Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування. Частина 1 Сервер оцінювання робіт»

Виконав: студент 2-го курсу, групи 1ПІ-20м (д/н)
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Костюк К. А.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Войтко В. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«___» _____ 2021 р.

Опонент: к.т.н., проф. каф. КН Месюра В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«___» _____ 2021 р.

Допущено до захисту
Завідувач кафедрою ПЗ
д.т.н., проф., Романюк О. Н.
«___» _____ 2021 р.

ВНТУ – 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 Інформаційні системи
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма Інженерія програмного забезпечення

УЗГОДЖУЮ
Директор КЗ «ВТЛ»
_____ О. М. Козяр
"____" _____ 2021 р.

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
_____ О. Н. Романюк
"_13_" __вересня__ 2021 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Костюку Костянтину Андрійовичу

1. Тема роботи: Розробка веб-ресурсу для розміщення і автоматизованої перевірки олімпіадних задач. Частина 1 Серверна частина.

керівник роботи: к.т.н., доц. кафедри ПЗ Войтко В. В. затверджені наказом вищого навчального закладу від “_24_” __вересня__ 2021 року № 277

2. Строк подання студентом роботи “_1_” __грудня__ 2021 року

3. Вихідні дані до роботи: Середовище розробки – Microsoft Visual Studio 2019, Мова розробки – C#, C++, C++/CLI, Операційна система – Windows. Методи оцінювання часу роботи програм, створення Windows-орієнтованих сервісів, REST-протокол для комунікації між клієнтською та серверною частинами через HTTPS.

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити): вступ; аналіз стану питання розробки серверної частини та задач дослідження; розробка структури і алгоритмів роботи серверної частини; розробка програмного засобу серверної частини; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): мета, об'єкт та предмет дослідження; аналіз аналогів; схема роботи веб-ресурсу «Codelabs»; діаграма компонентів серверної частини; блок-схема алгоритму роботи Windows сервісу; блок-схема алгоритму роботи модуля «Tester»; тестування додатку.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–4	к.т.н., доц. Войтко В. В.		
5	к.т.н., доц. каф. ЕПВМ Ратушняк О.Г.		

7. Дата видачі завдання "14" вересня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістреської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	15.09 – 20.09	Вик.
2	Розробка архітектури та алгоритмів роботи системи	20.09 – 02.10	Вик.
3	Вибір середовища та мови розробки	02.10 – 10.10	Вик.
4	Розробка програмного продукту	10.10 – 25.10	Вик.
5	Тестування роботи системи	25.10 – 05.11	Вик.
6	Економічна частина	05.11 – 20.11	Вик.
7	Оформлення матеріалів до захисту МКР	20.11 – 30.11	Вик.

Студент _____ Костюк К.А.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи

_____ Войтко В. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.421+004.75:004.455

Костюк К. А. Розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування. Частина 1 Сервер оцінювання робіт. Магістерська кваліфікаційно робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – 121 Інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. 142 с.

На укр. мові. Бібліогр.: 33 назв; рис.: 81; табл. 9.

У магістерській кваліфікаційній роботі удосконалено метод підвищення безпеки процесу отримування і перевірки завдань на боці серверу за рахунок використання легковісних віртуальних машин в процесі обробки даних, що дозволяє зменшити робочі ресурси серверу та підвищити рівень безпеки робочого процесу. Також проведено роботу над удосконаленням оцінювання часу роботи програми, а саме запропоновано виконати фіксацію тактової частоти процесора й підрахунок часу виконання програми на основі затрачених процесорних тактів, а не реального часу роботи програми.

У результаті розроблено серверну частину веб-ресурсу, яка подана у вигляді Windows сервісу, що здійснює компіляцію та автоматизовану перевірку вихідних кодів користувачів ресурсу на заданому наборі тестів відповідної задачі. Серверна частина передає та приймає дані через спроектоване JSON-API, яке створено з використанням принципів REST від клієнтської сторони, що відповідає за відображення та систематизацію даних від серверу та користувачів. Створений програмний продукт написаний на мовах програмування C# та C++ під операційну систему Windows, керується за допомогою команд, отриманих від клієнтської частини.

Ключові слова: спортивне програмування, windows-сервіс, тестування, компіляція, віртуалізація, безпека.

ANNOTATION

The master's qualification thesis describes the improvement to the security of the downloading and verifying processes for competitive programming solutions on the server-side through the use of lightweight virtual machines, which reduces the usage of server resources and increases the overal security of the workflow. Work has also been done to improve the evaluation of a program run time, namely, it is proposed to fix the CPU clock speed and calculate the program execution time based on spent CPU clocks, rather than real program run time.

As a result, the server part of the web resource was developed, which is presented in the form of a Windows service that compiles and automatically checks the users' source code on a given set of tests for the task. The server part transmits and receives data through a JSON-API, which is designed using REST principles, from the client-side, which is responsible for displaying and organizing data from the server and users. The created software product is written in C# and C++ programming languages for the Windows operating system and is controlled by commands received from the client part.

Keywords: sports programming, windows service, testing, compilation, virtualization, security.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У наш час програмування займає важливу роль у суспільстві, оскільки велику частину роботи за нас виконують комп'ютери. У багатьох школах з'являється предмет програмування та інформатика, у той час, як у вищих навчальних закладах вони існують давно та активно розвиваються [1]. Для студентів та учнів важливо регулярно проводити оцінювання та підтримувати їх мотивацію. Найкращим форматом завдання з програмування є написання власної програми, тому виникає необхідність перевіряти велику кількості програм. Але така перевірка може зайняти багато часу. Для цього існують системи автоматичної перевірки, проте їх не так багато і не завжди їх можна використати в навчальному процесі. Створення системи автоматичної перевірки задач з можливістю додання власних задач прийде на допомогу викладачу, та допоможе пришвидшити процес навчання.

Разом з цим кожного року проводиться десятки різноманітних турнірів та олімпіад різного рівня від місцевого до міжнародного, куди запрошуються усі бажаючі. У сфері інформатики розповсюдженими є олімпіади зі спортивного програмування, де потрібно створити програму, що буде розв’язувати певну проблему. Такі програми перевіряються у реальному часі під час проведення змагань.

Перевірка програми складається з декількох етапів таких як, компіляція вихідного коду та власне запуск розв’язку на певному наборі тестів для перевірки коректності його роботи. В залежності від типу олімпіади, перевірка може відбуватися, як на повному наборі тестів так і на деякому попередньому наборі, що містить лише базовий набір тестів. Крім правильності відповіді, оцінюється такі параметри роботи програми, як час витрачений на отримання відповіді та пам’ять використана програмою під час роботи.

Оскільки усі програми виконуються на деякому сервері паралельно, вони не повинні перешкоджати роботі одна одній та усій системі в цілому. Також не

можна допустити отримання правильних відповідей будь-яким шляхом, окрім прямого розв’язування задачі. Тому програми користувачів потрібно запускати в ізольованому середовищі та обмежити їх доступ до таких системних ресурсів, як мережа Інтернет чи файлова система операційної системи, де проводиться тестування.

