросервісного дизайну з чітко визначеними функціональними доменами забезпечує підвищену гнучкість розробки, розширену функціональність, оптимізовану підтримку та адаптивність до середовищ розгортання, що змінюються.

2.3. Алгоритми обробки та оцінювання результатів юніт тестів

Для автоматизованих навчальних систем, особливо під час оцінювання програмних рішень, важливо встановити чіткі, прозорі та обґрунтовані алгоритми для аналізу результатів виконання коду студентами. Основною метою цих алгоритмів є гарантування об'єктивності, відтворюваності та технічної надійності в процесі оцінювання. Це особливо важливо у сфері електронних навчальних платформ, де перевірка рішень має проводитися автоматично, мінімізуючи потребу в постійному втручанні викладача.

Алгоритми обробки включають кілька логічно впорядкованих етапів: компіляцію проєкту, аналіз якості коду та оцінку результатів модульного тестування. Ця методологія відображає стандартний життєвий цикл розробки програмного забезпечення, гарантуючи, що навчальне тестування тісно відповідає практичним потребам сучасного виробництва програмного забезпечення.

Початковим і фундаментальним етапом студентського проєкту є процес компіляції, який є важливим для проведення подальшого тестування або оцінки якості. Головною метою компіляції є перетворення вихідного коду на виконуваний файл. Під час цього етапу система перевіряє відповідність вихідного коду мовним стандартам, наявність усіх необхідних залежностей та базову структурну правильність. Компілятор відповідає за виправлення будь-яких синтаксичних, логічних та семантичних помилок, які можуть перешкоджати виконанню програми.

Після компіляції алгоритм оцінює, чи був процес успішним. У випадках критичних помилок, таких як відсутні типи, неправильні вхідні параметри або несумісність типів даних, оцінювання зупиняється. Подальші дії, включаючи заходи контролю якості або тестові запуску, вважаються недоречними та не виконуються. Стабільність системи верифікації гарантується цим механізмом, який ефективно запобігає виконанню нефункціонального коду. Ця функція є критично важливою для автоматизованих платформ, що обробляють численні одночасні спроби верифікації.

Після успішної компіляції система переходить до наступного етапу, який включає аналіз якості коду. Метою цього етапу є виявлення стилістичних, потенційно проблемних або нефункціональних помилок, які не перешкоджають компіляції, але впливають на загальну якість рішення. Ці помилки можуть включати непотрібні залежності, дублювання коду, порушення правил іменування змінних, відсутність документації, складні умовні вирази тощо.

Для виявлення таких проблем використовуються інструменти діагностики компілятора або сторонні аналізатори коду, такі як Pylint, аналізатори Roslyn, Maven (залежно від мови програмування, що використовується). Кожне повідомлення класифікується за рівнем значущості.

- Error — вказує на серйозну проблему, яка може призвести до збою або є неприйнятною з архітектурного погляду;
- Warning — позначає потенційно ризиковану ситуацію, хоча й у межах прийнятних параметрів, яка потребує розгляду;
- Information — надає цінну інформацію або поради щодо покращення стилю чи продуктивності.

Оцінка якості коду проводиться з використанням методу зваженого зниження оцінки на основі серйозності діагностичних повідомлень. Кожна помилка або попередження впливає на загальну оцінку пропорційно до її важливості. Наприклад, починаючи з максимальної оцінки 100, бали віднімаються на основі ваги категорії помилки.

Загальна формула, яка використовується для визначення показника якості, наведена нижче:

$$Q = 100 - (E \cdot w_e + W \cdot w_w + I \cdot w_i), \quad (1)$$

де:

- 100 — максимальне значення шкали оцінювання у відсотках;
- Q — фінальний бал процесу оцінювання якості;
- E, W, I — кількість помилок, класифікованих як Error, Warning, Information;
- w_e, w_w, w_i — вагові коефіцієнти, що визначають вплив кожної категорії помилок на загальний бал.

Використання цієї методології дозволяє досягти гармонійного балансу між функціональністю та якістю. Вона підкреслює ідею про те, що, попри технічно обґрунтовану реалізацію, ігнорування стандартів кодування або наявність стилістичних недоліків незмінно вплине на кінцевий результат.

Підсумковий етап перевірки передбачає виконання модульних тестів, тобто окремих тестових скриптів, призначених для перевірки відповідності функціональності коду заданим критеріям. Цей етап зосереджений на неупередженій оцінці того, чи відповідає реалізація вимогам завдання.

Зазвичай тестові файли зберігаються у визначеному каталозі, такому як «tests», і містять стандартні конструкції із сумісних тестових бібліотек, таких як xUnit, JUnit або PyTest. Ці тести виконуються в ізольованому контейнері Docker, щоб забезпечити цілісність середовища та запобігти потенційним побічним ефектам.

Функціональна правильність рішення визначається кількістю успішно пройдених тестів. Досягнення ідеального балу за всі тести призводить до максимального балу для поточного етапу. Однак часткові невдачі призведуть до пропорційного зниження балу. Формула для розрахунку функціонального балу наведена нижче:

$$S = \frac{T_p}{Tt} \cdot 100, \quad (2)$$

де:

- S — фінальний бал процесу оцінювання функціональності;
- T_p — кількість успішно пройдених тестів;
- Tt — загальна кількість тестів рішення;
- 100 — максимальне значення шкали оцінювання у відсотках.

Впроваджуючи послідовну систему зважування на всіх етапах аналізу, ми забезпечуємо чітке та просте для сприйняття формування результатів для різних спроб та студентів. Бездоганне виконання коду та успішне завершення всіх тестів призводить до максимального балу, попередньо визначеного викладачем. У разі невдалого проходження хоча б одного тесту, механізм оцінювання коригується пропорційно залежно від кількості пройдених тестів.

Система забезпечує студентів чіткою мотивацією, показуючи відсоток правильно реалізованої функціональності, а також слугує корисним інструментом для викладачів, щоб швидко виявляти поширені проблеми в рішеннях студентів. Крім того, для численних завдань систему можна налаштувати для розрахунку ваги кожного тесту, що дозволяє тестам з більшою вагою мати більший вплив на оцінювання. Хоча для вступних курсів рекомендується проста модель 1 тест = 1 бал, для складніших випадків можна ввести вагові коефіцієнти.

Після завершення всіх етапів перевірки результати оцінювання об’єднуються в єдину структуру. Ця структура включає загальний бал, бали за кожен окремий етап (компіляція, якість, функціональність), детальні повідомлення про помилки та позначки часу виконання. Усі дані надійно зберігаються в базі даних платформи та використовуються для створення звітів для викладача.

Під час роботи з системою, яка підтримує кілька мов програмування, таких як Java, Python та C#, важливо ретельно продумати архітектуру. Запропонований підхід підвищує гнучкість навчального процесу, дозволяючи студентам використовувати мову, яку вони вивчають або з якою почуваються найкомфортніше. Однак це також створює труднощі у впровадженні універсальних алгоритмів верифікації.

Деякі ключові аспекти, які слід враховувати при багатомовному аналізі, включають:

- Ізольоване середовище виконання необхідне для того, щоб кожна мова мала власне середовище компіляції та запуску. Це досягається за допомогою контейнерів Docker, попередньо налаштованих з інтерпретаторами, компіляторами та бібліотеками. Наприклад, Java використовує OpenJDK, Python використовує CPython з pytest, а C# використовує .NET SDK з xUnit.
- Механізм компіляції, що залежить від мови, повинен враховувати відмінності у форматах проєктів, структурах каталогів та залежностях. Кожна мова має своє унікальне представлення файлу проєкту, таке як .csproj для C#, pom.xml або build.gradle для Java, а Python може взагалі не мати явного файлу проєкту. Ці відмінності необхідно враховувати на етапі підготовки до запуску алгоритмів.
- Для різних мов програмування існує специфічний набір аналізаторів для оцінки якості коду:
    - C# : Аналізатори Roslyn, StyleCop;
    - Java : Checkstyle, PMD;
    - Пайтон : Pylint, Flake8.

Система повинна динамічно пов'язувати відповідні інструменти для кожної мови та стандартизувати результати за допомогою єдиної системи оцінювання ваг.

- Адаптація тестів до мови виконання є важливою в багатомовних середовищах. Вкрай важливо мати відповідні реалізації тестів для кожної використовуваної мови. Це може означати наявність окремих тестів для одного завдання, таких як JavaTest.java, SolutionTests.cs, test_solution.py. Система повинна точно поєднувати реалізацію студента з відповідними тестами та виконувати лише ті, що відповідають дійсності.
- Стандартизація результатів бази даних та журналу спроб. Незалежно від мовної різноманітності, результати повинні реєструватися в єдиному форматі, включаючи відсотки, ідентифікацію помилок та кількість пройдених тестів. Така стандартизація забезпечує порівнянність та спрощує візуалізацію статистики.

Врахування цих факторів дозволяє впровадити надійну, гнучку та масштабовану платформу оцінювання коду, яка є зручною для викладачів та легко зрозумілою для студентів, які використовують різні мови програмування.

2.4. Визначення вхідних та вихідних даних у процесі оцінювання коду

Ефективне функціонування автоматизованої системи оцінювання коду значною мірою залежить від чітко визначеної структури вхідних та вихідних даних. Ці дані, в контексті створення платформи онлайн-курсів для студентів ІТ-фахівців, які займаються дистанційним навчанням, слугують основою для взаємодії між користувачем, серверним компонентом системи та механізмом оцінювання.

На поточному етапі реалізації системи основним джерелом вхідних даних є репозиторій з відкритим кодом, розташований на платформі GitHub. Щойно студент надсилає рішення на перевірку, система автоматично клонує відповідну гілку репозиторію, запобігаючи змінам вмісту коду після закінчення терміну виконання завдання. Цей механізм забезпечує прозорість та контроль процесу подання. Кожна

спроба відрізняється унікальним UUID користувача та ідентифікатором завдання, що встановлює зв'язок між окремим студентом та тестовим сценарієм, а також полегшує моніторинг динаміки навчального прогресу.

У таблиці 2.2 окреслено основні вхідні дані, які відіграють вирішальну роль у процесі оцінки коду.

Таблиця 2.2

Вхідні дані в процесі оцінки коду

Параметр	**Опис**
GitHub-репозиторій	Код студентського рішення у вигляді окремої гілки
Ідентифікатор студента	Унікальний UUID, що використовується для прив’язки до акаунта
Ідентифікатор завдання	Системний ключ для визначення тестового сценарію
Мова програмування	Один із підтримуваних інтерпретаторів: C#, Java, Python
Дата/час спроби	Фіксується автоматично при поданні коду

Джерело: зроблено автором

Усі перевірки виконуються в рамках стандартизованого шаблону для забезпечення узгодженості, хоча це може обмежувати гнучкість у певних випадках. З усім тим, формат вхідних даних дозволяє масштабувати логіку перевірки. Коригуючи конфігурацію сервера, а не самі рішення, сценарії можна легко змінювати, не порушуючи загальної структури.

Система оцінювання використовує багатоетапну модель для аналізу результатів виконання коду. Після завершення тесту створюється набір вихідних даних, які слугують основою для зворотного зв'язку. Основним компонентом є звіт про результати оцінювання, який надає узагальнену інформацію про продуктивність на окремих етапах тестування: компіляції, тестуванні та аналізі якості коду. Звіт містить такі деталі, як стан виконання, отриманий бал та допустимий діапазон значень для кожного етапу. Бал автоматично розраховується на основі заздалегідь

визначених вагових коефіцієнтів, встановлених викладачем, що дозволяє створювати індивідуальну модель оцінювання, яка враховує складність завдання та освітні цілі.

Такий підхід забезпечує баланс між наданням корисної інформації, забезпеченням стабільності та простотою інтерфейсу користувача. Система надає пріоритет важливим аспектам навчального завдання, уникаючи зайвих технічних деталей, щоб уникнути перевантаження користувача.

У таблиці 2.2 показано загальну структуру звіту про результати оцінювання, де конкретні числові значення змінюються залежно від спроби студента та параметрів викладача.

Таблиця 2.2

Вихідні дані в процесі оцінки коду

Назва параметра	**Тип значення**	**Джерело формування**	**Призначення**
Статус спроби	Текст (Завершено: загальний бал / максимальний бал)	Результат повного циклу перевірки	Результат повного циклу оцінювання, що визначає загальний бал спроби.
Компіляція	Пройдено / Провалено	Аналіз логування компіляції	Результат процесу компіляції вказує на те, чи успішно було скомпільовано проєкт.
Тести	Пройдено / Провалено	Результати запуску юніт-тестів	Результати запуску модульних тестів показують, чи були вбудовані тести успішними.
Якість коду	Пройдено / Провалено	Інструменти аналізу якості коду	Аналіз якості коду за допомогою спеціальних інструментів для перевірки дотримання критеріїв.
Бал за компіляцію	Ціле число	Налаштування завдання викладачем	Представляє оцінку за успішну компіляцію, з вагою, визначеною викладачем.
Бал за тести	Ціле число	Процент кількості успішних тестів	Вказує на кількість успішних тестів, з автоматично розрахованим балом на основі тестів.
Бал за якість	Ціле число	Базується на класифікації порушень за рівнем критичності	Представляє оцінку аналізу коду, розраховану відповідно до правил якості, до якої може бути застосовано зважування.

