

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних
технологій

КУРСОВА РОБОТА

на тему: «Проектування та реалізація модуля тестування програмного коду
для автоматизованих систем оцінювання»

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
економіко-математичного моделювання та
інформаційних технологій

Клебан Юрій Вікторович

Національна шкала: _____

Кількість балів: _____ **Оцінка:** ЕКТС _____

Члени комісії:

Острог, 2024

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1. Теоретичні аспекти тестування програмного коду в автоматизованих системах	6
1.1. Аналіз сучасних підходів до автоматизованого тестування програмного коду та визначення вимог до модулів оцінювання.	6
1.2. Огляд інструментів та технологій для автоматизованого тестування програмного коду в контексті автоматизованих систем оцінювання.	10
РОЗДІЛ 2. Прикладна розробка сервісу автоматизованого тестування	14
2.1. Характеристика технологічного стека та архітектурних принципів рішення.	14
2.2. Методика реалізації сервісу в рамках обраної архітектури.	16
2.2.1. Компонент управління даними (HomeworkAssignment.Database)	17
2.2.2. Компонент доменної логіки (HomeworkAssignment.Domain)	18
2.2.3. Компонент передачі даних (HomeworkAssignment.DTOs):	18
2.2.4. Компонент зберігання даних (HomeworkAssignment.Persistence):	20
2.2.5. Компонент логіки застосунку (HomeworkAssignment.Application)	22
2.2.6. Компонент API для взаємодії (HomeworkAssignment.API)	23
2.2.7. Компонент технічної підтримки (HomeworkAssignment.Infrastructure)	24
2.3. Практичні приклади застосування та використання сервісу.	24
ВИСНОВКИ	28
ДЖЕРЕЛА ЛІТЕРАТУРИ	29
ДОДАТКИ	30

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів даних і складності програмних систем. Як наслідок, забезпечення якості програмного забезпечення має вирішальне значення для розробників. Тестування коду програмного забезпечення є ключовим аспектом раннього виявлення помилок і забезпечення стабільності та надійності продуктів.

У сфері автоматизованих систем оцінювання, які широко використовуються в навчальних закладах для оцінювання знань учнів, тестування програмного коду стає особливо важливим. Код повинен не тільки правильно функціонувати, але й адаптуватися до різних умов використання. Тому існує нагальна потреба в розробці тестового модуля для автоматизованої перевірки якості програмного коду.

Важливість створення якісного програмного забезпечення підкреслюється насамперед економічними міркуваннями. Різні галузеві стандарти, включаючи ISO 25010, підкреслюють той факт, що якість програмного забезпечення можна оцінити за кількістю помилок, виявлених під час тестування. Згідно з цими рекомендаціями, середній рівень помилок становить приблизно 6 на кожні 1000 рядків коду.

В освітньому контексті викладачі стикаються зі значними витратами часу на оцінювання завдань студентів. Наприклад, на курсах із кількістю студентів від 30 до 300 зазвичай потрібно близько 15 хвилин, щоб вручну оцінити одне завдання. Це означає, що загальний час оцінювання може становити від 7,5 годин (для 30 студентів) до 75 годин (для 300 студентів).

Впровадження автоматизованих процесів перевірки коду має потенціал для значного підвищення ефективності навчання. Дослідження показують, що автоматизовані системи здатні надавати студентам зворотний зв'язок значно швидше, ніж традиційні ручні методи тестування. Це не тільки економить дорогоцінний час викладачів, але й дає можливість студентам оперативно виправляти помилки та вдосконалювати свої навички програмування. Крім того, автоматизовані системи можуть проводити ретельні та комплексні перевірки на різних рівнях, мінімізуючи ризик непомічених помилок, які можуть виникнути під

час оцінювання вручну. Отже, використання автоматизованих систем оптимізує як освітній процес, так і загальну якість програмного забезпечення.

Дослідження щодо автоматизованого тестування програмного коду проводили як вітчизняні, так і міжнародні вчені: Алессіо Гамбі, Войкан Г., Девід Ло, Ільїна Наталія Володимирівна, Джекі Кеунг, Костецька О. П., Л.Б. Ліщинська, Ляховецький О. О., Морзе Н. В., Рупіч С. С., Сізоненко К. В., Сін Ся, Цибульник С. О. та багато інших. Українські дослідники зосередилися на адаптації сучасних технологій до вимог національної освітньої системи, а зарубіжні – на створенні нових алгоритмів, моделей, впровадженні засобів тестування в розгалужені програмні системи. Їхні дослідження слугують як теоретичною, так і практичною основою для розробки ефективних рішень у цій галузі.

Усі наведені вище факти визначають актуальність теми дослідження «Проектування та реалізація модуля тестування програмного коду для автоматизованих систем оцінювання».

Метою дослідження є створення модуля тестування програмного коду для автоматизованих систем оцінювання знань студентів. Акцент робиться на аналізі поточних методів тестування, встановленні вимог до продукту, розробці архітектури модуля, його практичному впровадженні та тестуванні його функціональності.

Для досягнення поставленої мети були визначені наступні завдання дослідження:

1. Провести критичний аналіз сучасних підходів і рішень у сфері тестування програмного коду.

2. Сформулювати та обґрунтувати ключові вимоги до розроблюваного програмного модуля.

3. Розробити концептуальну та технічну архітектуру модуля тестування.

4. Реалізувати програмний продукт відповідно до спроектованої архітектури.

Об'єктом дослідження є процес автоматизації тестування програмного коду в системах оцінювання знань.

Предметом дослідження є методи, моделі та інструменти для проектування, впровадження та використання модулів автоматизованого тестування програмного коду.

Практичне значення полягає в розробці ефективного інструменту автоматизованого тестування, який можна легко інтегрувати в освітній платформи. Завдяки використанню цього модуля підвищиться точність оцінювання завдань студентів, скоротиться час перевірки, а навчальний процес збагатиться оперативним та неупередженим зворотним зв'язком.

Структура роботи. Дослідження складається із вступу, двох розділів та висновків. Бібліографія включає 17 найменувань. У роботі подано 5 рисунків та 6 лістингів коду. Додатки займають 7 сторінок. Обсяг роботи – 37 сторінок друкованого тексту.

РОЗДІЛ 1. Теоретичні аспекти тестування програмного коду в автоматизованих системах

1.1. Аналіз сучасних підходів до автоматизованого тестування програмного коду та визначення вимог до модулів оцінювання.

За останні кілька років автоматизоване тестування стало вирішальним компонентом процесу розробки програмного забезпечення. Зростаюча складність систем програмного забезпечення та необхідність швидкого виведення продуктів на ринок змусили компанії шукати ефективні рішення для підвищення ефективності тестування. Автоматизація дозволяє швидше виконувати тести, мінімізує ризик людської помилки та забезпечує послідовність результатів.

Згідно зі Звітом про світову якість Capgemini за 2023 рік, у 2023 році відбулося збільшення впровадження автоматизованого тестування організаціями на 44% порівняно з 34% у 2021 році [2]. Ці результати підкреслюють зростаюче значення та ефективність автоматизованих практик у розробці програмного забезпечення. Він підкреслює важливість включення автоматизації поряд із традиційними методами тестування для підвищення якості програмного забезпечення та оптимізації робочих процесів.

Команди тестувальників залишаються в курсі тенденцій автоматизації тестування програмного забезпечення, щоб забезпечити сприятливу віддачу від інвестицій (ROI). На рисунку 1 відображені результати, зазначені Звітом про стан якості за 2022 рік, які демонструють, що близько 80% компаній успішно досягли позитивної рентабельності інвестицій завдяки автоматизації [12]. Крім того, як повідомляється в результатах дослідження промисловості від Kubitron, 72% компаній зазначили, що вони виділяють від 10% до 49% свого загального бюджету на забезпечення якості на витрати, пов'язані з автоматизацією тестування [13]. Наведені дані свідчать про те, що впровадження автоматизації тестування — це не просто тимчасова мода, а критичний крок до підвищення ефективності та якості процедур розробки програмного забезпечення.