Існують різні подібні системи для тренування і оцінювання робіт зі спортивного програмування, такі як E-olymp, Ejudge і Algotester. Однак вони мають певні недоліки та обмеження. Наприклад, E-olymp та Algotester не дозволяє переглядати детальний результат по кожному тесту [2; 3], Ejudge потребує складного встановлення та налаштування на власному сервері [4].

Отже, актуальною є розробка нової системи перевірки програм-розв’язків до задач зі спортивного програмування, яку так само можна використовувати під час навчання, як і під час олімпіад.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості систем автоматичної перевірки програм розв’язків задач зі спортивного програмування та оцінювання параметрів їх роботи за рахунок розробки та програмної реалізації спеціалізованого програмного забезпечення для веб-ресурсу, зокрема серверної частини, що дозволить підвищити надійність перевірки та оцінювання робіт.

Основними задачами дослідження є:

– розробка загальної моделі роботи серверної частини;

– розробка алгоритму роботи Windows сервісу;

– розробка алгоритму роботи модуля безпечного тестування «Tester»;

– розробка методу покращення точності оцінки часу тестування роботи програми;

– удосконалення методу перевірки завдань на боці серверу, який використовує легковісні віртуальні машини для підвищення безпеки;

– розробка інтерфейсу конфігурації сервісу;

– тестування програмного продукту.

Об’єкт дослідження – процес автоматичної перевірки розв’язків задач зі спортивного програмування, безпечної компіляції та запуску програм і точного оцінювання параметрів їх роботи.

Предмет дослідження – методи та засоби розробки серверної частини веб-ресурсу для автоматичної перевірки програм та оцінювання параметрів їх роботи.

Методи дослідження. У процесі дослідження використовувались:

– методи розробки Windows сервісу для розробки серверної частини програми та модулю безпечного тестування;

– методи математичного моделювання для перевірки коректного фіксування часу виконання програм;

– методи математичного аналізу для розробки методу аналізу часу роботи програми за рахунок підрахунку затрачених тактів роботи процесора;

– методи об’єктно-орієнтованого програмування та шаблони програмування для розробки архітектури класів та сервісів програмного забезпечення.

Наукова новизна отриманих результатів.

1. Подальшого розвитку дістав метод підвищення безпеки процесу отримування і перевірки завдань на боці серверу, який, на відміну від існуючих, використовує легковісні віртуальні машини в процесі обробки даних, що дозволяє зменшити робочі ресурси серверу та підвищити рівень безпеки робочого процесу.

2. Подальшого розвитку отримала модель оцінки часу роботи програми, що, на відміну від існуючих, орієнтована на використання віртуальних машин для обробки інформації і передбачає фіксацію тактової частоти процесора й підрахунок часу виконання програми на основі затрачених процесорних тактів, а не реального часу роботи програми, що дозволяє збільшити ресурсні

можливості сервера й підвищити стабільність і точність результатів оцінювання.

Практична цінність отриманих результатів. Розробленні алгоритми та програмні модулі серверної частини веб-ресурсу можуть бути використані для оцінювання результатів вирішення конкурсних задач спортивного програмування. Програмні засоби можуть використовуватися для проведення олімпіадних змагань і тренувань до них у дистанційній формі в режимі реального часу.

Впровадження. Результати досліджень використовуються у Комунальному закладі «Вінницький технічний ліцей» Вінницької міської ради, що підтверджується відповідним актом впровадження (додаток Б).

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать алгоритми роботи, програмна реалізація сервісу для автоматизованої перевірки задач зі спортивного програмування [5] та розвиток методу оцінювання часових параметрів роботи програми через підрахунок кількості затрачених на її роботу тактів процесора [6] і метод підвищення безпеки процесу отримування і перевірки завдань на боці серверу, який використовує легковісні віртуальні машини.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на:

- XLIX Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії Вінницького національного технічного університету (Вінниця, 2020).
- Міжнародній конференції «Молодь у світі сучасних технологій» (Херсон, 2020).

- Всеукраїнській науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2021).

Результати роботи доповідалися на Всеукраїнському конкурсі студентських наукових робіт зі спеціальності «Інженерія програмного забезпечення» й було отримано диплом переможця 2 ступеня (Тернопіль, 2021).

Публікації. Результати роботи опубліковані в трьох наукових працях – тезах-доповідях на конференціях: XLIX Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії Вінницького національного технічного університету (Вінниця, 2020) [2], Міжнародній конференції «Молодь у світі сучасних технологій» (Херсон, 2020) [3], Всеукраїнській науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2021) [5].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу; п’яти розділів; висновку; списку літератури, що містить 33 найменуваня, 5 додатків. Робота містить 81 ілюстрацію, 9 таблиць.

1 АНАЛІЗ СТАНУ ПИТАННЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ТА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання розробки серверної частини веб-ресурсу

Навчальний процес є важливою складовою життя кожного. Для студентів та учнів важливо регулярно проводити оцінювання та підтримувати їх мотивацію. З цією метою проводять різні змагання, конкурси, турніри, олімпіади, хакатони та інше по багатьом дисциплінам. Такі інтелектуальні змагання проводяться, щоб виявити й розвинути творчі здібності, підвищити інтерес до науково-дослідницької діяльності, сприяти професійній орієнтації, та мотивувати поглиблене вивчення предмету.

Найбільшої популярності такий підхід набув у сфері програмної інженерії та спортивного програмування. Кожного року проводиться десятки різноманітних турнірів різного рівня від місцевого до міжнародного, куди запрошуються усі бажаючі. Під час проведення таких заходів важливим є швидкість і об’єктивність оцінювання, а також можливість дистанційного проведення з залученням декількох майданчиків. У ручному режимі такі вимоги забезпечити достатньо важко, тому у роботу вступають автоматизовані системи, що дають змогу не тільки надати інформацію під час змагання, а й швидко зібрати та перевірити рішення учасників з різних куточків світу. Деякі з цих ресурсів також дають змогу тренуватися до змагань, або ж проводити власні змагання.

Такі платформи зазвичай представлені у вигляді веб-ресурсів, поєднуючи швидкість і надійність роботи з сучасним інтерфейсом та доступністю будь-кому. Користувачі, що мають доступ до мережі Інтернет, отримують можливість зареєструватися на сайті для олімпіади, тренуватися з дому в зручний для них час, що є особливо актуальним у сучасному світі.