Продовження табл. 2.2

Загальний бал	Ціле число	Агрегація вищеописаних балів	Загальний результат спроби з урахуванням коефіцієнтів
Кількість спроб	Ціле число	База даних	Служить для контролю обмеження повторних перевірок
Час виконання	Часовий штамп	База даних	Зберігається в журналах для аналітичних цілей.
Повідомлення про помилки	Текст або null	Сервер/логування компіляції чи тестів	Внутрішня інформація для налагодження, недоступна студенту

Джерело: зроблено автором

Таким чином, вихідні дані, отримані за допомогою автоматизованого оцінювання, є продуктом складної взаємодії різного походження: результатів компіляції, виконання тестів, записів перевірки коду та параметрів, встановлених викладачем. Завдяки застосуванню цього методу стає можливим адаптувати процедуру оцінювання до унікальних характеристик кожного завдання, зберігаючи при цьому узгодженість, відкритість та масштабованість. Ця стратегія закладає основу для майбутніх удосконалень функціональності, зокрема, з погляду надання детальної аналітики результатів викладачеві, збереження повної історії спроб та забезпечення гнучкого налаштування моделі оцінювання.

Висновки до розділу 2

У другому розділі було проведено поглиблене дослідження предметної області, що дозволило нам окреслити основні функціональні компоненти автоматизованої системи оцінки коду, побудованої на модульних тестах. У ньому окреслено необхідні вхідні та вихідні дані для належної роботи механізмів верифікації, такі як метадані про рішення студентів, результати компіляції, виконання тестів та оцінка якості коду. Прийняття чітко визначеної структури даних дозволяє стандартизувати підхід до зберігання та обробки результатів верифікації.

Особливий акцент робиться на аналізі ключових фаз автоматизованої верифікації, починаючи від отримання коду до репозиторію і закінчуючи передачею результатів викладачеві. Запропонована структура охоплює компіляцію, виконання модульних тестів, оцінку якості коду з урахуванням зважених коефіцієнтів та обчислення підсумкової оцінки. Ця методологія забезпечує об'єктивність результатів, враховуючи як технічну точність, так і стиль програмування.

Було узагальнено основні функції компонентів системи, з акцентом на ролі модулів управління завданнями, виконання верифікації, взаємодії з GitHub та зберігання результатів. Було визначено функціональні зв'язки між рівнями інфраструктури, застосунку та домену, кожен з яких сприяє реалізації механізмів оцінки та забезпечує масштабованість, відмовостійкість та розширюваність.

Аналіз підтвердив життєздатність мікросервісного підходу, структурованої організації даних та чіткого розмежування відповідальності між компонентами. Це створює міцну основу для ефективного впровадження та постійного вдосконалення системи, гарантуючи надійність та прозорість процесу оцінювання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ОЦІНЮВАННЯ ПРОГРАМНОГО КОДУ

3.1. Архітектурно-технологічні рішення та вибір інструментів

Під час етапу впровадження автоматизованої системи оцінки коду було обрано набір технологічних інструментів, які відповідають функціональним та нефункціональним вимогам, викладеним у розділі 2. Ключовими факторами вибору були масштабованість, підтримка мікросервісної архітектури, наявність сучасних інструментів інтеграції та автоматизації, активна спільнота розробників та високоякісна технічна документація.

Для реалізації серверної частини було обрано платформу .NET 9, відому своєю високою продуктивністю, кросплатформною підтримкою та надійною інфраструктурою для побудови REST та gRPC-сервісів. Основною мовою програмування, що використовується для реалізації серверної логіки, є C#, відома своєю статичною типізацією, об'єктно-орієнтованим підходом та розгалуженою екосистемою бібліотек і фреймворків. Архітектура системи, як показано на рисунку 3.1, розроблена як монолітний вебзастосунок, який може ізолювати ключові компоненти (такі як модуль оцінки коду) в мікросервіси, використовуючи gRPC для спрощеного розгортання та масштабованості.

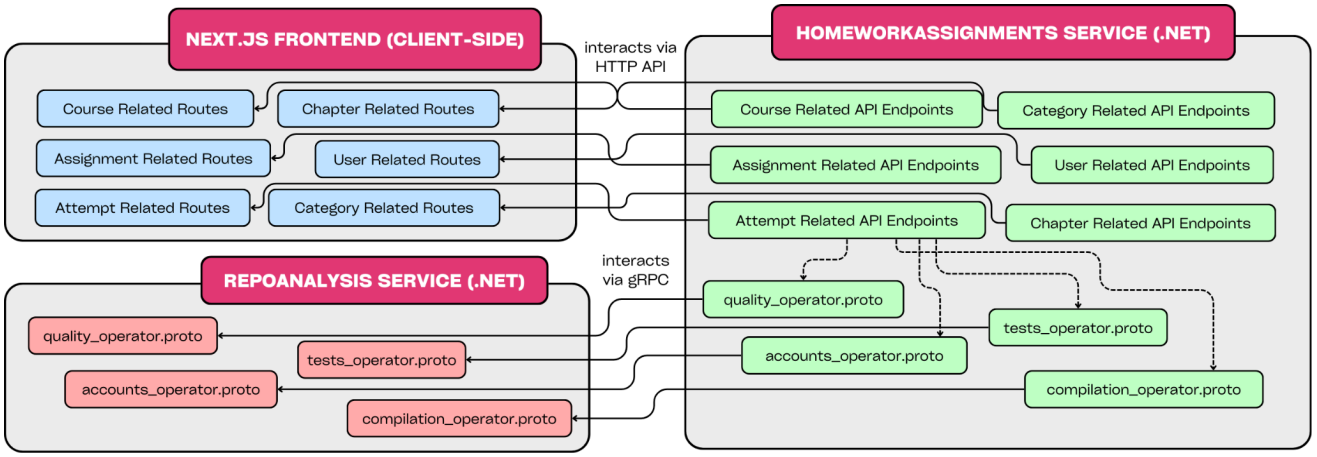

Рис. 3.1. Функціональна схема взаємодії компонентів системи через HTTP API та gRPC.

Джерело: зроблено автором

Клієнтська частина системи використовує React 18.3.1 з Next.js 15.2.4 для ефективного рендерингу на сервері, безперебійної маршрутизації та гнучкої організації компонентів інтерфейсу. Використання TypeScript підвищує безпеку типів та зменшує кількість помилок, які можуть виникнути на етапі компіляції. Це пов'язано з функцією статичної типізації, яка дозволяє розробникам виявляти поширені помилки до виконання, що призводить до створення більш надійного та стабільного коду.

Система використовує зовнішній сервіс під назвою Uploadthing для безперешкодного та безпечного зберігання файлів користувачів, включаючи фотографії, відео та вкладення курсів. За допомогою Uploadthing файли можна ефективно обробляти та передавати з боку клієнта, тим самим перекладаючи відповідальність за зберігання на хмарну інфраструктуру. Такий підхід допомагає зменшити навантаження на сервер і підвищує масштабованість, особливо під час роботи з великими обсягами контенту, створеного користувачами. Клієнтська програма містить бібліотеку uploadthing для керування файлами, що дозволяє користувачам вказувати типи файлів, обмеження та логіку завантаження. У наведеному нижче лістингу представлено конфігурацію маршрутизатора файлів (FileRouter), що використовується для завантаження зображень, відео та вкладень курсів.

Лістинг 3.1. Конфігурація маршрутизатора файлів Uploadthing у Next.js

```
import { createUploadthing, type FileRouter } from "uploadthing/next";

const f = createUploadthing();

export const ourFileRouter = {
    courseImage: f({ image: { maxFileSize: "4MB", maxFileCount: 1 } })
        .onUploadComplete(() => {}),
    courseAttachment: f(["text", "image", "video", "audio", "pdf"])
        .onUploadComplete(() => {}),
    chapterVideo: f({ video: { maxFileSize: "8GB", maxFileCount: 1 } })
        .onUploadComplete(() => {}),
} satisfies FileRouter;

export type OurFileRouter = typeof ourFileRouter;
```

Для автентифікації та контролю доступу вбудовано Keycloak 21.0.1, який пропонує підтримку протоколів OAuth 2.0 та OpenID Connect. Служба автентифікації працює як окремий елемент інфраструктури, взаємодіючи з додатком .NET через компоненти проміжного програмного забезпечення для централізованого керування сеансами, ролями та правами доступу. Підтримка часткової реплікації облікових записів від зовнішніх постачальників OIDC, наприклад GitHub, була інтегрована в основні механізми автентифікації Keycloak. Дана функціональність дозволяє отримувати та зберігати основні атрибути користувача (наприклад, адресу електронної пошти, ім'я, ідентифікатор) від зовнішнього постачальника під час початкового процесу автентифікації. Попередньо визначені атрибути зберігаються локально в Keycloak без необхідності повної синхронізації або дублювання профілів.

На рисунку 3.2 показано налаштування Keycloak, необхідні для безперебійної інтеграції з зовнішнім провайдером GitHub.

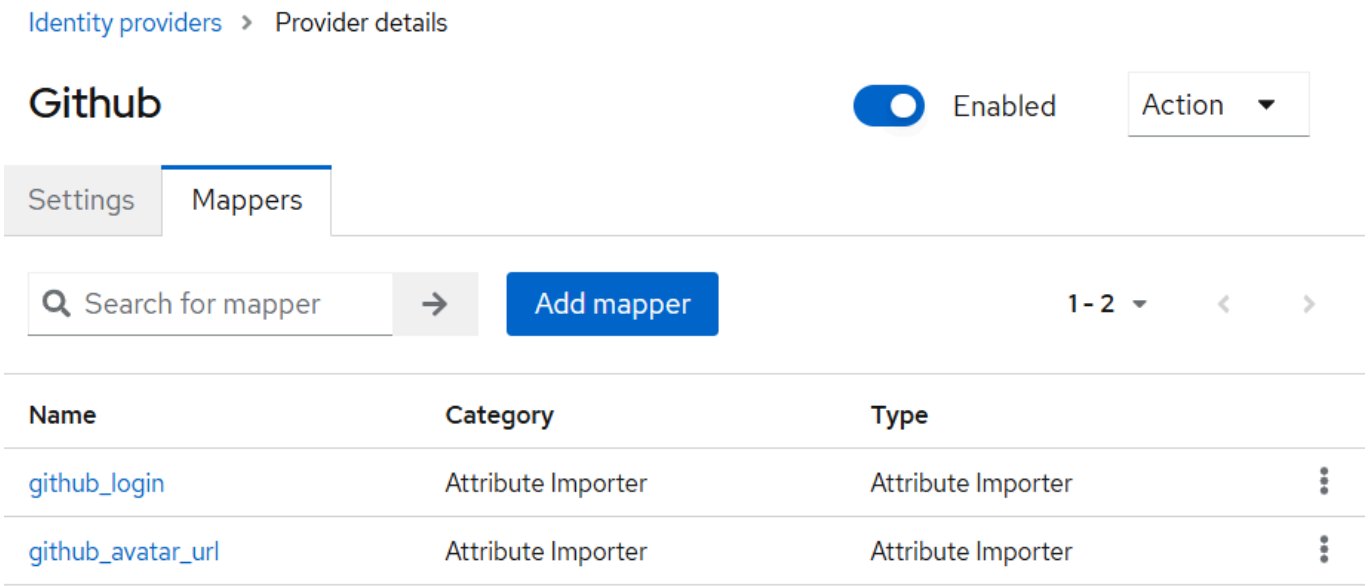

Рис. 3.2. Налаштування інтеграції Keycloak з GitHub провайдером.

Джерело: зроблено автором

Виступаючи посередником між платформою онлайн-курсів та сторонніми постачальниками автентифікації, Keycloak спрощує процес входу, не вимагаючи від користувачів окремої реєстрації. Такий централізований підхід дозволяє Keycloak керувати ролями, політиками доступу та журналами входу, мінімізуючи ризики, пов'язані з розбіжностями в профілях. Часткова синхронізація даних користувача

відбувається лише під час автентифікації, що усуває необхідність частих оновлень із зовнішніх джерел та зменшує дублювання даних.