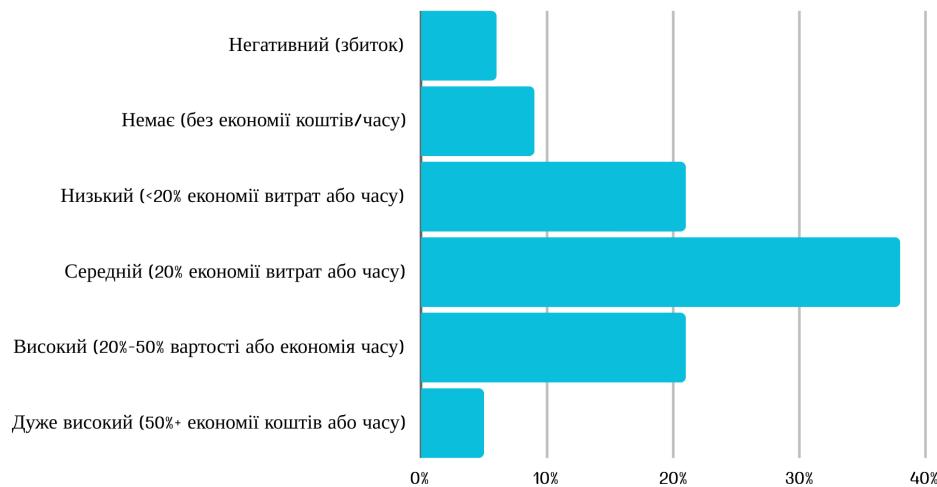


Рисунок 1.1.1. Рентабельність інвестицій на автоматизацію тестиування станом на 2022 рік.

Джерело: <https://katalon.com/resources-center/blog/the-state-of-quality-report-2022?ref=dogq.io>

Тестиування програмного коду відіграє вирішальну роль у життєвому циклі розробки програмного забезпечення, зосереджуючись на перевірці функціональності, надійності та відповідності вимогам. Цей систематичний процес призначений для виявлення будь-яких помилок, дефектів або невідповідностей у коді, які можуть вплинути на загальну якість кінцевого продукту.

Як зазначає ISTQB, тестиування — це комплексний процес, який охоплює всі види діяльності протягом життєвого циклу розробки програмного забезпечення, як статичні, так і динамічні, включаючи планування, підготовку та оцінку програмних продуктів і пов'язаних робочих продуктів, щоб переконатися, що вони відповідають заданим вимогам, продемонструвати придатність для своїх цільове призначення та виявлення дефектів [1].

Основні цілі тестиування включають виявлення дефектів програмного забезпечення, підтвердження відповідності системи специфікаціям і оцінку якості програмного забезпечення. ISTQB визначає різні методи тестиування, такі як чорний ящик, білий ящик та інші для ефективного виконання тестів. Різні типи тестиування, охоплені визначенням, включають одиничне, інтеграційне, системне, регресійне та приймальне тестиування. ISTQB підкреслює важливість документації в тестиуванні, включаючи плани тестиування, тестові випадки, звіти про результати тестиування та іншу відповідну документацію.

У цьому контексті доречно розглянути ключові методи, які набули популярності в останні роки. Нижче наведено основні підходи до тестування, які дозволяють командам покращити результати розробки програмного забезпечення.

Одним з основних методів, про який варто згадати, є тестування на основі поведінки (Behavior-Driven Development, BDD). Ця техніка передбачає створення тестових сценаріїв у зручному для користувача форматі, легко зрозумілому для всіх зацікавлених сторін, включаючи мову Gherkin, що сприяє покращенню спілкування між розробниками, тестувальниками та бізнес-аналітиками. Такі інструменти, як Cucumber і SpecFlow, відіграють важливу роль у полегшенні впровадження BDD, дозволяючи створювати автоматизовані тести у формі бізнес-сценаріїв [15].

Наступний метод передбачає тестування за допомогою паралельного виконання. Ця стратегія дозволяє одночасне тестування в кількох середовищах, що призводить до значного скорочення часу виконання. Такі інструменти, як Selenium Grid, TestNG і JUnit 5, підтримують тестування сумісності між браузерами та платформами.

Ще один популярний підхід – контейнерне тестування за допомогою Docker. Ця техніка дозволяє створювати ізольовані тестові середовища, забезпечуючи узгодженість тестових середовищ і спрощуючи процес масштабованості. Kubernetes можна використовувати для керування контейнерами, підвищуючи ефективність тестування.

Тестування API стало важливою практикою для перевірки функціональності, продуктивності та безпеки API. Використання таких інструментів, як Postman, RestAssured і SoapUI, зменшує залежність від готовності інтерфейсу, тим самим швидко виявляючи проблеми в логіці програми.

Інтеграція штучного інтелекту та машинного навчання в тестування відкриває нові можливості для оптимізації процесів. Такі інструменти, як Testim і AppliTools, дозволяють створювати тестові випадки, прогнозувати помилки та аналізувати журнали, підвищуючи адаптивність тестування до змін коду.

Тестування на основі моделі (MBT) передбачає автоматизовану генерацію тестів на основі системних моделей. Такі рішення, як GraphWalker і Spec Explorer,

пропонують повне охоплення складних сценаріїв тестування, мінімізуючи потребу в ручних зусиллях.

Включення тестування в безперервну інтеграцію/безперервне розгортання (CI/CD) є критично важливим елементом сучасних робочих процесів розробки. Автоматизоване виконання тестів для кожної модифікації коду за допомогою таких платформ, як Jenkins, GitLab CI/CD або Travis CI, не тільки прискорює цикл випуску, але й гарантує надійність кодової бази. Згідно з дослідженням DEVOPSdigest, використання інструментів CI/CD з автоматизованим тестуванням продемонструвало в середньому підвищення продуктивності на 20% порівняно з більш традиційними підходами [6].

Включення візуального регресійного тестування дозволяє відстежувати зміни в інтерфейсі користувача. Такі інструменти, як AppliTools і Percy, здатні визначати зміни інтерфейсу користувача, які можуть вплинути на взаємодію користувача.

Хмарне тестування набуває все більшої популярності як ефективний підхід. Використання таких інструментів, як BrowserStack і Sauce Labs, дозволяє масштабоване та дистанційне тестування, надаючи доступ до широкого спектра конфігурацій пристройів і браузерів без необхідності використання фіrmового обладнання.

Дизайн модулів оцінювання має відповідати поточним галузевим тенденціям. Наприклад, поширеність використання Selenium і JUnit серед 75% розробників для автоматизації тестування підкреслює необхідність включення підтримки цих інструментів у модуль оцінювання. Інтеграція з процесами CI/CD, яку, за прогнозами, запровадять 60% компаній до 2023 року, значно підвищить ефективність перевірки завдань, особливо в освітніх системах, як зазначено у звіті DEVOPSdigest [7].

1.2. Огляд інструментів та технологій для автоматизованого тестування програмного коду в контексті автоматизованих систем оцінювання.

Автоматизовані системи оцінювання використовуються в різних галузях, зокрема в освіті та фінансах, і дуже важливо визначити пріоритет якості тестування. Збої в цих системах можуть мати серйозні наслідки, оскільки вартість усунення проблем після випуску продукту буде значно вищою (у 30-100 разів), ніж усунення їх на ранніх стадіях розробки, таких як визначення вимог або проєктування [17]. Дослідження, проведені IBM, і оцінки таких поважних організацій, як Functionize і Ten10, також підкреслюють значне збільшення витрат, пов'язаних з виправленням помилок на пізніших етапах розробки.

Це підкреслює важливість не тільки розуміння поточних методологій автоматизованого тестування, але й встановлення критеріїв для модулів оцінювання, які гарантують точність, ефективність і гнучкість процедур тестування. У цьому розділі розглянуто основні платформи та технології, які можуть бути використані для автоматизації перевірки завдань.

В освітньому середовищі доступні різні платформи для автоматизації тестування коду учнів. Наступні параметри демонструють деякі з цих платформ:

Codecademy забезпечує інтерактивне навчання програмування з автоматичним тестуванням коду та зворотним зв'язком у реальному часі для покращення розвитку навичок. Учням коледжів і студентам вищих навчальних закладів, які відповідають вимогам, пропонуються спеціальні переваги та ресурси, зокрема доступ до інтерактивної навчальної програми за зниженою ціною понад 35% від звичайної ціни.