Задачі зі спортивного програмування передбачають написання власної програми, яку необхідно скомпілювати, а потім запустити, надавши вхідні дані, та перевірити результат її роботи. Тому такі веб-ресурси розділені на декілька

складових, таких як клієнтська та серверна частини. Перша доступна користувачам та надає загальний інтерфейс комунікації з ресурсом. Серверна частина призначена для перевірки рішень задач користувачів. Вона виконується в захищеному сервері та з повним контролем за ходом роботи програми, з метою виявлення можливостей шахрування. Саме від серверної частини залежить точність кінцевої оцінки, що складається з таких частин, як час затрачений на роботу розв'язку, правильність отриманого результату та інші. Серверна частина повинна використовувати мінімум ресурсів, щоб не вносити неточності в роботу програм розв'язків, зменшити похибки вимірювання часу.

Для перевірки рішень користувачів до кожної задачі створюють спеціальні тести відповідно до її умови. Тести містять вхідні дані для запуску розв'язку та очікуваний результат у вигляді файлу, або програми, що виконає генерування вхідних даних за певним алгоритмом. Якісні тести повинні відповідати технічній умові задачі, та покривати усі особливі випадки, що можуть бути у алгоритмі розв'язку, щоб виявляти рішення, які отримують правильну відповідь, проте використовують багато пам'яті або працюють занадто довго. Власне перевірка виконується спеціальною програмою, яку називають «чекер»[4]. На вхід такій програмі надається тестовий набір та результати роботи програми-розв'язку. Програма аналізує усі передані їй дані й видає вердикт.

Комп'ютерна безпека – це сучасна широка тема, повна складних проблем. Однак, як це зазвичай буває з великими, складними темами, обмеження проблемного простору дозволить легше побачити основні проблеми і розробити життєздатні рішення[9].

Розглянемо сценарій підготовки студентів до спортивного програмування та проведення змагань. Студентам необхідно написати програми, які виконують деякий алгоритм, і завантажити їх на сервер. Сервер компілює код та запускає програми, подані студентами, і порівнює результати їх роботи.

Серверна частина також повинна мати надійних захист від зовнішнього доступу та виконувати рішення в ізольованому середовищі, щоб не пошкодити

власну інфраструктура шкідливим кодом та тримати у безпеці код та результати інших користувачів системи.

Отже, постає питання, як можна запустити вихідний код, надісланий студентом, щоб перевірити його, але не нашкодити серверу. Одним із варіантів є використання технологій віртуалізації для запуску скомпільованої програми у ізольованому середовищі.

1.2 Порівняльний аналіз аналогів

Зараз існує багато систем для розміщення та автоматичної перевірки вихідних кодів розв’язків задач на певному наборі тестів. Тести можуть бути як чітко заданими так і генерованими в режимі онлайн. Більшість систем має закриті вихідні коди серверних частин, тому порівняння можливе лише за непрямими ознаками. Розглянемо найбільш популярні ресурси.

Ejudge – система для централізованого та автоматизованого проведення змагань та інших заходів, в яких необхідна автоматична перевірка вихідних кодів (рисунок 1.1). Система дозволяє проводити турнір по одній з чотирьох груп правил – ACM, Kirow, Moscow, Olympiad.

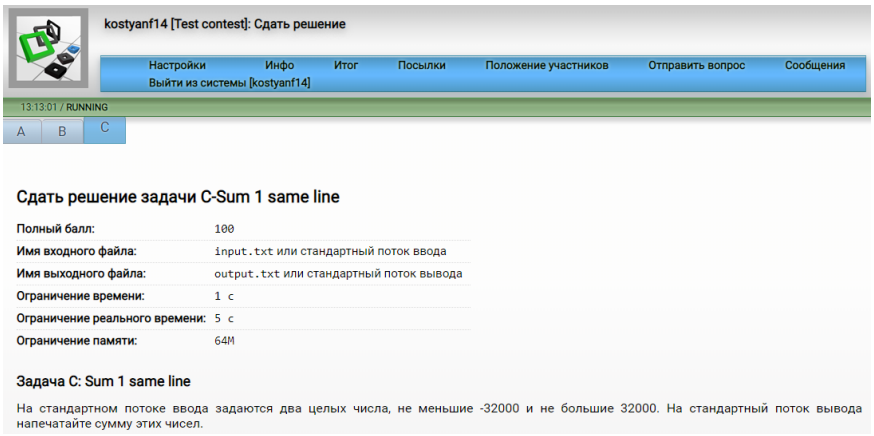


Рисунок 1.1 – Зовнішній вигляд системи Ejudge

Дана система має відкриті вихідні коди та дає можливість змінювати її вигляд згідно з поставленими задачами. Вона розрахована на OS Linux, проте може бути скомпільованою під Windows з обмеженими можливостями [5]. Для

її використання необхідний виділений сервер. У базовому режимі Ejudge не має високого рівню ізоляції виконуваних програм. Для повноцінної ізоляції необхідно встановлення патчу у ядро Linux, що є досить складною операцією. Крім того, це може створити інші проблеми на рівні ядра.

E-Olymp – інтернет-портал організаційно-методичного забезпечення дистанційних олімпіад з програмування для обдарованої молоді навчальних закладів України. Портал розроблено на базі Житомирського державного університету імені І. Франка. Система має закриті вихідні коди, проте після аналізу [6] сторінки «Допомога», можна зробити висновки, що портал використовує OS Linux. На сайті проводяться змагання за двома основними групами правил - ACM, Kirow.