PostgreSQL 15 служить основною СУБД, забезпечуючи високий рівень гарантії атомарності та узгодженості операцій, масштабованості та гнучкості. База даних платформи структурована відповідно до принципів нормалізації. Основу якої складають сутності, що представляють користувачів, курси, завдання, спроби та прогрес користувачів. Така структура дозволяє гнучко зберігати навчальні матеріали, контролювати виконання завдань та оцінювати результати. Детальну схему таблиць та зв'язків, що використовуються в системі, представлено в Додатку Г на рисунку Г.1.

На рисунку 3.3 показано взаємодію між основною базою даних системи та базою даних Keycloak під час автентифікації та синхронізації користувачів.

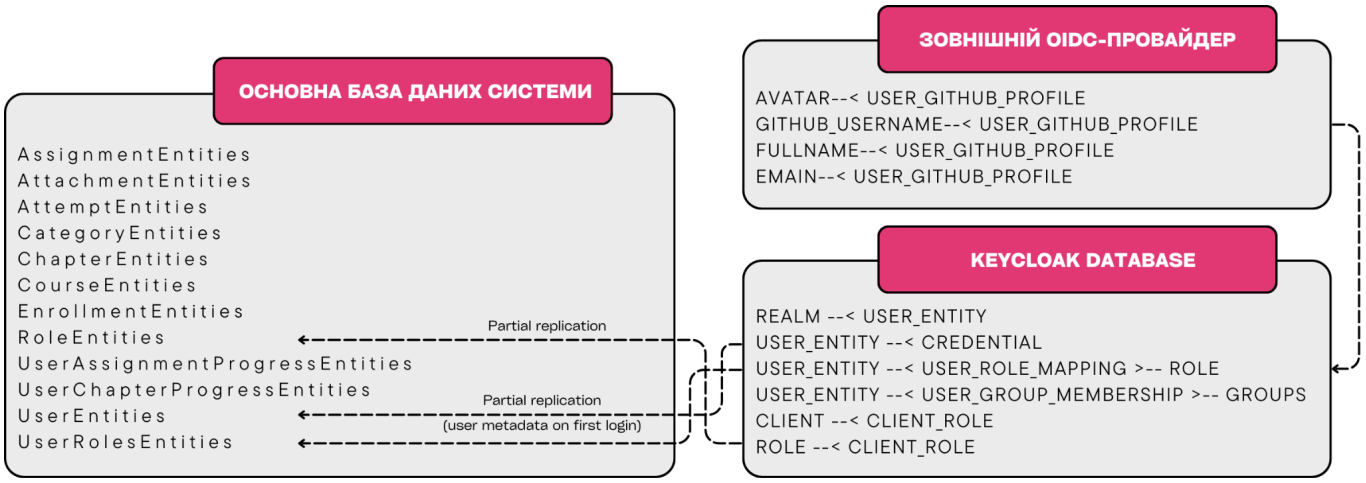

Рис. 3.3. Схема зв'язку між основною базою даних системи та базою даних Keycloak.

Джерело: зроблено автором

Для кешування та проміжного зберігання даних використовується Redis 8.0. Redis підтримує різні структури даних, такі як прості пари ключ-значення, набори, списки, черги та механізми pub/sub. Його впровадження дозволяє реалізувати гібридну модель кешування, ефективно зменшуючи навантаження на основну базу даних. Фрагмент коду, наведений у лістингу 3.2, демонструє, як список завдань розділу курсу кешується за допомогою Redis. Якщо потрібна інформація вже зберігається в кеші, вона буде отримана без необхідності додаткового запиту до бази

даних. В іншому випадку дані будуть отримані через службу доступу до бази даних, кешовані з певними параметрами часу життя, а потім доставлені користувачеві.

Лістинг 3.2. Використання Redis для керування кешем у контролері.

```
private readonly HybridCacheEntryOptions _cacheOptions = new()
{
    LocalCacheExpiration = TimeSpan.FromMinutes(5),
    Expiration = TimeSpan.FromMinutes(10)
};

/// <summary>
///     Getting a list of assignments for a chapter.
/// </summary>
[HttpGet]
public async Task<IActionResult> GetAssignments(Guid courseId, Guid
chapterId,
    CancellationToken cancellationToken = default)
{
    var cacheKey = cacheKeyManager.AssignmentList(courseId, chapterId);
    var cachedAssignments = await cache.GetOrCreateAsync(
        cacheKey,
        async _ => await assignmentService.GetAssignmentsAsync(chapterId,
cancellationToken),
        _cacheOptions,
        [cacheKeyManager.ChapterSingleGroup(courseId, chapterId)],
        cancellationToken);

    return Ok(cachedAssignments);
}
```

Фрагмент коду використовує метод GetOrCreateAsync, який спочатку намагається знайти запитувану інформацію в кеші, використовуючи cacheKey як ключ. Якщо дані відсутні в кеші, метод звернеться до функції GetAssignmentsAsync, щоб отримати їх з бази даних. Згодом отриманий результат буде збережено як у локальному кеші (протягом 5 хвилин), так і в Redis (протягом 10 хвилин) із призначеним часом закінчення терміну дії. Функція групування ключів дозволяє легко видаляти взаємопов'язані дані за потреби.

Процес оцінки коду автоматизовано в ізольованому середовищі контейнерів, що використовує Docker. Спроби студентів обробляються індивідуально в окремих контейнерах, які генеруються, налаштовуються та ініціюються автоматично для цілей тестування, а потім видаляються після оцінки. Цей метод забезпечує чисте середовище виконання, послідовну відтворюваність результатів та запобігає

потенційним перешкодам між різними сеансами тестування. Ізоляція на рівні контейнера підвищує безпеку та полегшує горизонтальне масштабування, дозволяючи паралельну обробку кількох спроб одночасно.

На рисунку 3.4 представлено структуру локального середовища розгортання мікросервісної архітектури платформи. У даному середовищі реалізовано повний набір компонентів, необхідних для забезпечення функціонування системи. Зокрема, включено підсистему автентифікації користувачів, побудовану на основі Keycloak, кешуючий механізм Redis, реляційну базу даних PostgreSQL, систему трасування запитів Jaeger, а також окремі мікросервіси, відповідальні за оцінювання домашніх завдань (course-project-back) та аналіз репозиторіїв (repo-analysis-grpc). Зазначені сервіси забезпечують централізовану координацію процесу автоматизованого оцінювання та здійснення структурного і якісного аналізу студентських рішень відповідно.

Name	Container ID	Image	Port(s)
valerijachyrko-homework-assesment-course-project-back	-	-	-
keycloak	d1bd588011a9	keycloak/keycloak:2	8080:8080
keycloak-db	1ebae253eadf	postgres:15	
homework-assignment-redis	90f35d7bc769	redis:latest	6379:6379
jaeger	6814f9b46e94	jaegertracing/all-in-c	16686:16686 Show all ports (3)
homework-assignment-db	953be27ff6de	postgres:latest	5432:5432
repoanalysis-service-course-project-back	-	-	-
repo-analysis-grpc	995d06d44d28	repo-analysis.grpc:d	

Рис. 3.4. Розгортання контейнеризованих сервісів у локальному середовищі.

Джерело: зроблено автором

Запропонована архітектурна модель і застосовані технологічні рішення формують підґрунтя для побудови надійної, масштабованої та безпечної платформи, призначеної для ефективної автоматизації процесу оцінювання програмного коду в освітньому контексті.

3.2. Особливості розгортання та інфраструктури

Щоб автоматизована система оцінки коду надійно функціонувала в реальних умовах навантаження, необхідно приділити увагу розгортанню та налаштуванню інфраструктури. Це включає налаштування сервісів та баз даних, забезпечення безперебійної координації між ними, захист даних, масштабованість та використання хмарних платформ. Архітектура повинна бути адаптивною до змін у навчальному процесі, водночас зберігаючи відмовостійкість та продуктивність.

Клієнтська частина застосунку розроблено за допомогою Next.js та розгорнуто на Vercel для миттєвого оновлення з гілки GitHub, підтримки рендерингу на стороні сервера та ефективної маршрутизації. Це забезпечує адаптивний інтерфейс платформи та безперебійну масштабованість без додаткових налаштувань. Візуальне представлення компонента ілюстровано на рис. 3.5.

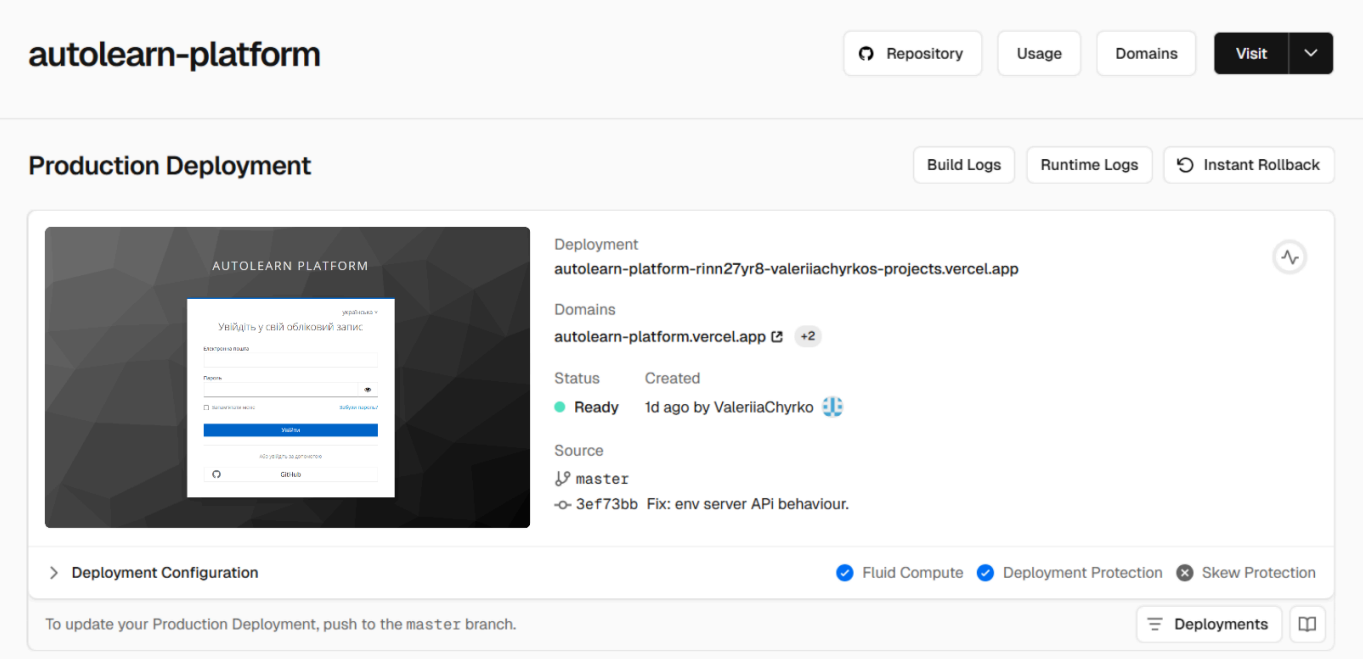

Рис. 3.5. Сторінка керування розгортання платформи на Vercel.

Джерело: зроблено автором

Логіка серверної частини виконується через .NET, розгорнутий на віртуальних машинах Azure. Такий підхід надає нам детальний контроль над середовищем виконання, дозволяє налаштовувати без обмежень інструментами PaaS та закладає основу для ізольованого запуску служб оцінки. Усі запити від Vercel надходять через

HTTPS безпосередньо до вузла, а потім передаються між вузлами сервера за допомогою gRPC. Див. рис. 3.6 для візуального представлення цієї конфігурації.