HackerRank пропонує студентам завдання та автоматично перевіряє їх код на правильність. Завдяки підтримці понад 30 мов програмування платформа дозволяє проводити онлайн-інтерв'ю за допомогою узгоджених інструментів. Платформа пропонує ряд категорій, наприклад SQL, Python і Java, а також спеціалізовані завдання, призначенні для підготовки кандидатів до співбесід з великими технологічними фірмами. Крім того, користувачі мають можливість створювати та ділитися власними завданнями на HackerRank.

LeetCode спеціалізується на виконанні завдань технічної співбесіди з автоматичною оцінкою рішень студентів для швидкого виявлення помилок і створення звіту. Це популярний ресурс для кандидатів, які готуються до співбесід у таких відомих компаніях, як Google, Amazon і Facebook [9]. *LeetCode* надає широкий спектр запитань, від простих до дуже складних, охоплюючи понад 14 мов програмування.

Codewars заохочує розв'язання проблем, нараховуючи бали за правильні рішення. Платформа пропонує систему рейтингів, яка дозволяє користувачам заробляти бали, вирішуючи завдання та просуваючись по рівнях. Крім того, користувачі мають можливість створювати власні кати (проблеми) і ділитися ними з іншими учасниками на *Codewars*.

EdX надає інтерактивні курси програмування на своїй освітній платформі, що дозволяє викладачам генерувати автоматизовані оцінки для оцінювання студентів. У звіті EdTech Trends Report 2023 передбачається послідовне розширення світового ринку електронного навчання з прогнозованою вартістю 491,35 мільярда доларів США до 2028 року [3].

Coursera пропонує інтерактивні курси програмування з практичними завданнями, які автоматично оцінюються за допомогою вбудованих систем перевірки коду. Викладачі мають можливість створювати завдання з автоматичним оцінюванням і пропонувати персоналізований зворотний зв'язок, полегшуєчи ефективний моніторинг просування студентів і забезпечуючи найкращі результати навчання.

Replit пропонує інтерактивне середовище кодування, де користувачі можуть писати та тестиувати код у реальному часі. Вона має інтегроване автоматизоване тестиування для швидкого зворотного зв'язку. Платформа забезпечує інтегроване середовище розробки зі співпрацею в реальному часі, що дозволяє кільком користувачам працювати над проєктами одночасно. *Replit* підтримує розміщення вебдодатків і баз даних, а також функції тестиування та налагодження.

GitHub Classroom, інструмент, спеціально розроблений для викладачів, дає змогу генерувати завдання з автоматичним оцінюванням за допомогою GitHub Actions. Ця платформа автоматизує процес створення репозиторіїв для окремих студентів або груп. Крім того, GitHub Classroom дозволяє створювати шаблони завдань, спрощуючи повторне використання та налаштування матеріалів для різних груп студентів.

Платформи, які розглядаються, надають різноманітні можливості для автоматизації тестування програмного коду, що дозволяє викладачам і студентам покращити процеси навчання та оцінювання. Існує помітне зростання інтересу до цих інструментів, що вказує на їх потенціал для інтеграції в навчальні програми.

Кожна платформа онлайн-навчання пропонує ряд функцій, адаптованих до конкретних уподобань і рівня досвіду користувачів. У наведеній нижче таблиці наведено дані щодо бази користувачів різних відомих онлайн-платформ для навчання програмування.

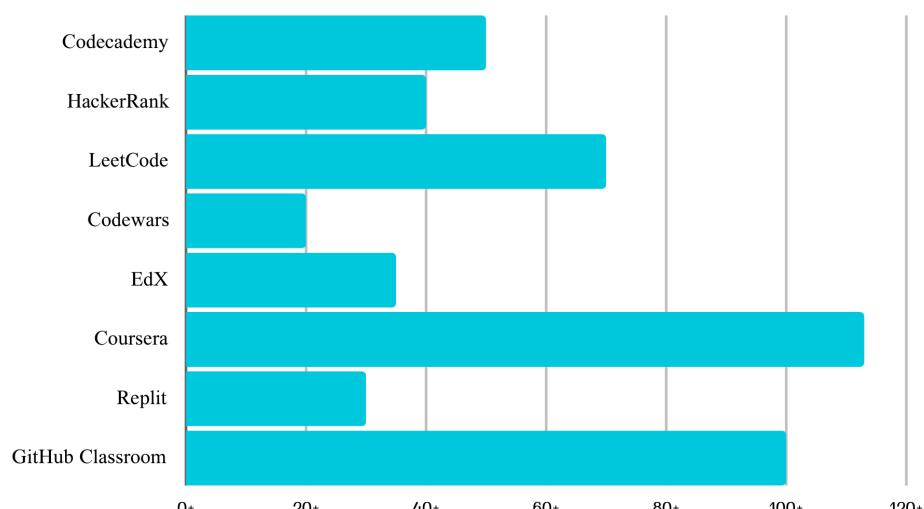


Рисунок 1.2.1 Кількість користувачів онлайн-платформ для навчання програмування (млн), станом на 2023 рік.

Джерело: зроблено автором

Аналіз даних про використання онлайн-платформ для навчання програмування у 2023 році показує суттєве уявлення про популярність цих платформ і тенденції онлайн-освіти в галузі програмування.

Серед розглянутих платформ виділяється Coursera з понад 113 мільйонами користувачів, що демонструє її широку глобальну привабливість [5]. Це можна пояснити її широкою пропозицією академічних курсів, співпрацею з університетами та навчальними закладами, надаючи користувачам можливість отримати міжнародно визнані сертифікати.

Попит на ресурси, зосереджені на практичних навичках програмування, очевидний завдяки таким платформам, як LeetCode (70+ мільйонів користувачів) і Codecademy (50+ мільйонів користувачів), які мають значну аудиторію [14, 4]. Ці платформи особливо затребувані для підготовки до технічних співбесід і вдосконалення алгоритмічних навичок.

У той час як такі платформи, як HackerRank і EdX, мають дещо менші користувацькі бази — понад 40 мільйонів і 35 мільйонів відповідно, вони пропонують різноманітні завдання та курси для покращення досвіду користувачів у певних сферах програмування, таких як розв'язання алгоритмічних проблем і знання технологій [11, 8].

Replit (30+ мільйонів користувачів) і GitHub Classroom (100+ мільйонів користувачів) займають унікальні ніші серед платформ програмування [16, 10]. Replit віддають перевагу розробникам за його інтегроване середовище для кодування та тестування, тоді як GitHub Classroom широко використовується навчальними закладами для оптимізації навчального процесу та керування завданнями студентів.

Загалом дані вказують на зростаючий інтерес до онлайн-навчання, оскільки платформи надають низку ресурсів для підвищення можливостей програмістів усіх рівнів кваліфікації.

РОЗДІЛ 2. Прикладна розробка сервісу автоматизованого тестування

2.1.Характеристика технологічного стека та архітектурних принципів рішення.

Вибір правильної технології є критично важливим кроком у процесі розробки, оскільки він безпосередньо впливає на ефективність і продуктивність системи. Це охоплює оцінку різних мов програмування, фреймворків, бібліотек і інструментів для реалізації сервісу, з акцентом на такі фактори, як модульність, масштабованість, безпека та інтеграція з існуючими системами.

Окрім вибору технології, встановлення архітектурних принципів має вирішальне значення для забезпечення стабільності та гнучкості рішення. У цьому розділі наведено огляд ключових архітектурних рішень, які можна використати для створення надійної служби автоматизованого тестування, разом з обґрунтуванням їх можливості в рамках вимог проєкту.

Рішення побудовано з використанням кількох ключових компонентів, таких як мова програмування C#, принципи чистої архітектури, шаблон команди-запиту(CQRS), RESTful API та реалізація Docker. Ці технології та методології працюють разом, щоб сформувати цілісну архітектуру, яка забезпечує високий рівень ефективності, гнучкості та масштабованості системи.

Ключовою технологією, яка використовується для цього рішення, є мова програмування C#, визнана своєю високою продуктивністю та потужними можливостями об'єктно-орієнтованого програмування. З наголосом на безпеці типів і розгалуженій бібліотечній екосистемі, особливо в .NET Core, C# чудово справляється зі створенням стійких програм на стороні сервера. У нашому проєкті C# є невіддільною частиною реалізації основної бізнес-логіки, гарантуючи надійність і гнучкість програмного коду.