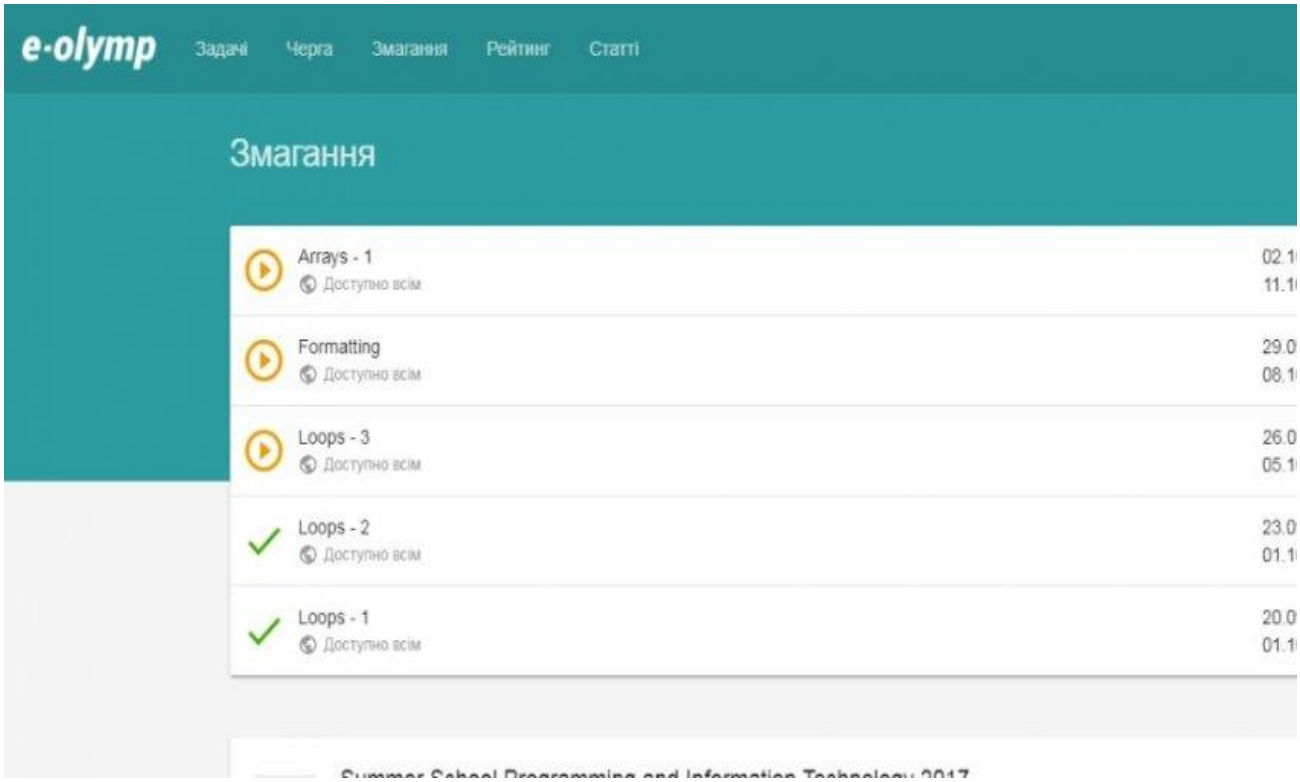


Рисунок 1.2 – Зовнішній вигляд системи E-Olymp

Algotester – система автоматичного тестування розв'язків з великим набором цікавих алгоритмічних задач різноманітної складності, ресурс для створення та публікації власних задач з можливістю проведення змагань за правилами ACM та Olympiad[7].

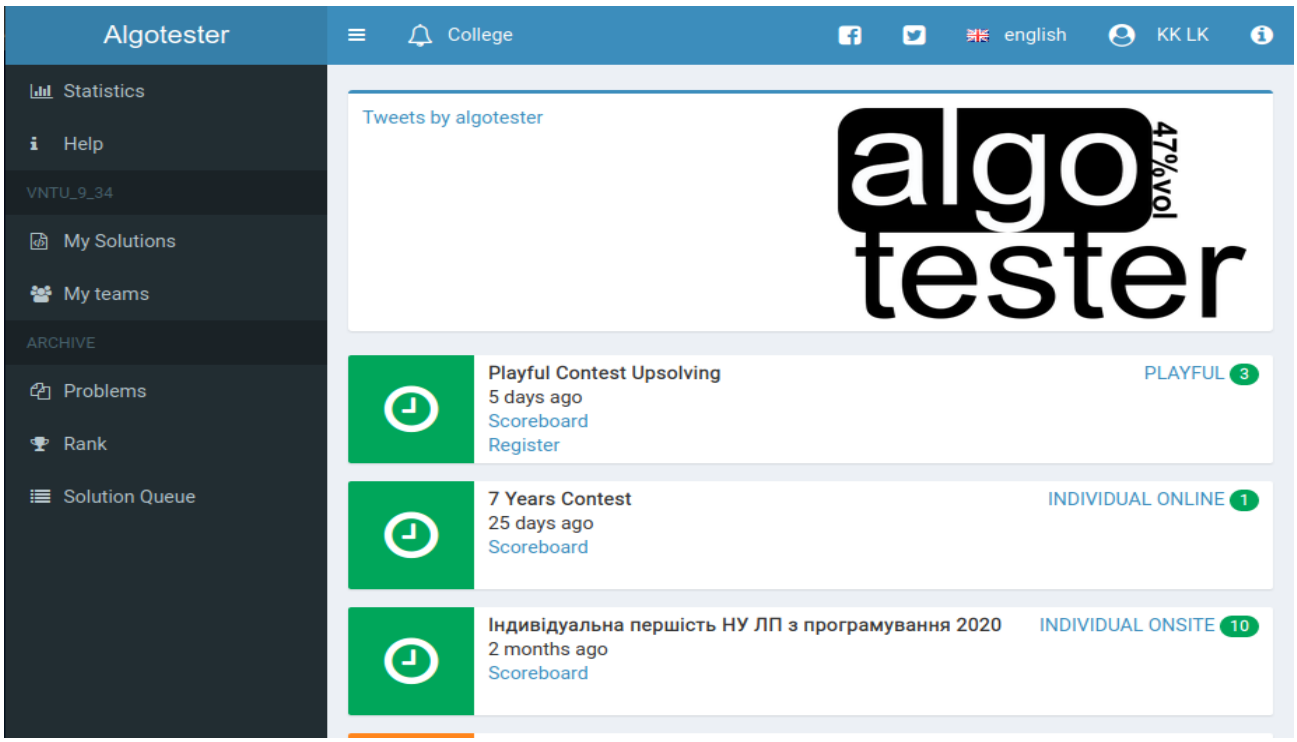


Рисунок 1.3 – Зовнішній вигляд системи Algotester

Проаналізувавши усі аналоги, визначено їх можливості та недоліки, сильні та слабкі сторони, які враховувались при створенні власного веб-ресурсу «Codelabs» (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Ejudge	E-Olymp	Algotester	Codelabs
Можливість задання різних обмежень по часу та пам’яті для різних мов програмування	-	-	-	+
Детальний результат по кожному тесту	+	-	-	+
Точний розрахунок часу роботи програми	+/-	-	-	+
Багатопоточність компіляції/запуску рішень на перевірку	+	+	+	+
Можливість по-тестової компіляції	+	+	+	-
Можливість запуску на власних серверах	+	-	-	+
Безпечний запуск на Windows	-	-	-	+

Таблиця порівняльних характеристик показала, що розробка власної системи є доцільною. В результаті нове рішення покриває недоліки існуючих аналогів та забезпечує більш точний розрахунок часу, а як результат правильність оцінювання, разом з забезпеченням безпечної роботи.

1.3 Постановка задач розробки

Після аналізу актуальності стану питання тестування розв’язків до олімпіадних задач з програмування та порівняння існуючих аналогів за певним переліком критеріїв було визначено завдання, які необхідно виконати для розробки серверної частини системи «Codelabs»:

– розробити загальну модель роботи серверної частини;

– розробити алгоритм роботи Windows сервісу;

– розробити алгоритм роботи модуля безпечного тестування «Tester»;

– розробити метод покращення точності оцінки часу тестування роботи програми;

– удосконалити метод перевірки завдань на боці серверу, який використовуватиме легковісні віртуальні машини для підвищення безпеки;

– розробити інтерфейс конфігурації сервісу;

– провести тестування програмного продукту.

1.4 Висновки

У першому розділі було розглянуто актуальність питання автоматизованої перевірки задач з програмування під час турнірів чи олімпіад, та з метою покращення навичок, набутих студентами під час навчального процесу.