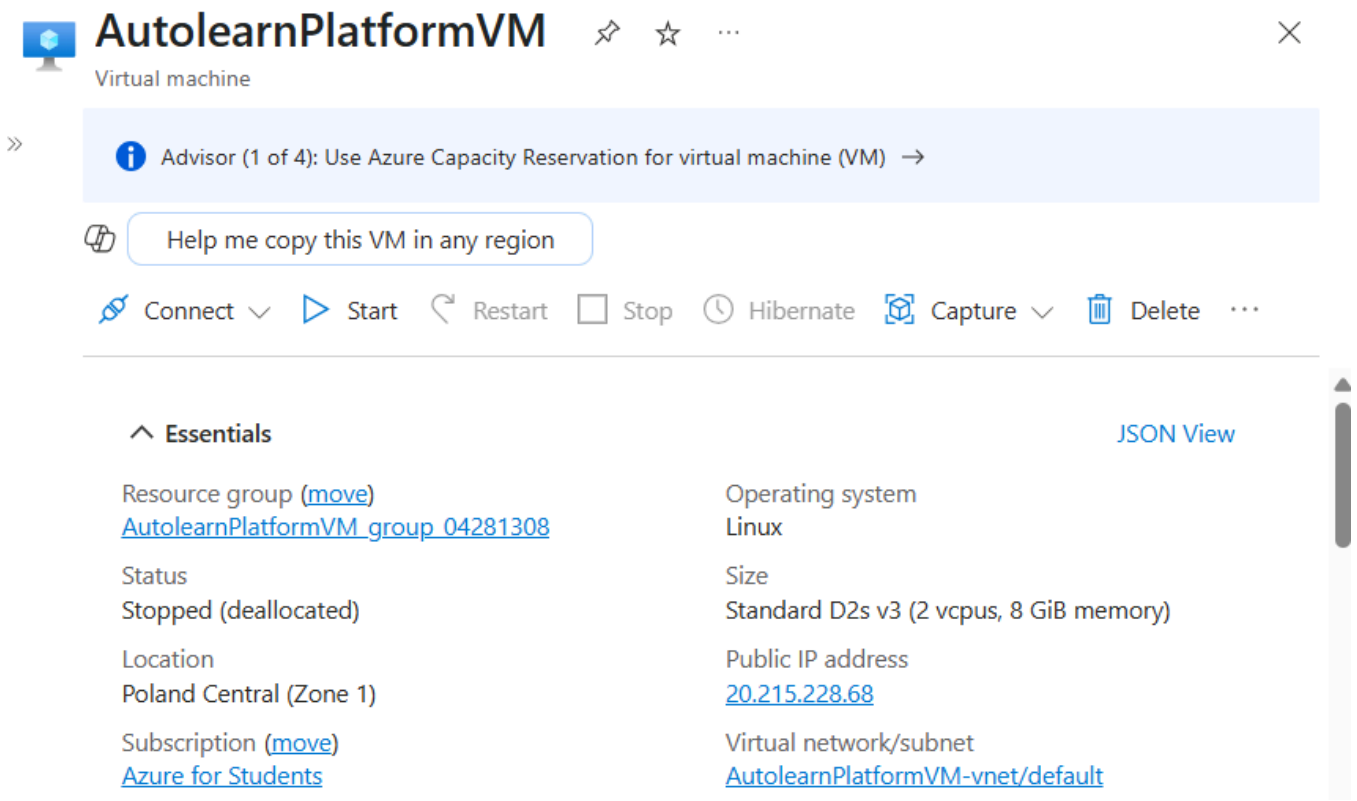

Рис. 3.6. Сторінка керування розгортання платформи на Vercel.

Джерело: зроблено автором

Компонент авторизації реалізовано з використанням системи управління ідентифікацією Keycloak, яка розгорнута як окремий мікросервіс на хмарній платформі Railway. Такий підхід забезпечує ізоляцію функціональності автентифікації та авторизації, що підвищує безпеку та дозволяє масштабувати компонент незалежно від основного застосунку. Сервіс Keycloak використовує реляційну базу даних PostgreSQL для зберігання облікових записів користувачів, ролей, політик доступу та інших конфігурацій безпеки. Обробка запитів автентифікації та авторизації здійснюється відповідно до специфікацій стандарту OpenID Connect, що забезпечує сумісність із широким спектром клієнтських і серверних технологій. У даній системі цей протокол використано для безшовної інтеграції з бекендом, реалізованим на платформі .NET, що суттєво спрощує обробку токенів доступу та реалізацію контролю доступу на рівні API. Структуру та ключові параметри конфігурації авторизаційного сервісу подано на рисунку 3.7.

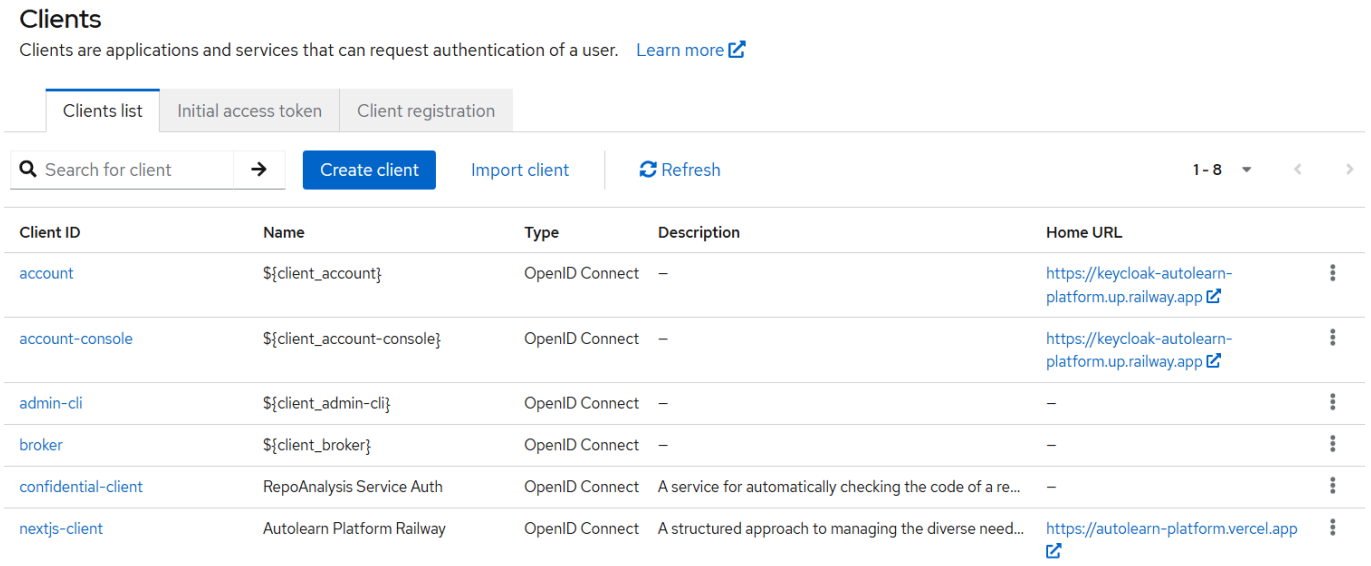

Рис. 3.7. Параметри Keycloak-сервера на Railway.

Джерело: зроблено автором

Основною системою керування базами даних, що використовується в поточній системі реалізації програмного забезпечення, є PostgreSQL. Вона містить усі освітні сутності, такі як курси, завдання, записи спроб та результати оцінювання. Для підвищення продуктивності для кешування часто запитуваних даних та зберігання інформації про сервіси використовується Redis, що ефективно зменшує навантаження на основну СУБД. Обидва сервіси розгорнуті як окремі контейнери в хмарному середовищі Railway. Структурну структуру цих сервісів можна побачити на рисунку 3.8.

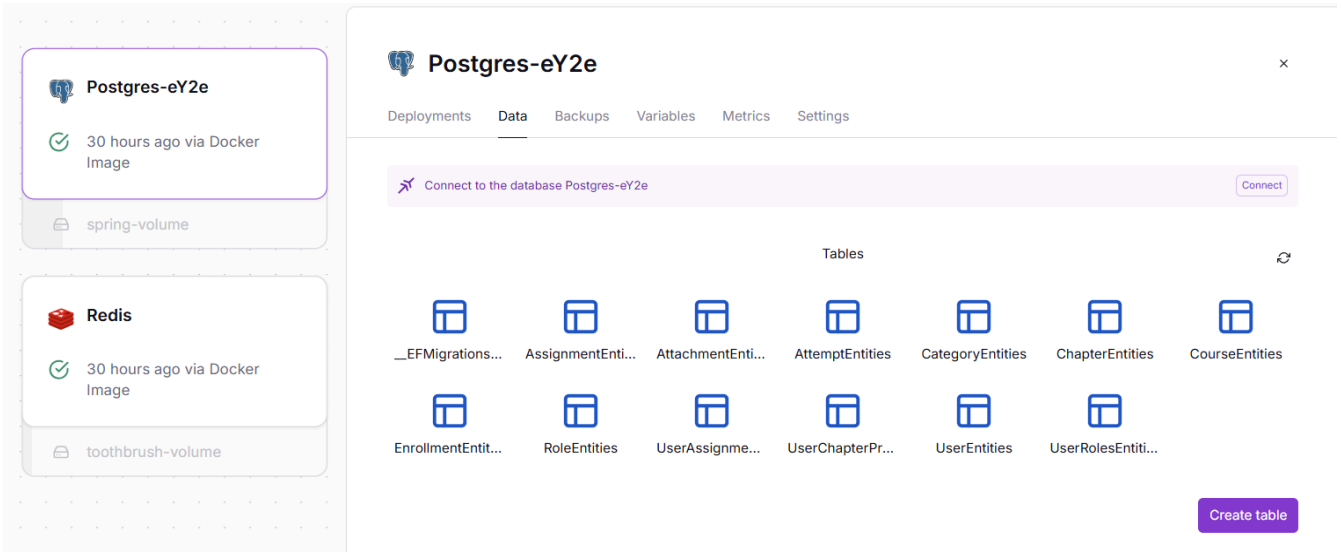

Рис. 3.8. Конфігурація Redis та PostgreSQL на Railway.

Джерело: зроблено автором

Процес взаємодії системи з користувачем відповідає послідовному архітектурному шаблону. Після отримання запиту користувача інтерфейс, побудований на платформі Vercel з фреймворком Next.js, надсилає HTTPS-запит до серверної частини, яка розміщена на віртуальній машині Azure та використовує .NET API. У випадках, коли дані автентифікації відсутні або недійсні, API надсилає запит на авторизацію до Keycloak, розгорнутого в середовищі Railway. Після успішної автентифікації API переходить до запиту даних зі сховища даних. Залежно від типу інформації, доступ надається або до кешу Redis, або безпосередньо до основної реляційної бази даних PostgreSQL, обидва з яких також розміщені на платформі Railway. Оброблені дані повертаються у відповіді, яка потім надсилається на клієнтський застосунок для візуалізації користувачеві. Це завершує повний цикл обробки запиту, що охоплює автентифікацію, доступ до даних та доставку відповіді. На рисунку 3.9 показано узагальнену діаграму послідовності запитів та взаємодії компонентів.

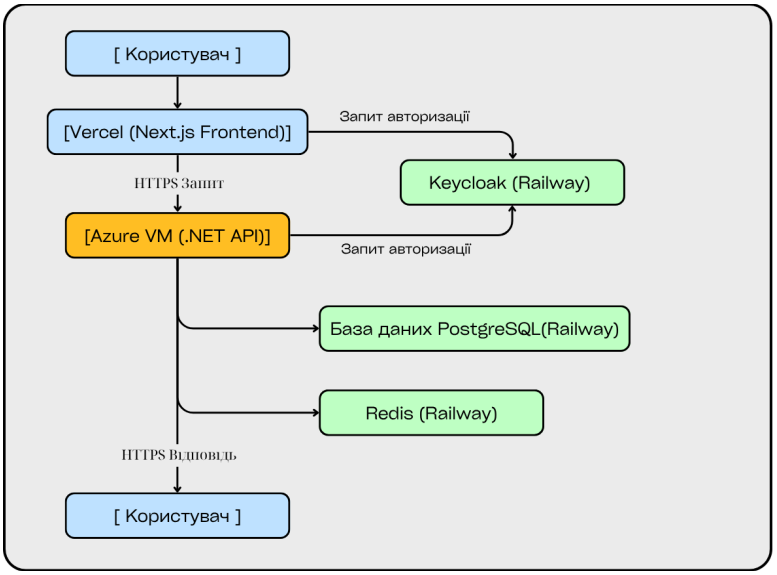

Рис. 3.9. Схема послідовності взаємодії користувача з компонентами системи.

Джерело: зроблено автором

Архітектуру розгортання платформи розроблено з урахуванням ключових аспектів гнучкості, масштабованості та безпеки. Використання можливостей Vercel, Azure та Railway дозволяє ізолювати компоненти системи, забезпечити автоматизоване CI/CD-розгортання, а також ефективно обробляти запити без втрати продуктивності.

3.3. Програмна реалізація основних компонентів системи

Цей розділ зосереджений на технічній реалізації ключових системних модулів, з акцентом на клієнтській частині, механізмах автентифікації, взаємодії API, управлінні станом інтерфейсу та підсистемі аналізу якості програмного коду. Кожен компонент розглядається з погляду логіки реалізації, прикладів коду та візуалізації інтерфейсу.

Система використовує централізовану автентифікацію користувачів за допомогою сервісу Keycloak, що забезпечує єдиний процес входу на всіх модулях платформи та надійний захист даних. Після натискання кнопки входу користувачі перенаправляються на зовнішній сервер Keycloak для автентифікації, а потім повертаються до програми з токеном JWT. Для спрощення авторизації програма використовує бібліотеку next-auth, що забезпечує безперешкодну інтеграцію постачальника Keycloak у програму Next.js. Наведений фрагмент коду в лістингу 3.3 демонструє ініціалізацію процесу входу через Keycloak.