Архітектурний дизайн проєкту дотримується принципів чистої архітектури, яка передбачає розбиття програми на окремі модулі з метою спрощення тестування, обслуговування та модифікації коду. Цей модульний підхід підвищує гнучкість у тестуванні та модифікації коду, оскільки кожному компоненту призначається чітко

визначена роль і він працює незалежно від інших частин системи. Чиста архітектура полегшує модифікацію або заміну окремих модулів без необхідності переписувати інші розділи програми, тим самим мінімізуючи ризики під час масштабування системи та сприяючи її довгостроковій підтримці. Відокремлюючи бізнес-логіку від деталей реалізації, дизайн забезпечує цілеспрямований підхід до важливих аспектів, таких як бізнес-процеси та функціональність додатків, залишаючись адаптованим до змін в інших елементах, не порушуючи загальну роботу системи.

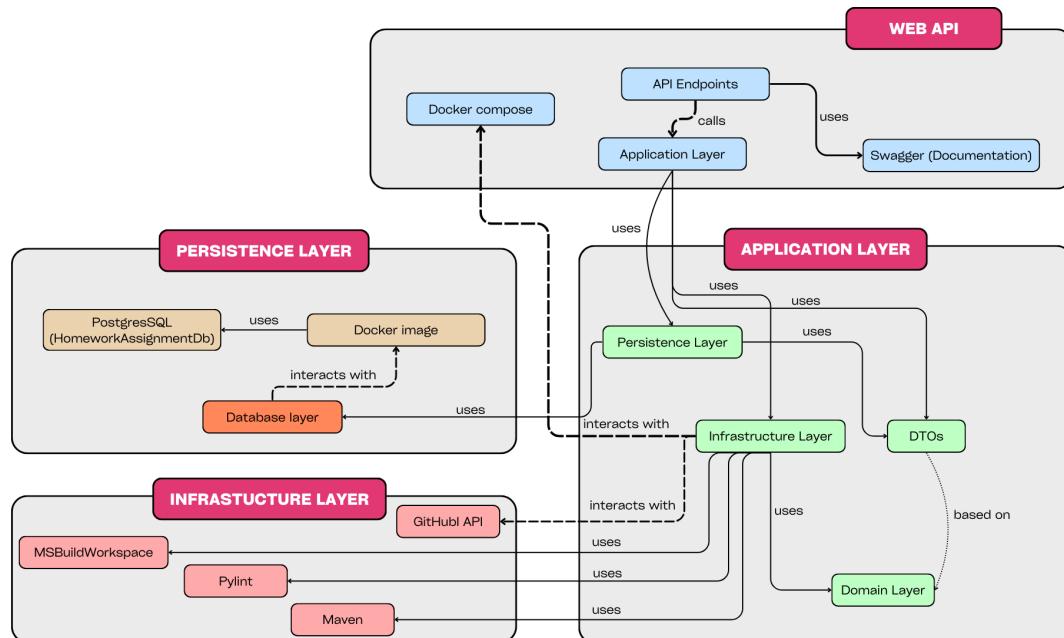


Рис. 2.1.1 Архітектура системи HomeworkAssignment.

Джерело: зроблено автором

Крім того, дотримання принципів SOLID відіграє ключову роль в архітектурному підході. Наприклад, принцип єдиної відповідальності (SRP) гарантує, що кожен клас виконує одне завдання, тоді як принцип відкритості/закритості (OCP) дозволяє розширити функціональність системи без зміни існуючого коду. Інші принципи, такі як заміна Лісков (LSP) і інверсія залежностей (DIP), пропонують гнучкість і полегшують підтримку.

Для забезпечення ефективного управління системними операціями використовується реалізація шаблону «Команда-Запит» (CQRS). Цей шаблон розділяє операції на команди, які змінюють стан системи, і запити, які отримують дані. Застосовуючи цей підхід, служба може оптимізувати свою продуктивність

шляхом рівномірного розподілу робочого навантаження між операціями читання та запису, сприяючи масштабуванню окремих компонентів.

Крім того, система використовує RESTful API як важливий компонент. Цей інтерфейс використовує стандартні методи HTTP, щоб забезпечити простий і зручний засіб доступу до функцій служби. Такий вибір дизайну забезпечує бездоганну сумісність із різноманітними клієнтськими програмами та забезпечує швидку інтеграцію із зовнішніми системами.

Для оптимізації та керування середовищем розробки та розгортання Docker реалізовано для цілей контейнеризації. Цей інструмент дозволяє створювати ізольовані тестові середовища та сприяє стабільноті в локальному та робочому середовищах. Крім того, Docker полегшує розгортання та масштабування системи, особливо якщо його доповнюють такі оркестрові платформи, як Kubernetes.

Усі зазначені технології та принципи формують основу нашого сервісу, забезпечуючи його високу продуктивність, адаптивність та простоту підтримки.

2.2. Методика реалізації сервісу в рамках обраної архітектури.

Впровадження служби автоматизованого тестування керується вибраною архітектурою, яка поєднує принципи чистої архітектури, шаблон команди-запиту (CQRS), RESTful API для взаємодії компонентів і контейнеризацію Docker для масштабованості та ізоляції середовища.

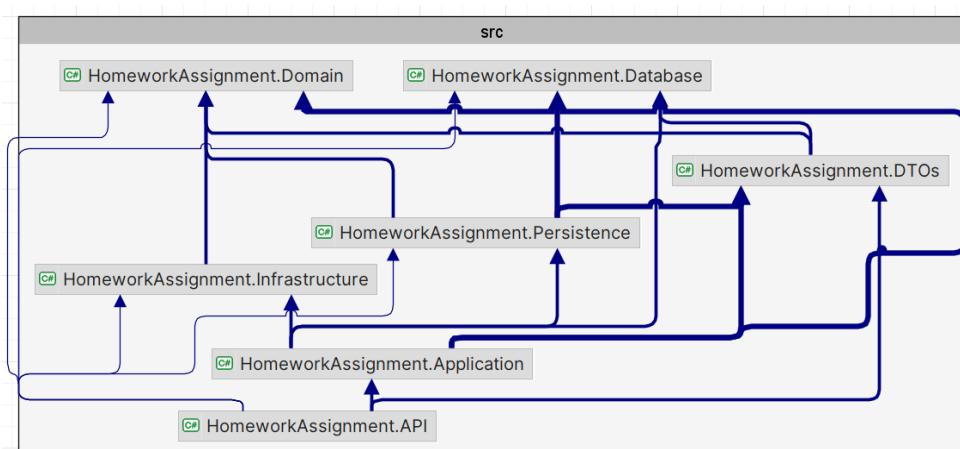


Рис. 2.2.1 Загальна структура проектів та їхні залежності.

Джерело: зроблено автором

Нижче наведено детальний огляд основних елементів системи та їх відповідних функцій.

2.2.1. Компонент управління даними (*HomeworkAssignment.Database*)

Компонент, що контролює збереження та пошук даних у базі даних. Він містить параметри підключення та конфігурації інтеграції для Entity Framework, а також визначення моделей, які віддзеркалюють таблиці бази даних. Важливим аспектом цього компонента є використання провайдерів для визначення типу бази даних, зазначеного у файлі конфігурації. Щоб досягти цього, компонент використовує механізм залежностей через IServiceCollection, уможливлюючи адаптовану конфігурацію та ініціалізацію контекстів бази даних на основі параметрів.

Лістинг 1. Клас “DependencyInjection” модуля *HomeAssignment.Database*.

```
public static class DependencyInjection
{
    public static void AddDatabaseServices(this IServiceCollection services)
    {
        services.AddScoped<IHomeworkAssignmentDbContextProvider,
        HomeworkAssignmentDbContextProvider>();

        services.AddScoped<IHomeworkAssignmentDbContextFactory>(provider =>
        {
            var contextFactoryProvider =
            provider.GetService<IHomeworkAssignmentDbContextProvider>();
            return contextFactoryProvider!.GetFactory();
        });

        services.AddScoped<IHomeworkAssignmentDbContext>(provider =>
        {
            var contextFactory =
            provider.GetService<IHomeworkAssignmentDbContextFactory>();
            return contextFactory!.CreateDbContext();
        });
    }
}
```

У межах цього компонента, використовуючи код *DependencyInjection*, відповіальність лежить на налаштуванні та реєстрації служб, які забезпечують доступ до бази даних через *DbContext*. Ключові елементи включають:

IHomeworkAssignmentDbContextProvider : цей постачальник визначає тип контексту бази даних, який буде використовуватися. Така гнучкість дозволяє плавно змінювати тип бази даних без зміни основного коду програми.