Також було проаналізовано можливі підходи до вирішення даного питання шляхом порівняння систем-аналогів, окреслення їх сильних та слабких сторін, таких як безпечний запуск програм користувачів, проведення паралельного тестування, точність визначення результатів і можливість задання коефіцієнтів для різних мов програмування з метою покращення оцінювання.

Серед аналогів було виділено такі програмні продукти, як Ejudge, E-Olymp, Algotester. Основними недоліками серверної частини розглянутих аналогів, можна визначити відсутність детального результату по кожному тесту, що є незручним для користувачів під час тренування, не точний розрахунок часу роботи програми та відсутність можливості запуску системи на ОС Windows. Тому було вирішено розробити власну серверну частину, яка б мала необхідний функціонал.

З огляду на недоліки аналогів було визначено такі основні задачі для розробки, як розробка методу безпечного запуску програм та управління процесом розв’язку, розробка способу оцінювання використаних часу та пам’яті та розробка Windows сервісу з модулями програми для компіляції, запуску та оцінювання результатів роботи.

У результаті порівняння було відображено необхідність розробки магістерської кваліфікаційної роботи та сформульовано задачі, які необхідно розв’язати для вирішення питання.

2 РОЗРОБКА МЕТОДУ І МОДЕЛІ РОБОТИ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Аналіз інформаційного забезпечення системи

Інформаційне забезпечення – це набір документів, нормативної бази і реалізованих рішень, які стосуються обсягу, розміщення і методів впорядкування інформації, яка знаходиться в інформаційній системі, що автоматизує обробку певних даних.

Основні вимоги, що повинні бути дотримані при створенні інформаційного забезпечення: цілісність, достовірність, контроль, захист від несанкціонованого доступу, єдність і гнучкість, стандартизація та уніфікація, адаптивність, мінімальна кількість помилок введення-виведення інформації [8].

Для забезпечення безпечного запуску скомпільованих програм рішень задач, необхідно запускати програму в обмеженому середовищі «sandbox», що буде повністю контролювати усі ресурси та дії виконувані програмою. Для цього оптимально використати наявні у системі Windows можливості по контролю над додатками через Windows API за допомогою функцій SetTokenIntegrityLevel та UpdateProcThreadAttribute. Перша функція накладає загальні обмеження на завантажувальний додаток, а друга – дає можливість обмежити ресурси для основного потоку програми.

Серверна частина веб-ресурсу «Codelabs» буде отримувати вхідні дані, такі як вихідні коди рішень задач, тести до них та обмеження для перевірки, через API клієнтської частини. Передачу даних необхідно здійснювати у машино-читаємому форматі, такому як JSON, XML або Protocol Buffers [9]. Для забезпечення безпеки передачі даних до серверної частини потрібно використовувати шифрування транспортного потоку та авторизацію по каналу зв’язку [10].

JSON формат передачі даних є одним з стандартних форматів, що використовується при розробці веб-орієнтованих додатків. Він є простим у розумінні для людини і дозволяє легко редагувати данні при необхідності. Може бути опрацьованим без детальних знань структури об’єктів, що

передаються. Також він має чудову підтримку зі сторони браузера та менші розміри ніж XML.

XML формат часто застосовується у Windows-орієнтованих додатках. Він є зручним для опису структур інтерфейсу користувача, проте може бути використаний для опису та передачі даних. XML має більші розміри у порівнянні з JSON форматом. Так само як і JSON він є простим для розуміння людиною і може бути легко опрацьованим. Часто застосовується при використанні Simple Object Access Protocol при передачі даних.

Protobuf новий формат даних, що розроблено у компанії Google. Згідно з нею, protocol buffers в декілька раз збільшує швидкість обробки даних та суттєво зменшує обсяги передаваної інформації. Однак уся інформація передається у бінарному вигляді, що унеможливлює її опрацювання людиною без додаткових перетворень. Для опрацювання необхідно підтримувати точність схем об’єктів між клієнтською та серверною частинами.

Стандартним підходом до шифрування даних у веб-ресурсах є HTTPS. Він має декілька стандартів шифрування TLS. HTTPS - це стандартний HTTP трафік, що передається через SSL або TLS тунель, більшість складових пакету даних шифруються: URL-запити, включаючи шлях та назву ресурсу, параметри запиту, заголовки. Шифрування гарантує захист від аналізу трафіку та від атак типу «man-in-the-middle», за умови коректних налаштувань та підпису сертифікату довіреним центром видачі сертифікатів. Раніше усі HTTPS сертифікати були платними, проте зараз зараз існує сервіс Let’s Encrypt від компанії Mozilla.

SSL – криптографічний протокол, який використовується в процесі створення з’єднання між клієнтом і сервером, для забезпечення його безпеки. Початкова версія протоколу була розроблена компанією Netscape Communications у 1995 році, а нова версія SSL 3.0, була включена у стандарт RFC та перейменована у TLS.

TLS – криптографічний транспортний протокол для безпечної передачі даних в глобальній мережі Інтернет, який застосовується для навігації,

отримання пошти, спілкування, обміну файлами, тощо. У сучасній версії протоколу TLS 1.3 використовується асиметричне шифрування і сертифікати рівня X.509. Процес комунікації складається з таких етапів:

– «рукостискання», під час якого відбувається вибір версії протоколу, перевірка сертифікатів (клієнта та сервера);

– узгодження спільного (сеансового) ключа на основі одного з алгоритмів (наприклад Діффі-Геллмана [12], RSA);

– обмін повідомленнями, зашифрованими сесійним ключем.

Віртуалізація – це процес створення віртуального, тобто штучного, об'єкта чи середовища. Віртуалізація може відбуватися на рівні операційної системи. Тут розглядають метод віртуалізації, при якому ядро операційної системи підтримує не один, а декілька ізольованих примірників простору користувача. Ці примірники, які ще називають «контейнерами» або «зонами», з точки зору користувача є ідентичними реальному серверові. Ядро повністю забезпечує ізольованість контейнерів. Тоді програми з різних контейнерів працюють автономно і не можуть впливати одна на одну.

Основний недолік такого підходу – це накладання витрат системних ресурсів, таких як оперативна пам’ять та процесорний час, які необхідні для запуску віртуального простору, операційної системи у ньому та для передачі даних до цільової програми.

Для усунення цього недоліку розглянемо можливість віртуалізації окремих процесів за допомогою легковісних віртуальних машин та бібліотеки libkrun.

Отже, на основі проведеного аналізу, було обрано JSON протокол для передачі даних між клієнтом та сервером через HTTPS протокол з шифруванням TLS 1.3, біліотеку libkrun для запуску легковісних віртуальних машин.

2.2 Розробка загальної моделі системи

Загальна модель роботи системи зображена на рисунку 2.1.