Лістинг коду 3.3 Ініціалізація входу через Keycloak.

```
const signInUser  = async () => {
   const result = await signIn("keycloak", { redirect: false });
   if (result?.error) {
       console.error("Sign-in error:", result.error);
   } else {
       router.push('/');
   }
};
```

Система пропонує можливість вибору однієї з трьох мов інтерфейсу: української, англійської та польської під час етапу входу, що підвищує зручність для користувачів. Крім того, підтримується відновлення пароля через Gmail, що дозволяє користувачам відновити доступ до свого облікового запису, просто вказавши свою адресу електронної пошти для отримання посилання для відновлення пароля. Вхід через GitHub також є опцією, що дозволяє користувачам зручно отримувати доступ до платформи без створення нового облікового запису. Приклад

форми входу користувача, яка дозволяє вибір між різними методами автентифікації, показано на рисунку 3.10.

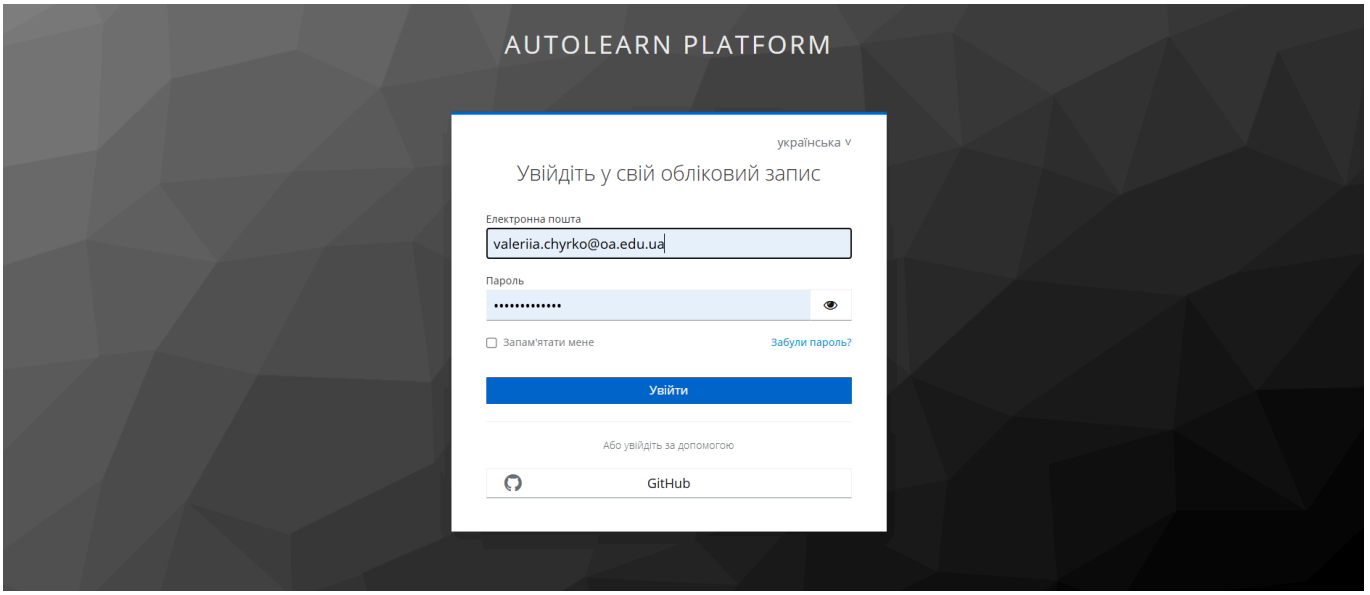

Рис. 3.10. Інтерфейс користувача сторінки авторизації.

Джерело: зроблено автором

Бібліотека React Query використовується для оптимізації взаємодії між клієнтською програмою та серверним API. Ця бібліотека включає різні асинхронні шаблони керування запитами, такі як кешування, анулювання даних, повторні спроби запитів у разі помилок та автоматичне оновлення стану інтерфейсу на основі змін у джерелі даних. Використовуючи ці функції, можна зменшити навантаження на сервер, підвищити швидкість рендерингу компонентів та підвищити стійкість до мережевих проблем. В основі його функціональності лежить декларативне визначення API-запитів, результати якого автоматично зберігаються у внутрішньому кеші та повторно використовуються для майбутніх запитів з подібними ідентифікаторами. Такий підхід забезпечує послідовне та масштабоване управління даними всередині програми.

Лістинг коду 3.4 Приклад отримання даних через React Query.

```
const { data, isLoading, isError } = useQuery<CoursesProgressResponse,
Error>({
  queryKey: ["coursesProgress"],
  queryFn: fetchCoursesWithProgress,
});
```

У наведеному лістингу функція useQuery використовується для ініціювання запиту до API для отримання детальної інформації про прогрес користувача в курсі. Відповідь від сервера зберігається у змінній data, тоді як isLoading та isError вказують на статус запиту. React Query дозволяє автоматично оновлювати ці значення без необхідності ручного керування станом або складних реалізацій кешування.

Інформація, отримана з API, автоматично передається відповідним компонентам візуального інтерфейсу. Наприклад, на сторінці пошуку курсів, яка розроблена як окреме представлення, деталі про доступні курси динамічно відображаються у вигляді карток. У разі затримки завантаження або помилки компоненти плавно переходять до скелета заповнювача або сторінки повідомлення про помилку. Це покращення покращує взаємодію з користувачем та підвищує стійкість інтерфейсу до мережевих проблем. Інтерфейс сторінки пошуку курсу, як показано на рисунку 3.11, дозволяє користувачам переглядати добірку доступних навчальних програм. Кожна програма представлена своєю назвою, коротким описом та індикатором прогресу.

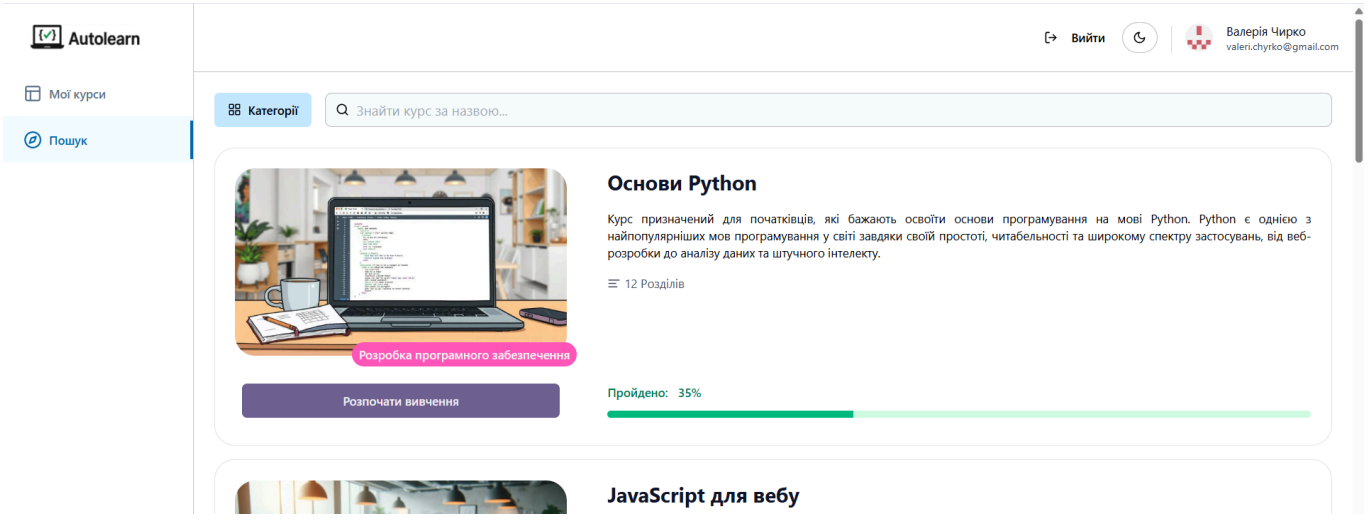

Рис. 3.11. Інтерфейс користувача для сторінки пошуку курсу.

Джерело: зроблено автором

Платформа використовує механізм, який дозволяє студентам надсилати свої програмні рішення для автоматичної перевірки безпосередньо через вебінтерфейс. Кожне завдання пов'язане з відповідним репозиторієм GitHub, з якого система

отримала код під час початкового надсилання. Перед повторним надсиланням користувач вибирає гілку, з якої слід отримати оновлене рішення, що запускає процес аналізу якості коду. Технічний аспект включає реалізацію асинхронного методу під назвою VerifyProjectCompilation. Цей метод приймає параметри репозиторію та гілки, автентифікується через Keycloak, надсилає gRPC-запит до відповідного сервісу компіляції та аналізує відповідь для оцінки.

Лістинг коду 3.5 Метод перевірки компіляції проєкту.

```
public async Task<int>
VerifyProjectCompilation(RequestRepositoryWithBranchDto query,
CancellationToken ct = default)
{
    var token = await _keycloakTokenService.GetAccessTokenAsync();
    if (string.IsNullOrEmpty(token))
        throw new RpcException(new Status(StatusCode.Unauthenticated, "No
access token"));

    var metadata = new Metadata { { "Authorization", $"Bearer {token}" } };
    var request = new RepositoryWithBranchQuery { ... }; // параметри
репозиторію

    var response = await _client.VerifyProjectCompilationAsync(request,
metadata, cancellationToken: ct);
    return response.Score;
}
```

У наведеному лістингу показано ключові етапи виклику служби компіляції. Вищезгадана функціональність інтегрована в інтерфейс користувача за допомогою кнопки «Надіслати рішення». Після натискання якої відображається форма, що дозволяє користувачам вибрати власну гілку репозиторію GitHub, пов’язану з завданням. Після підтвердження надсилання відображається статус перевірки.

Рис. 3.12. Інтерфейс користувача для аналітики спроби подання рішення.
Джерело: зроблено автором

У контексті автоматизованої перевірки рішень, проведення тестів у контрольованому середовищі є критично важливим. Для досягнення цієї мети використовується інструментарій Docker, який ізолює процес тестування від основного середовища платформи. Ця методологія гарантує відтворюваність результатів, відсутність потреб у конфігураціях хост-системи та безпеку під час запуску неперевіреного коду. Ізольоване тестування досягається шляхом запуску контейнерів у відповідному середовищі виконання. В лістингу 3.6 наведено ілюстрацію запуску базової команди.

Лістинг коду 3.6 Метод запуску команди у контейнері.

```
public async Task<ProcessResult> RunCommandAsync(DockerCommandOptions
options, CancellationToken cancellationToken)
{
       var dockerCommand =
       $"docker run --rm -v \"{options.RepositoryPath}:/workspace\" -w
/workspace/{options.WorkingDirectory} {options.DockerImage}
{options.Command}{(string.IsNullOrWhiteSpace(options.Arguments) ?
string.Empty : $" {options.Arguments}")}";

   var isWindows = Environment.OSVersion.Platform == PlatformID.Win32NT;
   var shellFileName = isWindows ? "cmd.exe" : "sh";
   var shellArguments = isWindows ? $"/c \"{dockerCommand.Replace("\\",
"/")}\"" : $"-c \"{dockerCommand}\"";

   var processStartInfo = new ProcessStartInfo
   { … }; //// параметри процесу

   var result = await _processService.RunProcessAsync(processStartInfo,
cancellationToken);

   return result;
}
```

Використані інструменти та технології розроблені для забезпечення стабільності, безпеки та масштабованості, що є вирішальними для підтримки якості автоматизованої оцінки коду на освітній платформі. Впровадження цієї системи сприяє створенню надійного середовища для розвитку навичок програмування студентів та забезпеченню прозорості в оцінюванні їхньої роботи.

Висновки до розділу 3

У наступному розділі розглядаються практичні аспекти розробки автоматизованої системи оцінки коду Autolearn. Архітектура системи базується на мікросервісному підході з використанням сучасних вебтехнологій та хмарних сервісів. Клієнтська частина реалізована за допомогою фреймворку Next.js, що пропонує зручну навігацію, динамічне оновлення даних через React Query та підтримку автентифікації через Keycloak.

Серверну частину застосунку побудовано на .NET 9.0, використовуючи сервіси gRPC для зв'язку між компонентами. Зберігання та кешування даних обробляються за допомогою PostgreSQL та Redis відповідно, що забезпечує оптимізовану продуктивність під час обробки великої кількості запитів. Система має механізм фонової обробки спроб студентів, включаючи клонування репозиторію, компіляцію та виконання модульних тестів у контейнерах Docker. Це забезпечує контрольоване та відтворюване середовище оцінювання з посиленими заходами безпеки.