IHomeworkAssignmentDbContextFactory : Виконуючи роль фабрики, цей компонент генерує контекст бази даних, отримуючи дані від постачальника. Він створює екземпляр DbContext, спрощуючи процес створення контексту.

IHomeworkAssignmentDbContext : це сам контекст бази даних, що полегшує взаємодію з базою даних шляхом реалізації основних операцій CRUD (створення, читання, оновлення, видалення).

2.2.2. Компонент доменної логіки (*HomeworkAssignment.Domain*)

Компонент доменної логіки керує логікою домену програми, охоплюючи основні сутності та функції журналювання. Він складається з різних класів та інтерфейсів, які втілюють бізнес-логіку програми, а також інструменти для налаштування та використання функцій журналювання.

Прикладом такої сутності може бути сутність «Student», який слугує представленням студентів у системі. Ця сутність розроблена для включення інтерфейсів *IUser* і *IHaveGitHubProfile*, що забезпечує безперебійну взаємодію зі студентом як користувачем і полегшує створення профілю GitHub.

Ключові моменти цього класу:

Інтерфейс *IUser* визначає базові властивості для будь-якого користувача, включаючи ідентифікатор користувача, повне ім'я, електронну пошту, хеш пароля, роль, а також час створення та оновлення об'єкта користувача.

Інтерфейс *IHaveGitHubProfile* містить додаткові атрибути, необхідні для інтеграції GitHub. Він містить властивості для профілю GitHub, зокрема унікальний ідентифікатор профілю, ім'я користувача GitHub, URL-адресу профілю та URL-адресу зображення профілю.

2.2.3. Компонент передачі даних (*HomeworkAssignment.DTOs*):

Компонент DTOs забезпечує передачу даних між різними частинами системи, зокрема між сервером та клієнтами. Використання Data Transfer Objects дозволяє ефективно перетворювати складні структури даних у простіші формати, зручні для

передачі. DTOs відповідають за серіалізацію та десеріалізацію даних у форматах, таких як JSON, що є важливим для реалізації API.

У контексті цього проекту компонент передачі даних ефективно інтегрує AutoMapper для оптимізації перетворення об'єктів на різних рівнях системи. Ця інтеграція покращує процес передачі даних між моделлю домену та об'єктом передачі даних, автоматизуючи процедури відображення та зменшуючи ручні зусилля, необхідні для визначення протоколів перетворення для кожного об'єкта.

Лістинг 2. Клас “RequestStudentDtoValidator” модуля HomeAssignment.DTOs.

```
public class RequestStudentDtoValidator : AbstractValidator<RequestStudentDto>
{
    private const int MinPasswordLength = 8;
    private const int MaxLengthNamePropertyLength = 128;
    private const int MaxLengthUserNamePropertyLength = 64;
    private const int MaxLengthUrlPropertyLength = 128;

    public RequestStudentDtoValidator()
    {
        RuleFor(dto => dto.FullName)
            .NotNull().NotEmpty().WithMessage("Full name cannot be empty.")
            .MaximumLength(MaxLengthNamePropertyLength)
            .WithMessage($"Full name cannot exceed {MaxLengthNamePropertyLength} characters.");

        RuleFor(dto => dto.Password)
            .NotNull().NotEmpty().WithMessage("Password cannot be empty.")
            .MinimumLength(MinPasswordLength)
            .WithMessage($"Password must be at least {MinPasswordLength} characters long.")
            .Matches(@"[A-Z]").WithMessage("Password must contain at least one uppercase letter.")
            .Matches(@"[a-z]").WithMessage("Password must contain at least one lowercase letter.")
            .Matches(@"\d").WithMessage("Password must contain at least one number.")
            .Matches(@"[\w_]").WithMessage("Password must contain at least one special character.");

        RuleFor(dto => dto.Email)
            .NotNull().NotEmpty().WithMessage("Email cannot be empty.")
            .EmailAddress().WithMessage("Invalid email format.");

        RuleFor(dto => dto.GithubUsername)
            .NotNull().NotEmpty().WithMessage("Github username cannot be empty.");
    }
}
```

```

    .MaximumLength(MaxLengthUserNamePropertyLength)
    .WithMessage($"Github username cannot exceed
{MaxLengthUserNamePropertyLength} characters.");

    RuleFor(dto => dto.GithubProfileUrl)
        .NotNull().NotEmpty().WithMessage("Github profile url cannot be
empty.")
        .MaximumLength(MaxLengthUrlPropertyLength)
        .WithMessage($"Github profile url cannot exceed
{MaxLengthUrlPropertyLength} characters.");

    RuleFor(dto => dto.GithubPictureUrl)
        .NotEmpty().WithMessage("Github picture url cannot be empty.")
        .MaximumLength(MaxLengthUrlPropertyLength)
        .WithMessage($"Github picture url cannot exceed
{MaxLengthUrlPropertyLength} characters.")
        .When(dto => dto.GithubPictureUrl != null);
}
}

```

Крім того, FluentValidation використовується для забезпечення виконання правил перевірки для DTO. Як показано в наданому вище лістингу, клас RequestStudentDto встановлює правила перевірки за допомогою FluentValidation. Ці правила охоплюють перевірку обов'язкових полів, обмежень щодо довжини символів, форматів пароля, електронної пошти та URL-адреси, а також критеріїв складності пароля (наприклад, включення великих літер, цифр і спеціальних символів). Перевірка слугує гарантією точності отриманих даних, підтримуючи таким чином цілісність і безпеку системи.

2.2.4. Компонент зберігання даних (*HomeworkAssignment.Persistence*):

Компонент зберігання даних, або Persistence, забезпечує прямий доступ до бази даних через ORM (Object-Relational Mapping). Він пропонує такі функції обробки даних, як створення, оновлення, видалення та отримання завдань, результатів спроб і профілів користувачів.

Лістинг 3. Клас “CreateAssignmentCommandHandler” модуля HomeAssignment.Persistence.

```

public sealed class CreateAssignmentCommandHandler :
 IRequestHandler<CreateAssignmentCommand, RespondAssignmentDto>
{

```

```

private readonly IHomeworkAssignmentDbContext _context;
private readonly IMapper _mapper;

public CreateAssignmentCommandHandler(IHomeworkAssignmentDbContext context,
IMapper mapper)
{
    _context = context ?? throw new ArgumentNullException(nameof(context));
    _mapper = mapper ?? throw new ArgumentNullException(nameof(mapper));
}

public async Task<RespondAssignmentDto> Handle(CreateAssignmentCommand command,
CancellationToken cancellationToken)
{
    if (command is null) throw new ArgumentNullException(nameof(command));

    var compilationSection =
_mapper.Map<ScoreSection>(command.AssignmentDto.CompilationSection);
    var testsSection =
_mapper.Map<ScoreSection>(command.AssignmentDto.TestsSection);
    var qualitySection =
_mapper.Map<ScoreSection>(command.AssignmentDto.QualitySection);

    var assignment = Assignment.Create(
        command.AssignmentDto.OwnerId,
        command.AssignmentDto.Title,
        command.AssignmentDto.Description,
        command.AssignmentDto.RepositoryName,
        command.AssignmentDto.Deadline,
        command.AssignmentDto.MaxScore,
        command.AssignmentDto.MaxAttemptsAmount,
        compilationSection,
        testsSection,
        qualitySection
    );

    var assignmentEntity = _mapper.Map<AssignmentEntity>(assignment);
    var addedEntity = await
_context.AssignmentEntities.AddAsync(assignmentEntity, cancellationToken);

    return _mapper.Map<RespondAssignmentDto>(addedEntity.Entity);
}
}

```

Застосовуючи принцип CQRS (Command Query Responsibility Segregation), цей компонент відокремлює операції зміни стану даних від операцій пошуку даних. Команди зміни, як-от створення завдання або оновлення результатів спроби,

реалізовані для модифікації даних. З іншого боку, запити використовуються для операцій пошуку даних, таких як пошук завдань або перегляд історії спроб. Такий підхід оптимізує обробку та виконання різних операцій з даними, забезпечуючи чітку відповідальність за кожен тип операції.