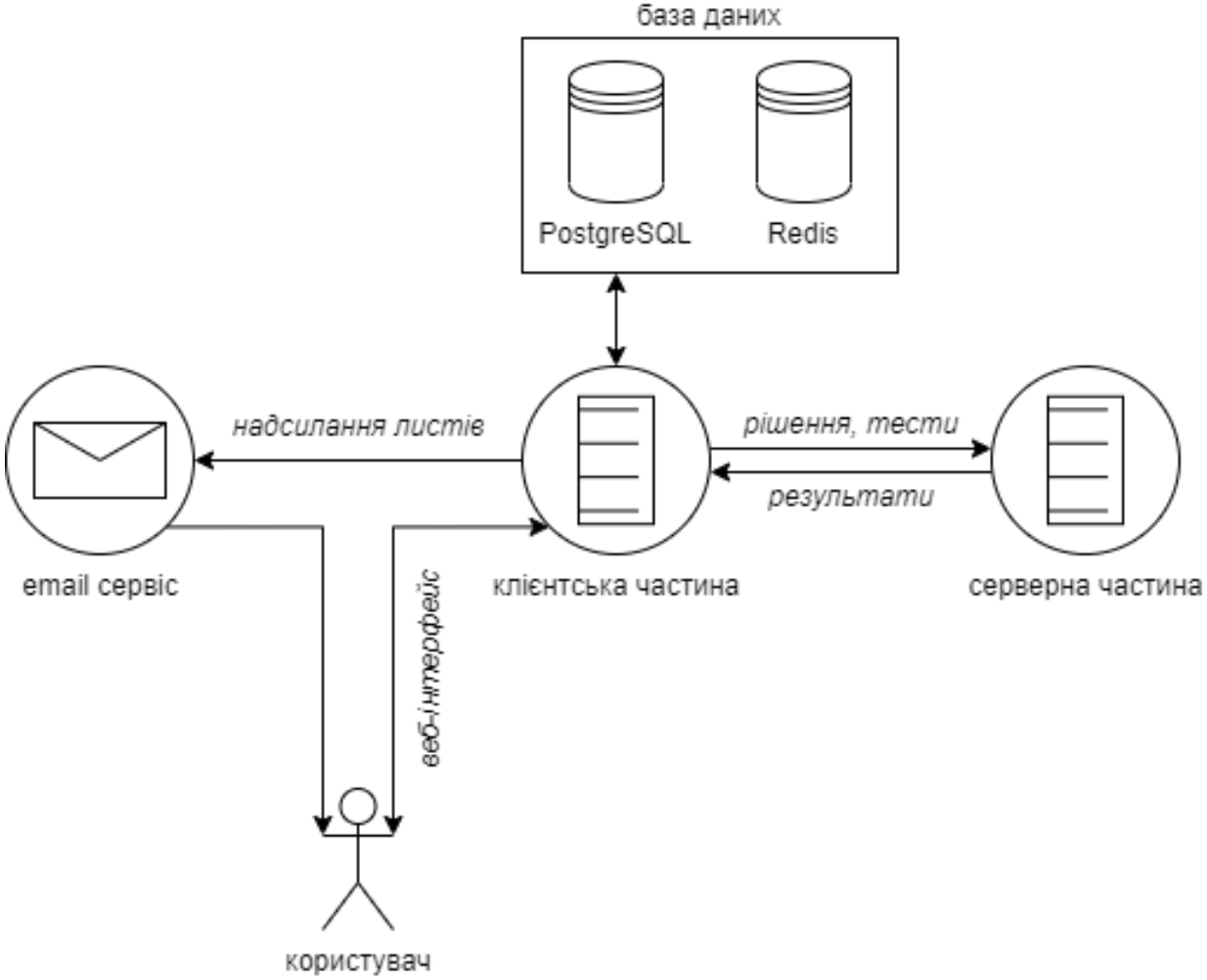


Рисунок 2.1 – Загальна модель роботи веб-ресурсу «Codelabs» [2]

Серверна частина представлена у вигляді Windows сервісу, що управляється командами отриманими з клієнтської сторони. Оскільки програма управляється адміністратором системи, то достатньо мати мінімальний користувацький інтерфейс та проводити логування дій, що виконує програма. Для запуску та зупинки програми використовується стандартне вікно сервісів Windows, а для логування доступно 5 рівнів, такі як: Debug, Info, Warning, Error, Fatal.

Для того, щоб розроблюваний сервіс для тестування рішень задач зі спортивного програмування був максимально корисним і зручним у використанні, він повинен правильно оцінювати користувацькі рішення. Це включає в себе як перевірку правильності роботи алгоритму, так і точність визначення об’єму використаної пам’яті програми та асимптотику її роботи.

Також варто надавати вичерпну інформацію стосовно тестування користувацького рішення: має бути лог компілятора, та інформація від чекера про отриманий результат та очікуваний результат.

Для виконання вище вказаних функцій мають бути створені такі модулі: HttpsCodelabsApiClient, WorkerTaskManager, Logger, ProblemCache, Tester, ConfigReader. Модель взаємодії модулів наведено на рисунку 2.2. Детальний опис кожного з модулів наведений нижче.