Хмарна інфраструктура структурована з використанням гібридного підходу: клієнтська частина розміщена на Vercel, серверна логіка знаходиться у віртуальній машині Azure, а пов'язані служби (Redis, PostgreSQL, Keycloak) розташовані в Railway. Така конфігурація забезпечує адаптивне масштабування та ефективне управління ресурсами, миттєво адаптуючись до коливань робочого навантаження.

Підсумовуючи, платформа Autolearn демонструє, як сучасні методи можуть бути ефективно використані для розробки масштабованих, безпечних та гнучких систем для IT-освіти. Завдяки використанню хмарної інфраструктури, мікросервісної архітектури, автоматизованого тестування коду та інтерактивного вебінтерфейсу, ця платформа забезпечує автоматизоване оцінювання в освітній сфері з високою ефективністю.

ВИСНОВКИ

У межах кваліфікаційного проєкту було здійснено поглиблене дослідження проблематики автоматизованого оцінювання програмного коду студентів із використанням модульних (юніт) тестів. Актуальність обраної тематики зумовлена зростаючою потребою у підвищенні об’єктивності, ефективності та масштабованості освітнього процесу в закладах вищої освіти, зокрема в умовах дистанційного навчання та цифрової трансформації освітніх послуг. Автоматизація процесу перевірки коду дозволяє не лише зменшити навантаження на викладачів, а й забезпечити прозорість, відтворюваність і незалежність оцінювання, що є критично важливими факторами в сучасній педагогічній практиці.

Після детального аналізу предметної галузі, а також функціональних вимог до систем навчального менеджменту, було сформульовано архітектурні засади побудови системи, визначено її функціональні та нефункціональні характеристики. Зокрема, було акцентовано на потребі у модульності, масштабованості, безпечності, стійкості до збоїв та здатності до інтеграції з іншими компонентами цифрового освітнього середовища. Усі ці аспекти стали основою для побудови технічного рішення, яке здатне гнучко адаптуватися до змінних умов та потреб освітнього процесу.

Розроблена система автоматизованого оцінювання базується на мікросервісній архітектурі, що дозволяє виділяти окремі функціональні компоненти в автономні сервіси. До складу системи входять сервіси для управління обліковими записами, компіляції програмного коду, виконання юніт тестів, аналізу якості коду та збирання аналітичних даних. Комунікація між цими сервісами реалізована за допомогою високопродуктивного протоколу gRPC, що забезпечує ефективну взаємодію з низькими затримками. Окрему увагу приділено питанням інформаційної безпеки: механізми централізованої автентифікації та авторизації реалізовані на основі Keycloak із використанням токенів формату JWT, що дозволяє гарантувати захищений доступ до сервісів системи.

Підсистема оцінювання створена з урахуванням принципів ізоляції та верифікованості. Користувацький код виконується в ізольованому

контейнеризованому середовищі Docker, що унеможливлює втручання в інфраструктуру або маніпуляції з результатами. Оцінювання реалізовано за допомогою алгоритмів, які визначають результат на основі кількості успішно пройдених юніт тестів. При цьому генеруються розгорнуті результати, що включають загальну оцінку, а також бали за окремими секціями завдання. Усі зібрані дані зберігаються у базі даних PostgreSQL, що відкриває можливості для статистичного аналізу та покращення якості освітніх процесів.

Система використовує гібридну модель розгортання, яка поєднує переваги хмарних і локальних рішень для досягнення високої продуктивності. Зокрема, клієнтська частина, реалізована з використанням фреймворку Next.js, розміщена на платформі Vercel, що забезпечує швидке масштабування та доступність. У той час як серверні компоненти функціонують на віртуальній машині Microsoft Azure. Допоміжні інфраструктурні сервіси, зокрема PostgreSQL, Redis та Keycloak, розгорнуті за допомогою Railway, що дозволяє централізовано управляти конфігураціями та оновленнями. Для зменшення навантаження на систему впроваджено кешування даних за допомогою Redis із підтримкою механізмів групування ключів та періодичного очищення кешу через фонові задачі, реалізовані в середовищі .NET.

Досягнута мета створення надійної, стабільної та адаптивної системи оцінювання коду підтверджується результатами експериментального впровадження та тестування. Розроблене рішення може бути використане як у навчальних цілях, так і як фундамент для розширення у повноцінну платформу управління навчанням, з можливістю інтеграції функцій динамічного налаштування завдань, аналізу освітніх результатів та підключення до зовнішніх хмарних сервісів. Таким чином, проєкт сприяє модернізації інструментарію цифрової освіти та відкриває перспективи для подальших досліджень і розробок у цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASTQB. (n.d.). Glossary of Software Testing Terms. URL: https://astqb.org/resources/glossary-of-software-testing-terms/ (Дата звернення: 29.10.2024).

2. Battat M. What is Visual Testing? Applitools Blog, Aug 7, 2024. URL: https://applitools.com/blog/visualtesting/ (дата звернення: 11.02.2025).

3. Gennarelli V. How to grade programming assignments on GitHub. GitHub Blog, June 13, 2017. URL: https://github.blog/2017-06-13-how-to-grade-programming-assignments-on-github/ (дата звернення: 12.02.2025).

4. Messer M., Brown N.C.C., Kölling M., Shi M. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. Computer Science Education, 2024 (ArXiv preprint). DOI: 10.48550/arXiv.2306.11722.

5. Pankiewicz M., Baker R., Ocumpaugh J. Using intelligent tutoring on the first steps of learning to program: affective and learning outcomes. Proceedings of the International Conference on Educational Data Mining, 2021.

6. Sigma Software University. Ручне і автоматизоване тестування: різниця, переваги і недоліки. 2023. URL: https://university.sigma.software/manual-testing-vs-automation-testing (дата звернення: 11.02.2025).

7. Srinivasan B., Nehru M., Parthasarathi R., Mukherjee S., Thankachan J.A. Automated Computer Program Evaluation and Projects – Our Experiences. 2024. (ArXiv:2404.04521).

8. Вапнічний С., Беца А. Розробка інформаційної онлайн-системи для вивчення початків програмування. Вісник Чернівецького національного університету (серія «Педагогічні науки»), №18(174), 2022, с. 7–12.

9. Волошина Т.В., Глазунова О.Г., Гуржій А.М., Пархоменко О.В., Кухаревич О.В. Платформи та системи автоматизованої перевірки завдань з програмування: аналіз, критерії добору та приклад використання. Відкрите освітнє е-середовище сучасного університету, 2020, Вип. 8, с. 1–9.

10. Ala-Mutka K. Programming exercises evaluation systems – an interoperability survey // Proceedings of the 4th International Conference on Computer Supported Education (CSEDU 2012). 2012.

11. Edwards S.H., Pérez-Quiñones M.A. Web-CAT: a tool for automatically assessing student programs // Proceedings of the 3rd International Conference on Education and Information Systems, Technologies and Applications (EISTA 2003). 2003.

12. GitHub Education. Use autograding – GitHub Classroom documentation [Електронний ресурс]. — 2024. — Режим доступу: https://docs.github.com/en/education/manage-coursework-with-github-classroom/use-autograding (Дата звернення: 19.02.2025).

13. Leal J.P., Silva F.M.A. Mooshak: a Web-based multi-site programming contest system // Software–Practice & Experience. 2003. Vol. 33, №6. С. 567–581.

14. Міністерство освіти і науки України. Автоматичне оцінювання для вчителів, дедлайни завдань для учнів та відстеження успішності для батьків — оновили платформу ІТ-студій [Електронний ресурс]. — 2024. — Режим доступу: https://mon.gov.ua/news/it-studii-onovlennya (Дата звернення: 20.02.2025).

15. Orvalho P., Janota M., Manquinho V. GitSEED: a Git-backed automated assessment tool for software engineering and programming education // arXiv:2409.07362 [Електронний ресурс]. — 2024. — Режим доступу: https://arxiv.org/abs/2409.07362 (Дата звернення: 20.02.2025).

16. Сумський обласний інститут післядипломної педагогічної освіти. JustClass – безкоштовна платформа для автоматичної перевірки домашніх завдань [Електронний ресурс]. — 2022. — Режим доступу: https://bit.ly/justclass-sumy (Дата звернення: 21.02.2025).

17. Волошина Т.В., Глазунова О.Г., Шестопалова Л.О., Литвинова С.Г. Платформи та системи автоматизованої перевірки завдань з програмування: аналіз, критерії добору та приклад використання // Open educational environment of modern University. 2020. №8. С. 154–163.

18. Altkom Software. Keycloak for Identity and Access Management [Електронний ресурс]. — Режим доступу: https://altkomsoftware.com (Дата звернення: 11.03.2025).

19. Auth0. gRPC Explained [Електронний ресурс]. — Режим доступу: https://auth0.com (Дата звернення: 11.03.2025).

20. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. — Addison-Wesley, 2021. — 432 с.

21. Fowler M. Microservices Resource Guide [Електронний ресурс]. — Режим доступу: https://martinfowler.com (Дата звернення: 11.03.2025).

22. Microsoft Azure Architecture Center. Microservices Design Guide [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com (Дата звернення: 11.03.2025).

23. Momjian B. PostgreSQL: Scaling and Performance [Електронний ресурс]. — Режим доступу: https://momjian.us (Дата звернення: 12.03.2025).

24. Newman S. Building Microservices. — O'Reilly, 2021. — 432 с.

25. Payara Blog. Advantages of Microservices [Електронний ресурс]. — Режим доступу: https://blog.payara.fish (Дата звернення: 12.03.2025).

26. Red Hat. Keycloak Documentation [Електронний ресурс]. — Режим доступу: https://www.redhat.com (Дата звернення: 12.03.2025).

27. Redis Labs. High Performance Redis Clusters [Електронний ресурс]. — Режим доступу: https://redis.io (Дата звернення: 12.03.2025).

28. Richards M. Microservices vs Service-Oriented Architecture. — O'Reilly, 2020. — 198 с.

29. Taibi D., Lenarduzzi V. Microservices Scalability Patterns // IEEE Software. — 2021. — №38(3). — С. 50–57.

30. Alégroth E., Feldt R., Kolström P. Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing // Information and Software Technology. 2015. Vol. 57. P. 248–264. DOI: https://doi.org/10.1016/j.infsof.2014.05.011.

31. Дмитрієва Н. О., Чирва О. О. Методика оцінювання якості програмного забезпечення в освітньому середовищі // Вісник Харківського національного

університету імені В. Н. Каразіна. Серія: Математика, прикладна математика та механіка. 2021. №94. С. 61–69. DOI: 10.26565/2221-5646-2021-94-06.

32. ESLint. Find and fix problems in your JavaScript code [Електронний ресурс]. — Режим доступу: https://eslint.org/ (Дата звернення: 15.03.2025).

33. IEEE 829-2008 — IEEE Standard for Software and System Test Documentation. Нью-Йорк: IEEE, 2008. 83 с.

34. ISO/IEC 25010:2011 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Женева: ISO, 2011. 34 с.

35. ISO/IEC/IEEE 29119-4:2015. Програмна та системна інженерія. Тестування програмного забезпечення. Частина 4. Техніки тестування (Software and systems engineering – Software testing – Part 4: Test techniques). Женева: ISO, 2015. 62 с.

36. Pylint. Code analysis for Python [Електронний ресурс]. — Режим доступу: https://pylint.pycqa.org/ (Дата звернення: 15.03.2025).

37. SonarQube. Continuous Code Quality [Електронний ресурс]. — Режим доступу: https://www.sonarsource.com/products/sonarqube/ (Дата звернення: 15.03.2025).

38. ISO/IEC/IEEE 29119-4:2015 — Software and systems engineering — Software testing — Part 4: Test techniques. Женева: ISO, 2015. 62 с.

39. Microsoft Corporation. .NET Microservices: Architecture for Containerized .NET Applications [Електронний ресурс]. — 2022. — Режим доступу: https://dotnet.microsoft.com/en-us/learn/aspnet/microservices-architecture (дата звернення: 20.03.2025).

40. Payara Services Ltd. Microservices for the Enterprise with Payara Platform [Електронний ресурс]. — 2025. — Режим доступу: https://www.payara.fish (дата звернення: 20.03.2025).