2.2.5. Компонент логіки застосунку (*HomeworkAssignment.Application*)

Компонент програми відповідає за керування бізнес-логікою та обробку запитів від користувачів або інших систем, які взаємодіють із програмою. Він координує та виконує різні операції, включаючи оцінку завдань, перевірку тестів і компіляцію коду. Він також взаємодіє з різними частинами системи.

GitHubService є важливою частиною цього компонента, зв'язуючись із такими службами, як IAssignmentService, IBranchService, ICommitService, IGitHubBuildService, IStudentService та ITeacherService для оцінки успішності студентів за допомогою даних зі сховищ GitHub. Ця служба аналізує конкретні завдання, переглядає коміти та розгалуження, а також оцінює якість коду, компіляції та виконання тестів.

Детальну реалізацію GitHubService можна знайти в Додатку А.

Основні методи цього сервісу включають:

GetStudentBranches: отримує список гілок сховища студентів, пов'язаних із певним завданням. Цей метод перевіряє дані студентів і викладачів, а також перевіряє гілки, де студент зробив коміти.

VerifyProjectCompilation: Визначає, чи проект студента компілюється на певній гілці, досліджуючи останній коміт студента та використовуючи IGitHubBuildService для виконання перевірки компіляції. Метод повертає максимальний бал компіляції, якщо проект компілюється успішно.

VerifyProjectQuality: Оцінює якість коду зобов'язань студентів, використовуючи IGitHubBuildService для призначення оцінки якості. Остаточний бал розраховується на основі діапазону між мінімальним і максимальним балами якості.

VerifyProjectTests: Перевіряє кількість успішно складених тестів у зобов'язаннях студента. Цей метод використовує IGitHubBuildService для оцінки відсотка пройдених тестів і генерує відповідну оцінку тесту.

Усі помилки керуються шляхом реєстрації їхніх відповідних повідомлень за допомогою ILogger, допомагаючи в документуванні історії помилок і покращуючи відстеження проблем.

2.2.6. Компонент API для взаємодії (*HomeworkAssignment.API*)

Компонент API забезпечує взаємодію між клієнтським інтерфейсом та сервісами бізнес-логіки. Він надає кінцеві точки доступу для виконання операцій, таких як отримання інформації про завдання або подання результатів оцінювання. API використовує контролери для обробки HTTP-запитів і повернення відповідей, забезпечуючи таким чином функціональність інтерфейсу.

Наведемо приклад типового методу, що є частиною контролера, що відповідає за взаємодію користувача через вебінтерфейс.

Лістинг 4. Метод “GetProjectQualityVerification” модуля HomeAssignment.API.

```
[HttpGet("quality/{githubProfileId:guid}/{assignmentId:guid}/{branch}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public async Task<ActionResult<int>> GetProjectQualityVerification(Guid
githubProfileId, Guid assignmentId,
    string branch)
{
    var result = await _gitHubService.VerifyProjectQuality(githubProfileId,
assignmentId, branch);
    return StatusCode(StatusCodes.Status200OK, result);
}
```

Метод GetProjectQualityVerification обробляє запит HTTP GET для оцінки якості коду проекту студента. Для цього потрібні три параметри: githubProfileId(ідентифікатор профілю студента на GitHub), assignmentId(ідентифікатор завдання) і branch(назва гілки сховища). Метод використовує _gitHubService службу для оцінки якості коду для зазначеного завдання. Отримавши результати оцінювання, метод відповідає статусом 200 OK, вказуючи оцінку якості як числове значення. У разі будь-яких помилок під час виконання запиту повертається статус 500 Internal Server Error.

Цей метод оптимізує оцінку якості коду, використовуючи дані з GitHub, переводячи оцінку в числову оцінку якості. Стандартні статуси HTTP використовуються для вказівки успіху чи невдачі операції.

2.2.7. Компонент технічної підтримки (*HomeworkAssignment.Infrastructure*)

Компонент відповідає за безпосередню взаємодію з репозиторієм і виконання завдань, пов'язаних із перевіркою компіляції, виконанням тестів і оцінкою коду. Він надає послуги для роботи з різними мовами програмування, такими як C#, Java і Python.

Наприклад, служба *DotNetTestsRunner* (додаткову інформацію про комплексне впровадження DotNetTestsRunner див. у Додатку Б) може запускати тести для проектів, розроблених на платформі .NET, ініціюючи процес командного рядка за допомогою dotnet. Подібним чином можна створювати служби для проектів Java і Python для запуску тестів за допомогою відповідних інструментів, таких як Maven та pytest відповідно. Крім того, цей компонент керує результатами тестування, включаючи успішні та невдалі тести, і використовує регулярні вирази для аналізу результатів. Будь-які помилки, виявлені під час виконання тесту, записуються для подальшого дослідження, забезпечуючи зручний і прозорий процес оцінювання.

Компоненти рішення взаємодіють через чітко визначені інтерфейси та служби, що забезпечує гнучкість, масштабованість і легкість розширення. Чіткий розподіл обов'язків між бізнес-логікою, даними та інтерфейсом сприяє модульності та спрощує обслуговування та розробку програм.

2.3. Практичні приклади застосування та використання сервісу.

Автоматизоване тестування відіграє значну роль у сучасній розробці програмного забезпечення. У цьому розділі розглядаються практичні сценарії, коли служба автоматизованого тестування може виявитися корисною для різних груп користувачів, включаючи викладачів, студентів, а також організацій.

Використовуючи цю можливість, оцінку програмного коду можна оптимізувати та пришвидшити, гарантуючи неупереджені результати. Крім того, ця система допомагає зменшити навантаження на викладачів і дозволяє студентам самостійно переглядати свою роботу перед здачею, що зрештою економить час для всіх залучених сторін. Ось кілька прикладів, які демонструють, як цю послугу можна застосувати на практиці.

Сценарій 1: Впровадження автоматизації для оцінювання завдань студентів.

Використовуючи службу автоматизованого тестування, викладачі можуть спростити процес перевірки завдань студентів. Ця є цінним інструментом для ефективної перевірки студентських робіт, що зрештою спрощує процес оцінювання.

Створення завдання викладачем: викладач розробляє завдання, яке містить тести для автоматичної перевірки коду. Це може включати оцінку функцій, перевірку точності результатів або оцінку ефективності алгоритмів.

Подання коду студента: після виконання завдання студент надсилає свій код через вебінтерфейс або спеціалізований клієнт. Потім система отримує код і починає процес перевірки.

Автоматичне тестування: сервіс компілює наданий код, виконує тести та створює звіт. Цей звіт містить:

- Оцінку успішності компіляції: рейтинг, що вказує, чи код скомпільовано без помилок.
- Прохідні бали тесту: оцінка того, чи відповідає код вимогам тесту.
- Бали якості коду: бали, які нараховуються за дотримання стандартів програмування.



Рис. 2.3.1 Загальний вигляд звіту про спробу виконання завдання.

Джерело: зроблено автором

Використовуючи цей підхід, викладачі можуть автоматизувати оцінювання типових завдань, заощаджуючи значний час, який раніше витрачався на ручне тестування. Крім того, це сприяє об'єктивнішому оцінюванню завдяки дотриманню прозорих критеріїв тестування.

Сценарій 2: Самостійне навчання студента.

Студенти можуть скористатися послугою автоматизованого тестування, щоб самостійно оцінити свої навички програмування перед здачею завдань. Це дає їм можливість отримати швидкий відгук і підвищити якість свого коду перед остаточним надсиланням. Цей метод дозволяє учням самостійно вдосконалювати свої технічні навички, пропонуючи негайну перевірку та виправлення помилок без необхідності покладатися на відгуки викладачів. Завдання можна надсилати через вебінтерфейс або командний рядок (CLI), при цьому сервіс проводить автоматизоване тестування та надсилає відповідний відгук.