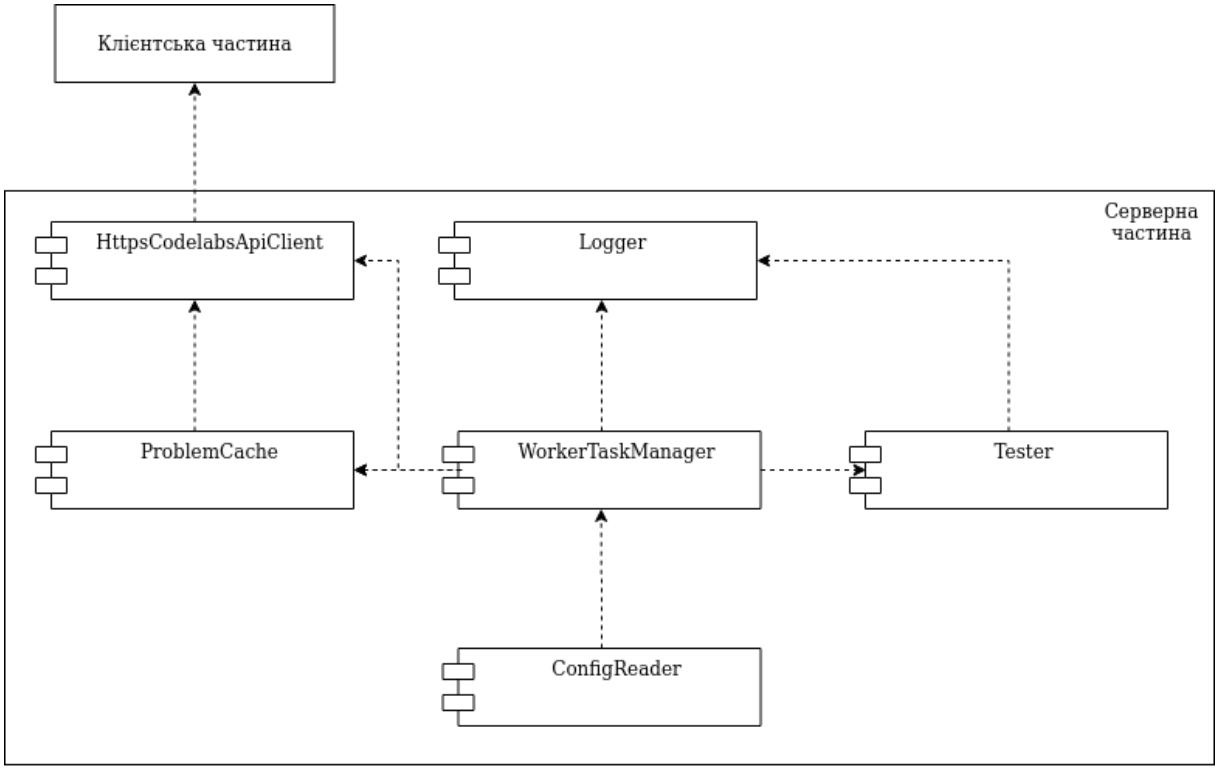


Рисунок 2.2 – Модель взаємодії компонентів серверної частини

WorkerTaskManager – головний модуль програми, що реалізовує основну бізнес-логіку роботи, відправляє команди на компіляцію та тестування розв’язку, надсилає результати до модулю комунікації.

HttpsCodelabsApiClient – модуль комунікації з клієнтською частиною за допомогою JSON-API.

ProblemCache – модуль, що зберігає дані про задачу, які можна використати для повторного тестування, такі як обмеження за часом та пам’яттю, набори тестових пар та вихідний код чекера.

Tester – модуль, що виконаний у вигляді окремої бібліотеки на мові C++, та виконує безпечний запуск програм на сервері з контролем ресурсів, що використовує Windows API.

ConfigReader – модуль для отримання даних з файлу конфігурації, в якому зберігаються налаштування, пов’язані з методами запуску рішень, адреса серверу API клієнтської частини, команди компіляторів, шлях для збереження логів.

Logger – модуль, який виконує логування усіх операцій, що відбуваються в сервісі, та надає callback-функції для виклику логування з інших модулів, у тому числі, написаних на C++.

Усі модулі, окрім модуля «Tester», виконані на мові C#.

2.3 Розробка алгоритму роботи Windows сервісу

Для роботи сервісу «Codelabs» необхідно запрограмувати наступні функції: завантаження та обробка файлу конфігурації, ініціалізація модулів компіляції, безпечного запуску скомпільованого файлу та модулю комунікації з клієнтською частиною, обробка повідомлень від менеджера сервісів Windows, обробка команд від клієнтської частини.

Було розроблено основний алгоритм сервісу, що взаємодіє з усіма модулями системи (рисунок 2.3), та алгоритм роботи модуля «Tester» (рисунок2.4).

Основний алгоритм Windows сервісу складається з наступних кроків:

Крок 1. Початок.

Крок 2. Читання та завантаження конфігурації з відповідного файлу.

Крок 3. Запуск усіх модулів програми та їх початкове налаштування.

Крок 4. Нескінченний цикл. Цей цикл буде перервано при отриманні команди на вихід.

Крок 5. Читання команди від менеджеру сервісів Windows.

Крок 6. Перевірка команди на рівність «вихід». Якщо перевірка успішна – перервати виконання циклу (Крок 4) та перейти до кроку 14.

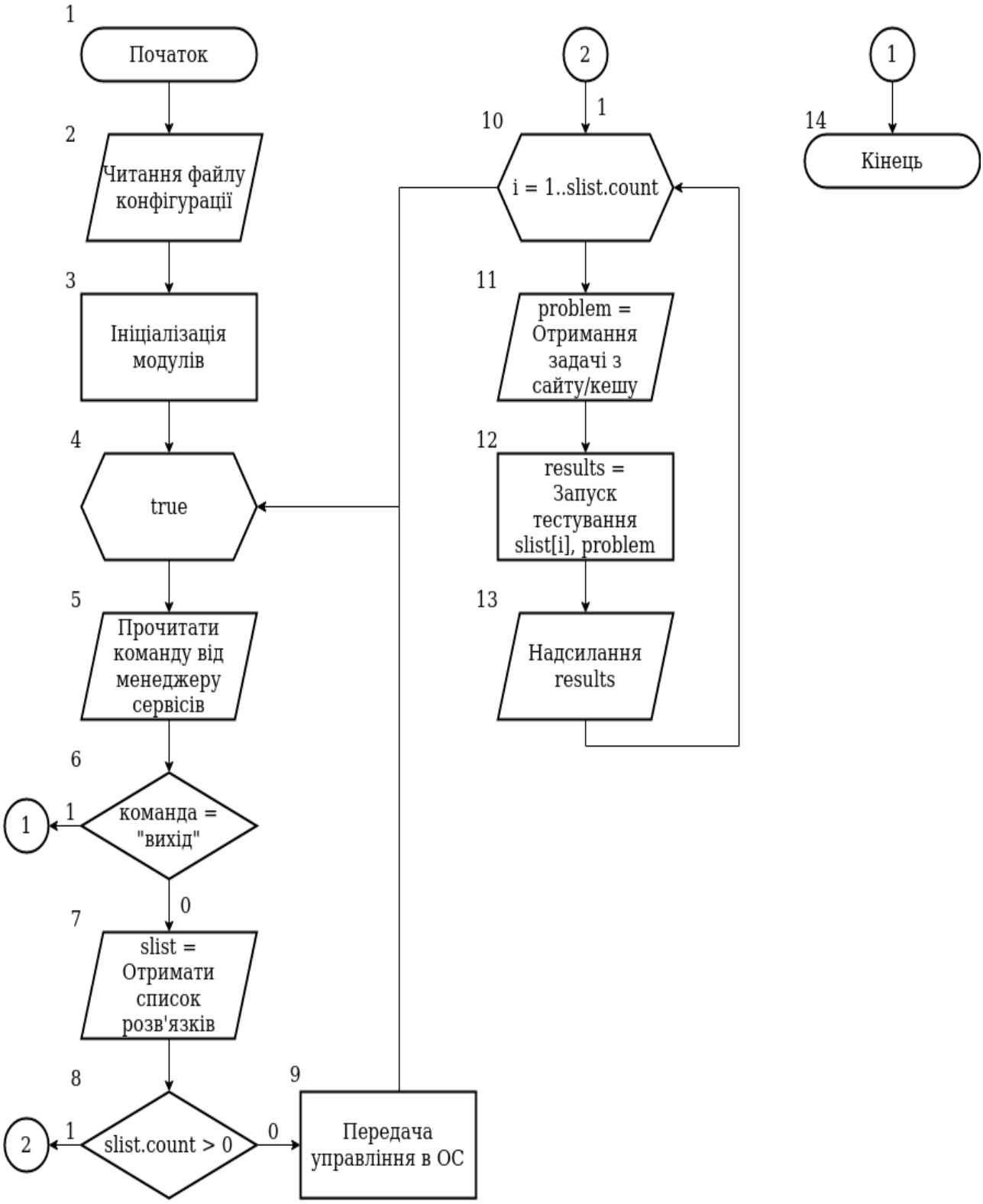


Рисунок 2.3 – Блок-схема алгоритму роботи Windows сервісу

Крок 7. Отримати список доступних для тестування розв’язків від клієнтської частини.