ДОДАТОК А

Таблиця А.1

Порівняльна характеристика інструментів для автоматизованого оцінювання коду

Система	**Підтримка мов програмування**	**Інтеграція з LMS**	**Підтримка CI/CD**	**Тип зворотного зв'язку**	**Вартість**	**Можливість використання в Україні**	**Складність впровадження**
Web-CAT	Java, C++, Scheme, Prolog, Standard ML, Pascal (та інші)	LTI (SSO, передача оцінок)	Ні	Результати тестів; покриття; аналіз стилю коду (Checkstyle, PMD); ручна оцінка	Безплатно, відкритий код	Так (відкритий код)	Висока
Mooshak	C, C++, Pascal, Java (за замовчуванням)	Ні	Ні	Результати запуску (accepted/rejected)	Безплатно (відкрита, Artistic License 2.0)	Так (open-source)	Середня
UVa Online Judge	C (C89), C++ (C++98, C++11), Pascal, Java, Python	Ні	Ні	Результати запуску (Accepted/Rejected)	Безплатно	Так (онлайн-сервіс)	Низька (онлайн-сервіс)

GitSEED	Будь-яка (мовно незалежна)	Ні (лише GitLab)	Так (GitLab CI)	Результати тестів; розширений аналіз (витоки пам’яті, локалізація помилок, покриття, аналіз схожості)	Безплатно , відкритий код	Так (відкритий)	Висока
GitHub Classroom	Будь-яка (через GitHub Actions)	Так (LTI для Canvas, Moodle тощо)	Так (GitHub Actions)	Автотестування (green check/red X); коментарі викладача	Безплатно	Так	Низька
GradeScope	Будь-яка (через Docker)	Так (Canvas, Moodle, Blackboard тощо; потребує ліцензії)	Ні	Автотестування; Code Similarity (аналіз схожості, *не* повноцінний антиплагіат система)	Платна (інституційна ліцензія)	Так	Середня
JustClass	Немає (зосереджено на тестах, не на коді)	Ні	Ні	Автооцінка інтерактивних тестів/завдань	Безплатно	Так (українська платформа)	Низька

Джерело: зроблено автором

ДОДАТОК Б

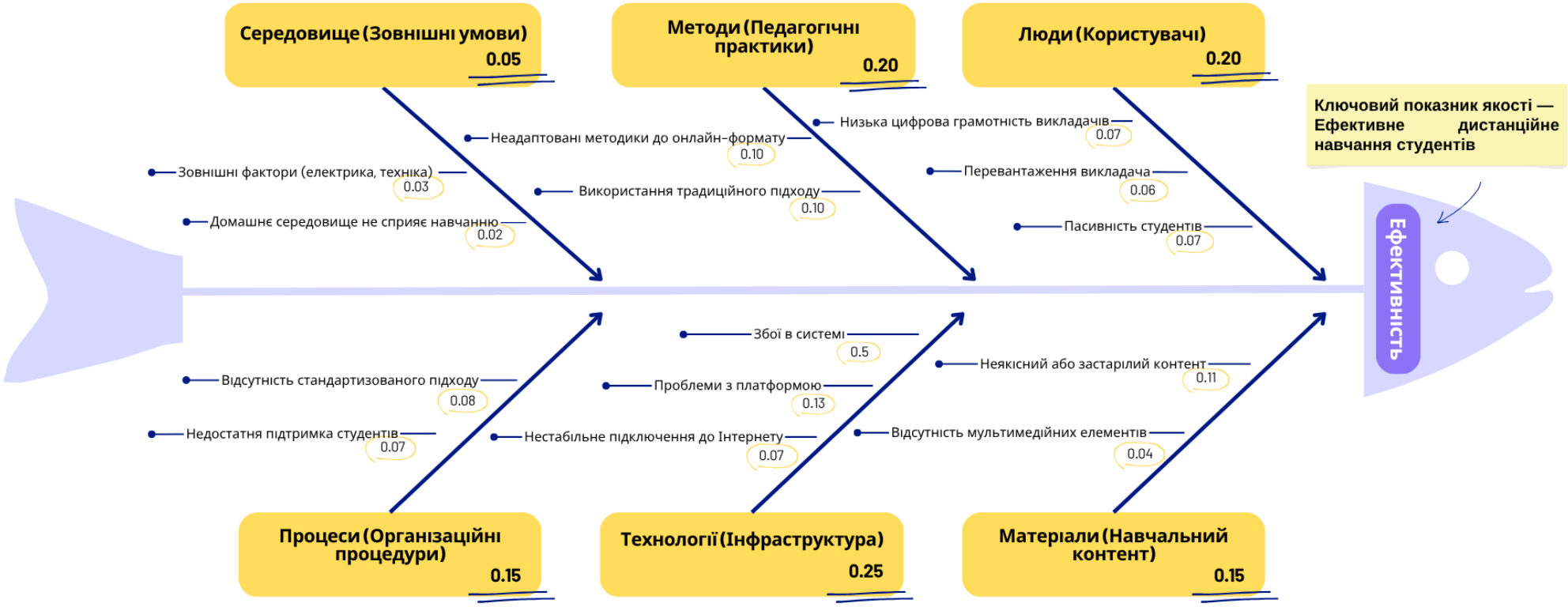

Рис. Б.1. Діаграма Ісікави — фактори впливу на ефективність системи.

Джерело: зроблено автором

ДОДАТОК В

Таблиця В.1

Вимоги користувачів до функціональності платформи та їх вагова значущість

№	Вимога користувача	Важливість
1	Інтуїтивний інтерфейс	5
2	Доступ 24/7	4
3	Надійне тестування	5
4	Редагування курсів	4
5	Аналітика для адмінів	3
6	Безпека даних	5
7	Мобільна адаптивність	4

Джерело: зроблено автором

Таблиця В.2

Матриця залежностей між користувацькими вимогами та технічними характеристиками системи

Вимоги/ Характеристики	Кл.-сервер	Auth	Адаптивність	Тести	Аналітика	Хмара	Безпека
Інтерфейс	3	1	9	0	0	0	1
Доступ 24/7	9	1	3	0	0	3	3
Тестування	0	3	0	9	0	1	3
Курси	3	3	3	0	1	0	0
Аналітика	1	0	0	3	9	1	0
Безпека	1	9	0	0	1	3	9
Адаптивність	0	0	9	0	0	0	0

Джерело: зроблено автором

Таблиця В.3

Взаємозв’язки між технічними характеристиками у моделі «Будинку якості»

№	Характеристика 1	Характеристика 2	Зв’язок	Пояснення
1	Архітектура клієнт-сервер	Модулі аутентифікації /авторизації	+	Легше вбудовувати системи контролю доступу в централізованій архітектурі

продовження Табл. В.3

2	Архітектура клієнт-сервер	Фреймворки з адаптивною верстkою	0	Архітектура і візуальна адаптивність не мають прямого зв’язку
3	Архітектура клієнт-сервер	Інтеграція з системою тестування	+	Клієнт-сервер дозволяє централізовано реалізувати запуск і зберігання тестів
4	Архітектура клієнт-сервер	Аналітичний модуль	+	Централізоване збирання даних полегшує реалізацію аналітики
5	Архітектура клієнт-сервер	Хмарне зберігання даних	+	Розподілене зберігання і обробка даних краще реалізуються в клієнт-серверній архітектурі
6	Архітектура клієнт-сервер	Резервне копіювання та шифрування	+	Клієнт-сервер дає можливість централізовано керувати безпекою і резервуванням
7	Модулі аутентифікації/ авторизації	Фреймворки з адаптивною верстkою	-	Додаткові заходи безпеки можуть ускладнити адаптивність (наприклад, CAPTCHA)
8	Модулі аутентифікації/ авторизації	Інтеграція з системою тестування	0	Аутентифікація і тестування працюють окремо, але можуть взаємодіяти
9	Модулі аутентифікації /авторизації	Аналітичний модуль	+	Логування входів/виходів може бути джерелом для аналітики
10	Модулі аутентифікації/ авторизації	Хмарне зберігання даних	+	Зберігання даних облікових записів часто реалізується у хмарі
11	Модулі аутентифікації/ авторизації	Резервне копіювання та шифрування	+	Безпечна аутентифікація підсилює потребу в надійному шифруванні та збереженні даних
12	Фреймворки з адаптивною верстkою	Інтеграція з системою тестування	0	Вони не залежать одне від одного функціонально
13	Фреймворки з адаптивною верстkою	Аналітичний модуль	0	Адаптивність UI не впливає на функціональність аналітики
14	Фреймворки з адаптивною верстkою	Хмарне зберігання даних	0	Збереження даних не впливає на адаптивність інтерфейсу
15	Фреймворки з адаптивною верстkою	Резервне копіювання та шифрування	0	Безпека і дизайн інтерфейсу функціонують незалежно

16	Інтеграція з системою тестування	Аналітичний модуль	+	Тестові результати можуть бути використані для аналітики
17	Інтеграція з системою тестування	Хмарне зберігання даних	+	Результати тестів можуть зберігатися в хмарі
18	Інтеграція з системою тестування	Резервне копіювання та шифрування	+	Дані тестування мають зберігатися без втрат і захищено
19	Аналітичний модуль	Хмарне зберігання даних	+	Аналітичні дані потребують централізованого надійного збереження
20	Аналітичний модуль	Резервне копіювання та шифрування	+	Аналітична інформація часто є чутливою, тому її потрібно шифрувати
21	Хмарне зберігання даних	Резервне копіювання та шифрування	+	Хмарне середовище вимагає додаткових заходів безпеки та резервування

Джерело: зроблено автором

Умовні позначення

● ● Сильний зв'язок = 9

● Середній = 3

◉ Слабкий = 1

○ Немає зв'язку = 0

\+ якщо дві характеристики підсилюють одна одну

0 якщо вони конфліктують

\- відсутній зв'язок або несуттєвий

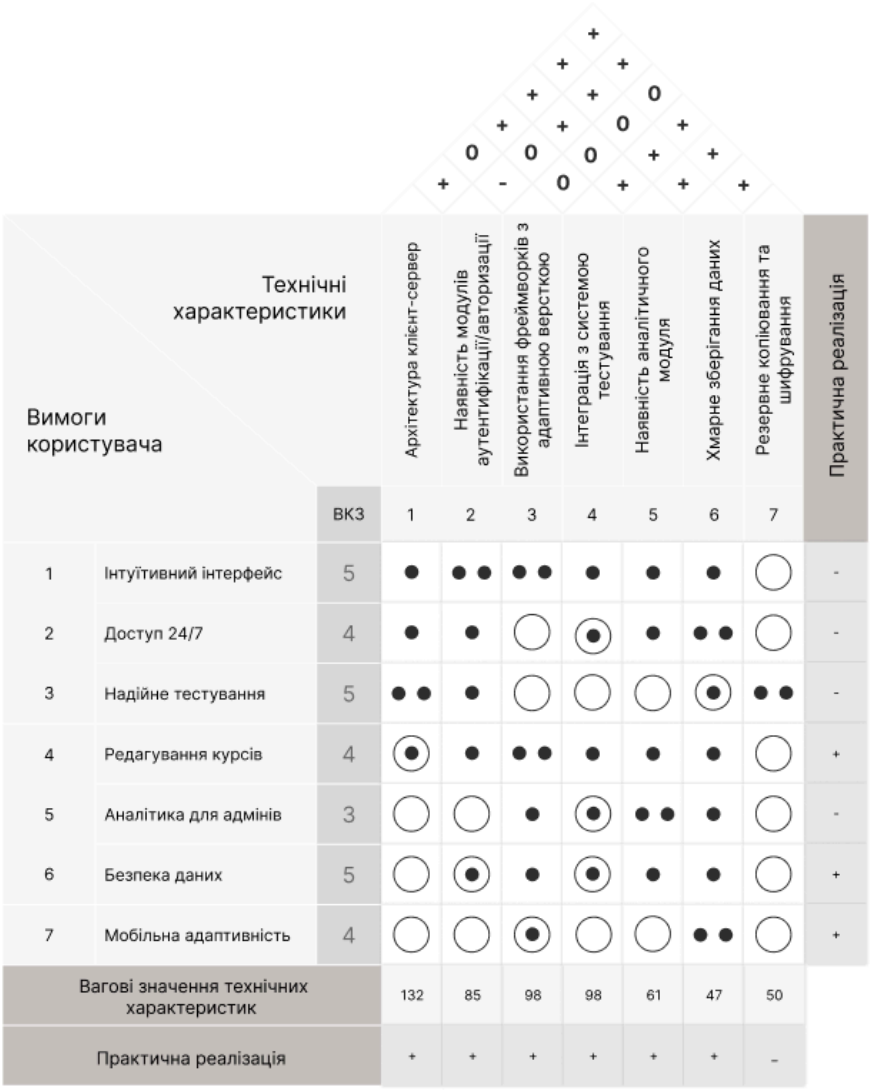

Рис. Б.1 Графічне представлення QFD-моделі відповідності вимог користувачів технічним характеристикам