Це сприяє самостійному навчанню та зростанню, оскільки студенти можуть оцінювати свої помилки, отримувати висновки та проводити численні тести без втручання викладача. Цей метод дає змогу їм засвоїти матеріал, покращити свої навички програмування та набути досвіду, який виявиться безцінним у вирішенні складніших завдань.

Функція автоматизованого тестування дозволяє студентам працювати над завданнями, не покладаючись на численні відгуки від викладачів, тим самим прискорюючи процес навчання та сприяючи зосередженню уваги на розвитку практичних навичок.

Сценарій 3: Компанії проводять автоматизоване тестування коду кандидатів на вакансії.

Використовуючи автоматизоване тестування під час процесу відбору, компанії можуть підвищити ефективність і точність під час оцінювання кандидатів, мінімізуючи можливість упередженості та помилок, які можуть виникнути під час ручного тестування. Автоматизовані системи можуть швидко оцінювати багатьох кандидатів, забезпечуючи оперативний зворотний зв'язок і прискорюючи процес тестування.

Крім того, автоматизація тестування коду допомагає забезпечити справедливість оцінювання кандидатів, піддаючи всіх осіб однаковому стандартизованому оцінюванню, незалежно від їхнього походження чи зв'язків. Це допомагає створити рівні умови для всіх кандидатів, надаючи їм рівні можливості

продемонструвати свої технічні здібності. Цей підхід дозволяє рекрутерам і технічним фахівцям зосередитися на більш стратегічних аспектах відбору, таких як оцінка відповідності кандидата корпоративній культурі та потенціалу для зростання в команді, а не наголошувати виключно на технічній кваліфікації.

Сценарій 4: Навчальні курси чи хакатони.

Автоматизовану платформу тестування можна використовувати для проведення навчальних семінарів або програмування, де люди можуть брати участь у вирішенні завдань у режимі реального часу. Це не тільки дозволяє перевірити точність виконання завдань, але й сприяє здоровій конкуренції між учасниками. Такий підхід забезпечує справедливе та неупереджене оцінювання учасників через автоматизовані системи, усуваючи необхідність ручного оцінювання суддями. Учасники можуть оперативно отримувати відгуки та вдосконалювати свої навички програмування, сприяючи більш ефективному навчанню.

Таким чином, застосування даного сервісу в різних сценаріях покращує оптимізацію процесів тестування та оцінювання програмного забезпечення, сприяючи ефективності, гнучкості та об'єктивності. Впровадження автоматизованих систем тестування не лише економить час, але й зменшує навантаження на викладачів та координаторів заходів, одночасно забезпечуючи прозоре середовище для всіх учасників.

ВИСНОВКИ

У ході цього дослідження було проведено аналіз і розробку модуля автоматизованого тестування програмного коду з можливістю впровадження в системи оцінювання знань студентів. Створення цього модуля є доречним в умовах сучасного етапу прогресивних інформаційних технологій, де вимоги до якості програмного забезпечення продовжують зростати разом зі складністю програмних систем та обсягом даних.

Результати дослідження підкреслюють важливість автоматизованих систем тестування в освітніх установах, оскільки вони пропонують суттєву економію часу викладачам при оцінюванні завдань, зменшують вплив людських помилок і забезпечують точнішу та неупереджену оцінку програмних рішень учнів. Крім того, ці системи пропонують учням миттєвий зворотний зв'язок, дозволяючи їм швидко виправляти помилки та вдосконалювати свої навички програмування.

Дослідження успішно досягло своєї мети, розробивши функціональний модуль тестування коду, який відповідає вимогам сучасних автоматизованих систем оцінювання. У реалізації модульної архітектури використано передові технології, включаючи мову програмування C#, принципи чистої архітектури та шаблон CQRS, що забезпечує гнучкість і масштабованість. Крім того, модуль пропонує можливості інтеграції з існуючими освітніми платформами, підвищуючи ефективність навчального процесу.

Результати цього дослідження мають значні практичні наслідки для навчальних закладів, дозволяючи автоматизувати перевірку завдань студентів, скоротити час оцінювання та підвищити якість зворотного зв'язку. Ці досягнення сприяють підвищенню ефективності навчання та просуванню навичок програмування серед студентів.

На основі виконаної роботи можна визначити, що автоматизація кодового тестування є важливим кроком у вдосконаленні систем оцінювання знань і може значно покращити освітній досвід у вищих навчальних закладах.

ДЖЕРЕЛА ЛІТЕРАТУРИ

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. Capgemini. (2023). WQR 2023 Final Report. URL: [https://prod.ucwe.capgemini.com/at-de/wp-content/uploads/sites/11/2023/11/WQR_2023_FINAL_WEB\(CG\).pdf](https://prod.ucwe.capgemini.com/at-de/wp-content/uploads/sites/11/2023/11/WQR_2023_FINAL_WEB(CG).pdf) (Дата звернення: 14.11.2024).
3. Classter. (2023). EdTech Statistics 2023. URL: <https://www.classter.com/blog/edtech/edtech-statistics-2023/> (Дата звернення: 14.11.2024).
4. Codecademy. (2023). Codecademy Platform Overview. URL: <https://www.codecademy.com> (Дата звернення: 07.11.2024).
5. Coursera. (2023). Coursera Platform Statistics. URL: <https://www.coursera.org> (Дата звернення: 07.11.2024).
6. DevOps Digest. (2023). State of Test Automation 2023. URL: <https://www.devopsdigest.com/state-of-test-automation-2023> (Дата звернення: 10.11.2024).
7. DevOps Digest. (n.d.). DevOps Digest Blog. URL: <https://www.devopsdigest.com/> (Дата звернення: 09.11.2024).
8. EdX. (2023). EdX Platform Overview and Statistics. EdX. URL: <https://www.edx.org> (Дата звернення: 07.11.2024).
9. GitHub. (2022). Leetcode Company-Wise Problems 2022. URL: <https://github.com/hxu296/leetcode-company-wise-problems-2022> (Дата звернення: 05.11.2024).
10. GitHub. (2023). GitHub Classroom Overview. GitHub. URL: <https://classroom.github.com> (Дата звернення: 07.11.2024).
11. HackerRank. (2023). The State of Quality Report 2023. URL: <https://katalon.com/reports/state-quality-2023> (Дата звернення: 03.11.2024).

12. Katalon. (2022). The State of Quality Report 2022. URL: <https://katalon.com/resources-center/blog/the-state-of-quality-report-2022?ref=dogq.io> (Дата звернення: 03.11.2024).
13. Kobiton. (2020). Test Automation 2020 Survey. URL: <https://info.kobiton.com/test-automation-2020-survey?ref=dogq.io> (Дата звернення: 12.11.2024).
14. LeetCode. (2023). LeetCode Platform Overview. URL: <https://leetcode.com> (Дата звернення: 07.11.2024).
15. PeerDH. (2023). Integrating Test Automation with Behavior-Driven Development in Scrum. URL: <https://peerdh.com/uk/blogs/programming-insights/integrating-test-automation-with-behavior-driven-development-in-scrum> (Дата звернення: 07.11.2024).
16. Replit. (2023). Replit Platform User Statistics. URL: <https://replit.com> (Дата звернення: 07.11.2024).
17. Ten10. (n.d.). How Much Are Software Bugs Costing You?. URL: <https://ten10.com/blog/how-much-are-software-bugs-costing-you/> (Дата звернення: 15.11.2024).