Джерело: зроблено автором

ДОДАТОК Г

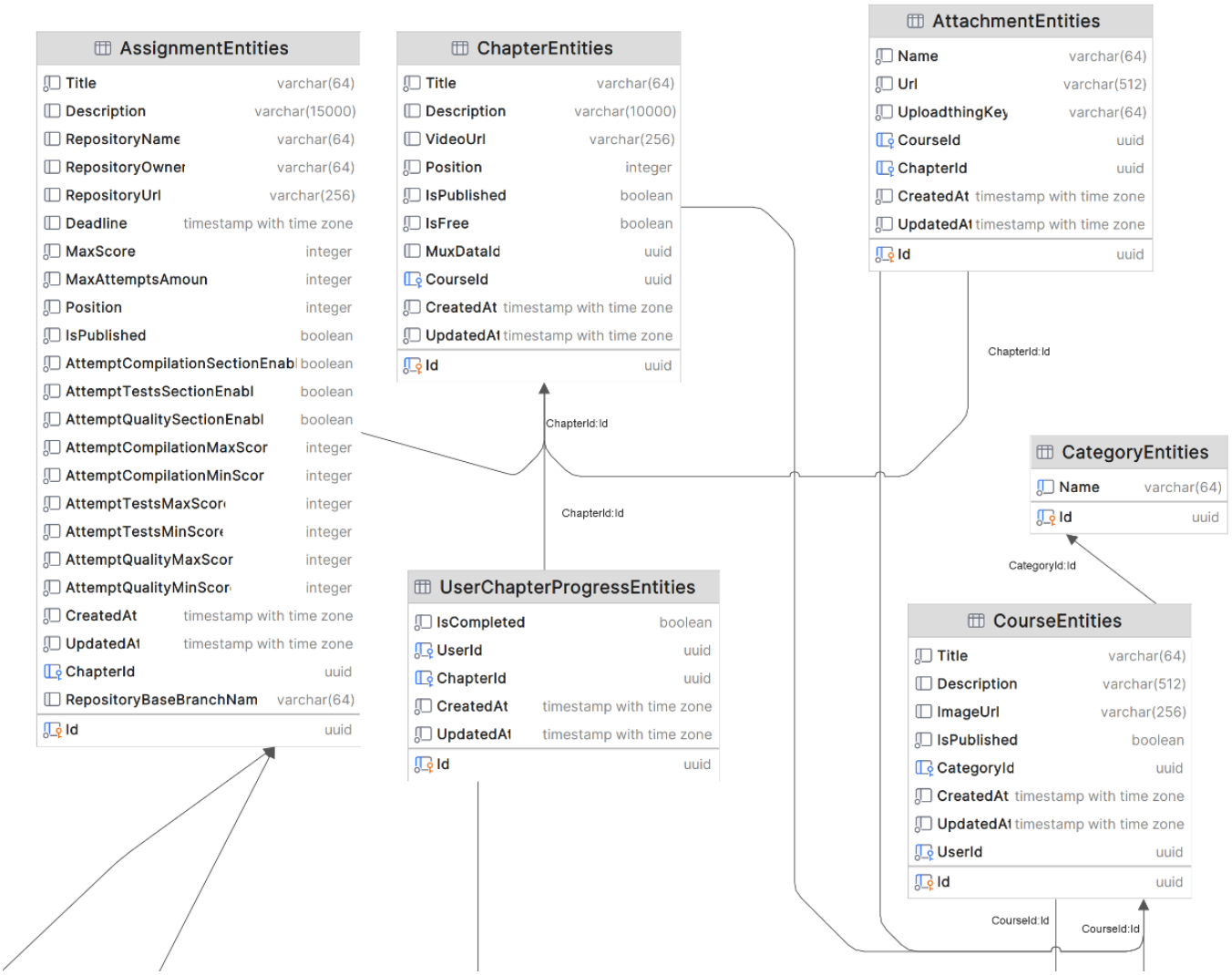

Рис. Г.1 Схема таблиць та їх взаємозв’язків основної бази даних платформи

Джерело: зроблено автором

На рисунку Г.1 представлено фрагмент реляційної моделі бази даних, що охоплює ключові сутності, пов’язані зі зберіганням та структурною організацією навчального контенту на платформі. Зокрема, розглянуто таблиці CourseEntities, CategoryEntities, ChapterEntities, AssignmentEntities, AttachmentEntities та UserChapterProgressEntities.

Таблиця CourseEntities є основною сутністю, яка відповідає за опис курсів, включаючи їхню назву, зображення, опис та стан публікації. Кожен курс належить до певної категорії, що задається зовнішнім ключем CategoryId і визначається в таблиці CategoryEntities. Остання містить мінімальний обсяг інформації — лише назву

категорії та її унікальний ідентифікатор, що достатньо для фільтрації та класифікації курсів.

Кожен курс містить розділи, які відображаються в таблиці ChapterEntities. Ця таблиця містить інформацію про назву, опис, посилання на відеоматеріали, позицію в межах курсу, а також прапорці публікації й доступності. Ієрархічна структура курсів реалізується через зв’язок CourseId, що дозволяє динамічно додавати або оновлювати розділи без порушення цілісності курсу.

До кожного розділу можуть бути прив’язані завдання — таблиця AssignmentEntities, що зберігає назву, опис, дедлайн, параметри GitHub-репозиторію та налаштування для оцінювання (включаючи активовані секції, межі оцінювання та кількість спроб). Така деталізація дозволяє адаптувати завдання до різних форматів перевірки знань — від автоматичної компіляції до повного тестування коду.

Таблиця AttachmentEntities забезпечує зберігання допоміжних матеріалів (презентацій, кодових прикладів, специфікацій тощо), що можуть бути пов’язані як з курсом загалом (CourseId), так і з конкретними розділами (ChapterId). Це дозволяє забезпечити студентів релевантними ресурсами у зручний для викладача спосіб.

Користувацький прогрес у вивченні контенту відображається через UserChapterProgressEntities, що містить інформацію про завершення користувачем певного розділу. Такий механізм дозволяє відслідковувати активність студентів, формувати звіти про навчальну динаміку та адаптувати освітній процес у реальному часі.

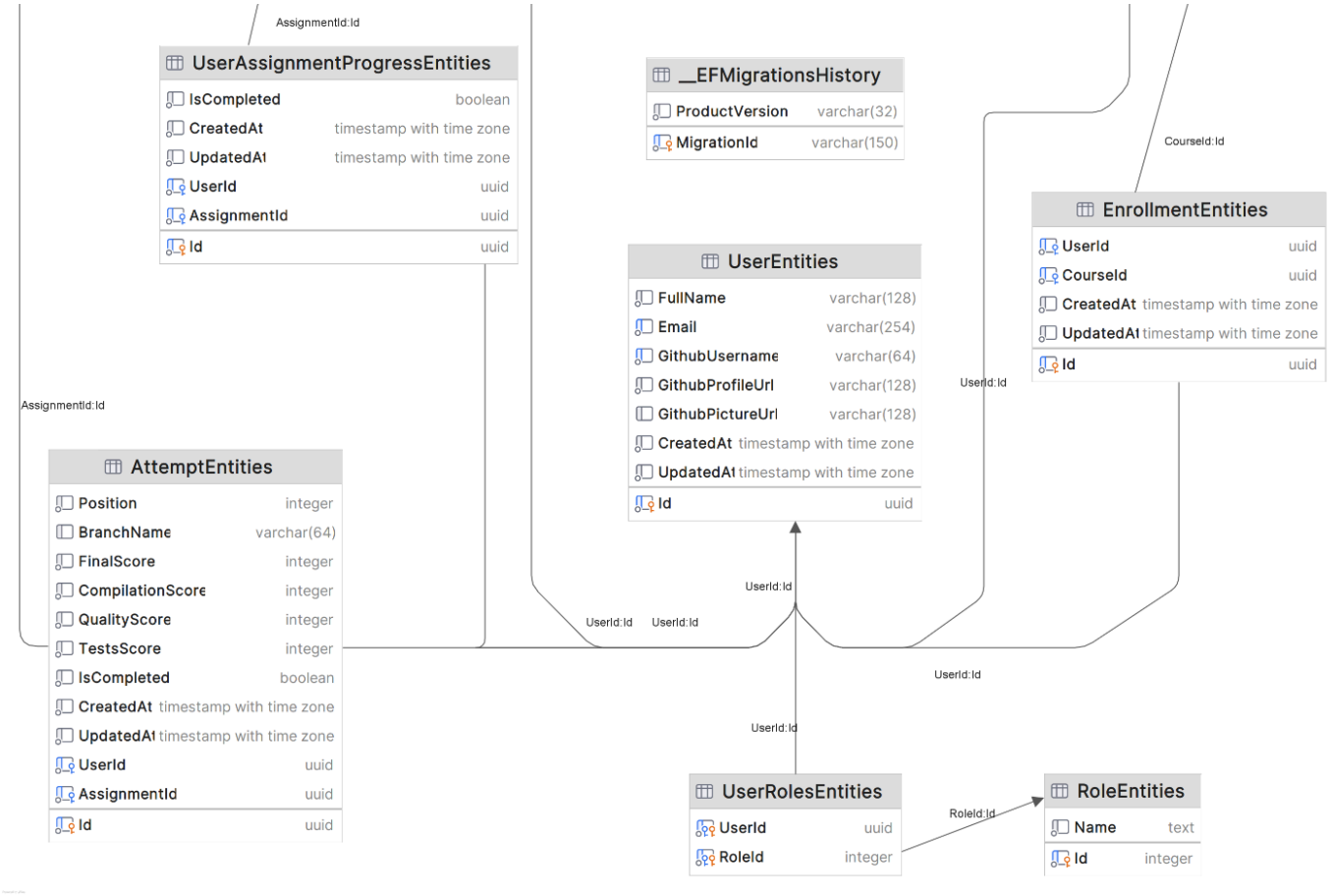

Рис. Г.2 Схема таблиць та їх взаємозв’язків основної бази даних платформи

Джерело: зроблено автором

На рисунку Г.2 представлено інші допоміжні таблиці, що забезпечують реалізацію логіки оцінювання, взаємодії користувачів з курсами, а також механізми автентифікації та авторизації. Зокрема, до цієї групи належать AttemptEntities, EnrollmentEntities, UserAssignmentProgressEntities, UserEntities, UserRolesEntities, RoleEntities та службова таблиця __EFMigrationsHistory.

Таблиця AttemptEntities містить інформацію про окремі спроби виконання студентом завдання, включаючи гілку репозиторію, оцінки за кожен компонент (компіляцію, якість коду, проходження тестів), фінальний бал, а також відмітку про завершеність спроби. Такий рівень деталізації дає змогу формувати прозорий механізм оцінювання з можливістю гнучкої конфігурації критеріїв оцінки на рівні кожної спроби.

У таблиці UserAssignmentProgressEntities зберігається інформація про факт завершення завдання конкретним користувачем. Це дозволяє системі не лише обмежувати подальші спроби відповідно до заданої кількості, а й забезпечувати контроль над послідовністю проходження матеріалу.

Таблиця EnrollmentEntities реалізує механізм запису користувача на курс, фіксуючи сам факт участі та час додавання. Вона забезпечує логічний зв’язок між студентом та відповідним навчальним контентом, і є основою для формування користувацьких потоків.

Сутність UserEntities виконує роль централізованого сховища інформації про користувача, включаючи його ім’я, електронну пошту, облікові дані GitHub (профіль, фото, ім’я користувача) та метадані створення. Наявність інтеграції з GitHub дозволяє здійснювати перевірку завдань безпосередньо на основі репозиторіїв студентів, що є ключовим для реалізації моделі автоматизованого оцінювання.

Контроль доступу реалізується за допомогою таблиць UserRolesEntities та RoleEntities, які відповідають за призначення ролей (наприклад, студент, викладач, адміністратор) конкретним користувачам. Це забезпечує гнучку реалізацію політик доступу на всіх рівнях системи. При цьому таблиця RoleEntities містить перелік доступних ролей, тоді як UserRolesEntities реалізує багатозначний зв’язок між користувачами та ролями.

Окремо варто згадати таблицю __EFMigrationsHistory, яка є службовою та використовується засобами Entity Framework для відстеження змін у схемі бази даних. Її присутність у структурі не є функціональною частиною предметного середовища, однак вона критично важлива для підтримання цілісності та відтворюваності моделі в процесі розробки.

Сукупність розглянутих таблиць демонструє ретельно спроєктовану модель, що дозволяє забезпечити як збереження навчального контенту, так і точне відображення взаємодій користувачів із системою. Високий рівень нормалізації, використання UUID-ідентифікаторів, чіткі зовнішні зв’язки та логічне розмежування функцій сутностей забезпечують масштабованість, розширюваність запропонованої архітектури.