ДОДАТКИ

Додаток А

Лістинг 5. Клас “GitHubService” модуля HomeAssignment.Application.

```

public class GitHubService : IGitHubService
{
    private readonly IAssignmentService _assignmentService;
    private readonly IBranchService _branchService;
    private readonly ICommitService _commitService;
    private readonly IGitHubBuildService _gitHubBuildService;
    private readonly ILogger _logger;
    private readonly IStudentService _studentService;
    private readonly ITeacherService _teacherService;

    public GitHubService(
        IStudentService studentService,
        ILogger logger,
        IAssignmentService assignmentService,
        IBranchService branchService,
        ITeacherService teacherService,
        IGitHubBuildService gitHubBuildService,
        ICommitService commitService)
    {
        _studentService = studentService;
        _logger = logger;
        _assignmentService = assignmentService;
        _branchService = branchService;
        _teacherService = teacherService;
        _gitHubBuildService = gitHubBuildService;
        _commitService = commitService;
    }

    public async Task<IEnumerable<string>> GetStudentBranches(Guid
githubProfileId, Guid assignmentId,
    CancellationToken cancellationToken = default)
    {
        try
        {
            var assignment = await
_assignmentService.GetAssignmentByIdAsync(assignmentId, cancellationToken);
            if (assignment == null) return null;

            var student = await
_studentService.GetStudentByIdAsync(githubProfileId, cancellationToken);
            if (student == null) return null;
        }
    }
}

```

```
        var teacher = await
_teacherService.GetTeacherByIdAsync(githubProfileId, cancellationToken);
    if (teacher == null) return null;

    var branches = await _branchService.GetBranchesAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        cancellationToken);

    var studentBranches = await
_branchService.GetBranchesWithCommitsByAuthorAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        branches.Select(b => b["name"]?.ToString())!,
        student.GithubUsername,
        cancellationToken: cancellationToken
    );

    return studentBranches;
}
catch (Exception ex)
{
    _logger.Log($"Error getting GitHub branches: {ex.InnerException}.");
    throw new Exception("Error getting GitHub branches", ex);
}
}

public async Task<int> VerifyProjectCompilation(Guid githubProfileId, Guid
assignmentId, string branch,
 CancellationToken cancellationToken = default)
{
    try
    {
        var assignment = await
_assignmentService.GetAssignmentByIdAsync(assignmentId, cancellationToken);
        if (assignment?.CompilationSection == null) return 0;

        var student = await
_studentService.GetStudentByIdAsync(githubProfileId, cancellationToken);
        if (student == null) return 0;

        var teacher = await
_teacherService.GetTeacherByIdAsync(githubProfileId, cancellationToken);
        if (teacher == null) return 0;

        var lastCommitSha = await _commitService.GetLastCommitByAuthorAsync(
            teacher.GithubUsername,
            assignment.RepositoryName,
```

```

        branch,
        student.GithubUsername,
        cancellationToken: cancellationToken);

    if (string.IsNullOrEmpty(lastCommitSha))
    {
        _logger.Log("No commits found for the specified author.");
        return 0;
    }

    var isCompile = await _gitHubBuildService.CheckIfProjectCompilesAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        branch,
        lastCommitSha,
        cancellationToken
    );

    return isCompile ? assignment.CompilationSection.MaxScore :
        assignment.CompilationSection.MinScore;
}
catch (Exception ex)
{
    _logger.Log($"Error verifying project compilation:
{ex.InnerException}.");
    throw new Exception("Error verifying project compilation", ex);
}
}

public async Task<int> VerifyProjectQuality(Guid githubProfileId, Guid
assignmentId, string branch,
    CancellationToken cancellationToken = default)
{
    try
    {
        var assignment = await
_assignmentService.GetAssignmentByIdAsync(assignmentId, cancellationToken);
        if (assignment?.QualitySection == null) return 0;

        var student = await
_studentService.GetStudentByIdAsync(githubProfileId, cancellationToken);
        if (student == null) return 0;

        var teacher = await
_teacherService.GetTeacherByIdAsync(githubProfileId, cancellationToken);
        if (teacher == null) return 0;

        var lastCommitSha = await _commitService.GetLastCommitByAuthorAsync(

```

```

        teacher.GithubUsername,
        assignment.RepositoryName,
        branch,
        student.GithubUsername,
        cancellationToken: cancellationToken);

    if (string.IsNullOrEmpty(lastCommitSha))
    {
        _logger.Log("No commits found for the specified author.");
        return 0;
    }

    var percentage = await
_gitHubBuildService.EvaluateProjectCodeQualityAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        branch,
        lastCommitSha,
        cancellationToken
);

    var minScore = assignment.QualitySection.MinScore;
    var maxScore = assignment.QualitySection.MaxScore;

    if (maxScore == minScore) return minScore;

    var finalScore = minScore + percentage / 100.0 * (maxScore - minScore);
    return (int)Math.Round(finalScore);
}
catch (Exception ex)
{
    _logger.Log($"Error verifying project quality: {ex.InnerException}.");
    throw new Exception("Error verifying project quality", ex);
}

public async Task<int> VerifyProjectTests(Guid githubProfileId, Guid
assignmentId, string branch,
    CancellationToken cancellationToken = default)
{
    try
    {
        var assignment = await
_assignmentService.GetAssignmentByIdAsync(assignmentId, cancellationToken);
        if (assignment?.TestsSection == null) return 0;

        var student = await

```

```
_studentService.GetStudentByIdAsync(githubProfileId, cancellationToken);
    if (student == null) return 0;

    var teacher = await
_teacherService.GetTeacherByIdAsync(githubProfileId, cancellationToken);
    if (teacher == null) return 0;

    var lastCommitSha = await _commitService.GetLastCommitByAuthorAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        branch,
        student.GithubUsername,
        cancellationToken: cancellationToken);

    if (string.IsNullOrEmpty(lastCommitSha))
    {
        _logger.Log("No commits found for the specified author.");
        return 0;
    }

    var percentage = await
_gitHubBuildService.EvaluateProjectCodePassedTestsAsync(
        teacher.GithubUsername,
        assignment.RepositoryName,
        branch,
        lastCommitSha,
        cancellationToken
    );

    var minScore = assignment.TestsSection.MinScore;
    var maxScore = assignment.TestsSection.MaxScore;

    if (maxScore == minScore) return minScore;

    var finalScore = minScore + percentage / 100.0 * (maxScore - minScore);
    return (int)Math.Round(finalScore);
}
catch (Exception ex)
{
    _logger.Log($"Error verifying project tests: {ex.InnerException}.");
    throw new Exception("Error verifying project tests", ex);
}
}
```

Лістинг 6. Клас “DotNetTestsRunner” модуля HomeAssignment.Infrastructure.

```
public class DotNetTestsRunner : ITestsRunner
{
    private const string PassedPattern =
@"Passed\s+(<TestName>[\w\.]+)\s+\[(<Time>\d+(\.\d+)?\s+ms)\]";
    private const string FailedPattern =
@"Failed\s+(<TestName>[\w\.]+)\s+\[(<Time>\d+(\.\d+)?\s+ms)\]";

    private readonly ILogger _logger;

    public DotNetTestsRunner(ILogger logger)
    {
        _logger = logger;
    }

    public async Task<IEnumerable<TestResult>> RunTestsAsync(string repositoryPath,
CancellationToken cancellationToken)
    {
        var testsDirectory = Path.Combine(repositoryPath, "tests");
        var testFiles = Directory.GetFiles(testsDirectory, "*.*proj",
SearchOption.AllDirectories);
        var testResults = new List<TestResult>();

        foreach (var testFile in testFiles)
        {
            var processStartInfo = new ProcessStartInfo
            {
                FileName = "dotnet",
                Arguments = $"test \"{testFile}\" --verbosity normal",
                RedirectStandardOutput = true,
                UseShellExecute = false,
                CreateNoWindow = true
            };

            using var process = new Process();
            process.StartInfo = processStartInfo;

            process.OutputDataReceived += (_, args) =>
            {
                if (string.IsNullOrEmpty(args.Data)) return;

                if (Regex.IsMatch(args.Data, PassedPattern))
                {
                    var match = Regex.Match(args.Data, PassedPattern);
                    testResults.Add(new TestResult

```

```

    {
        TestName = match.Groups["TestName"].Value,
        IsPassed = true,
        ExecutionTimeMs =
ExtractExecutionTime(match.Groups["Time"].Value)
    });
}
else if (Regex.IsMatch(args.Data, FailedPattern))
{
    var match = Regex.Match(args.Data, FailedPattern);
    testResults.Add(new TestResult
    {
        TestName = match.Groups["TestName"].Value,
        IsPassed = false,
        ExecutionTimeMs =
ExtractExecutionTime(match.Groups["Time"].Value)
    });
}

try
{
    process.Start();
    process.BeginOutputReadLine();
    await process.WaitForExitAsync(cancellationToken);
}
catch (Exception ex)
{
    _logger.Log($"Error running tests for {testFile}: {ex.Message}");
}
}

return testResults;
}

private static double ExtractExecutionTime(string timeOutput)
{
    if (double.TryParse(timeOutput.Replace(" ms", ""), out var time)) return
time;
    return 0;
}
}

```