Крок 8. Перевірка розміру списку на значення більше нуля. Якщо перевірка успішна – перейти до кроку 10.

Крок 9. Передати управління до операційної системи для очікування наступної ітерації циклу. Пауза необхідна, щоб зменшити навантаження на сервер та клієнтську частину.

Крок 10. Цикл по списку розв’язків. Після циклу перейти до кроку 4.

Крок 11. Отримати інформацію про задачу з кешу, а в разі відсутності у кеші завантажити з клієнтської частини.

Крок 12. Скомпілювати та запустити тестування поточного розв’язку відповідно до інформації про задачу та записати результати до змінної results.

Крок 13. Надіслати результати перевірки (results) до клієнтської частини.

Крок 14. Кінець.

У випадку помилки, що може трапитися на будь-якому кроці алгоритму, вона буде записана до файлу логу з усією інформацією, яка допоможе усунути помилку. Також в файлі конфігурації є можливість налаштувати ступінь детальності логування. В випадку найдетальнішого звітування кожен логічний етап програми буде супроводжуватись відповідним записом у лог файлі.

Керування програмою виконується через менеджер сервісів, що дозволяє запустити сервіс, зупинити його або перезавантажити.

2.4 Розробка алгоритму роботи модуля «Tester»

Модуль «Tester» є важливою складовою роботи програми і призначений для тестування розв’язку користувача. Він виконує функції компілювання вихідного коду чекера (програми для перевірки результатів роботи розв’язку) та вихідного коду розв’язку, безпечного запуску цих програм, а також зберігає результатів перевірки і роботи чекера, щоб передати їх у основний модуль.

У зв’язку з необхідністю виклику системних функцій Windows API для створення замкнутого середовища роботи програми, у якому вона не зможе вплинути на роботу інших компонентів системи, модуль виконано на мові C++.

Для виклику функцій з модулю використовується інтерфейс взаємодії C++/CLI, що дає можливість викликати native-функції з мови високого рівня C#[13].

Алгоритм роботи модуля наведено на рисунку 2.4

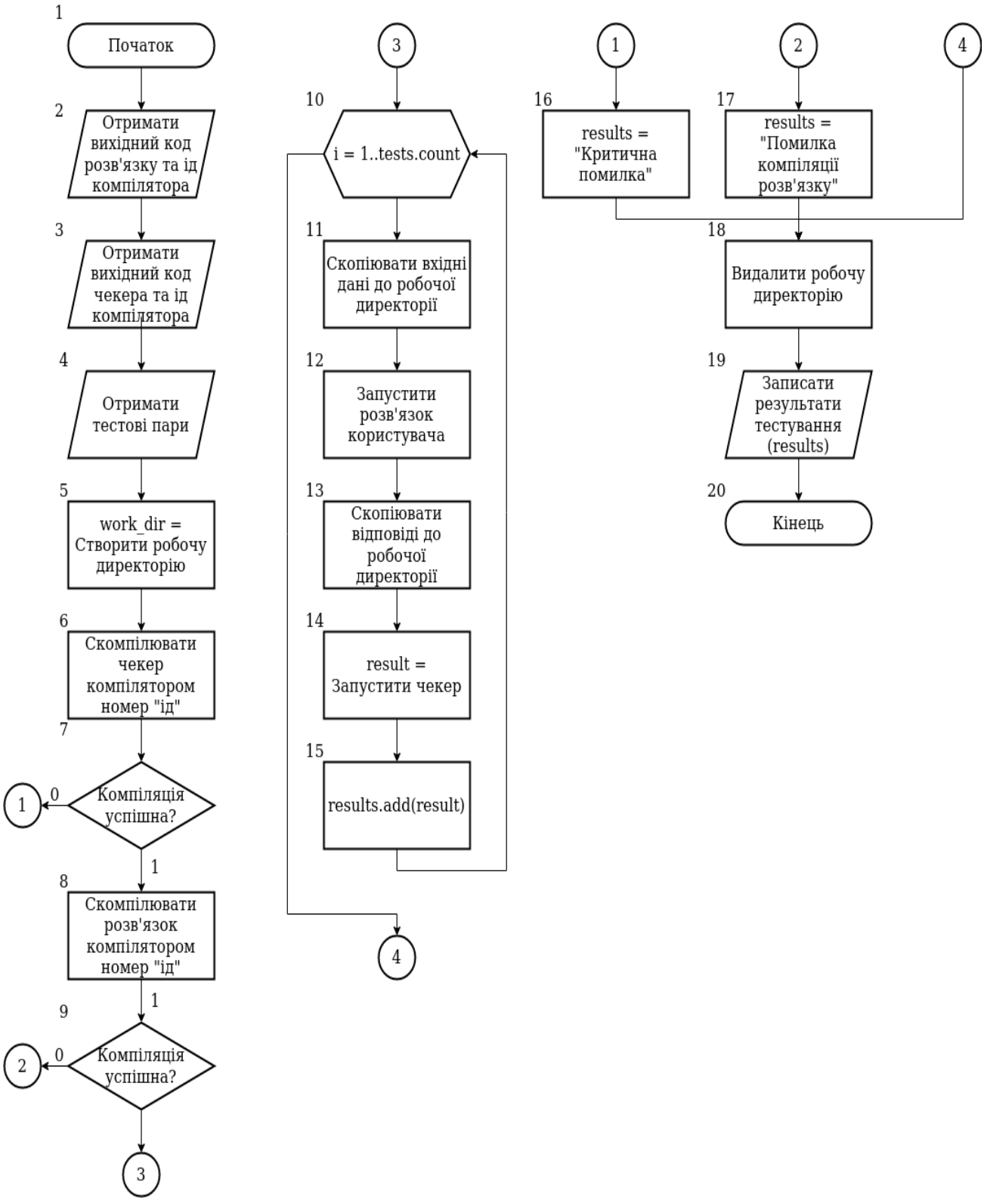


Рисунок 2.4 – Алгоритм алгоритму роботи модуля «Tester»

Цей алгоритм складається з наступних кроків:

Крок 1. Початок

Крок 2. Отримати вихідний код розв’язку задачі та номер компілятора обраного користувачем з основного модулю.

Крок 3. Отримати вихідний код чекера та номер його компілятора з основного модулю.

Крок 4. Отримати тестові пари (вхідні дані, відповіді) для поточної задачі з основного модулю.

Крок 5. Створити тимчасову робочу директорію, яка буде тимчасово зберігати вхідні дані, відповіді, результати роботи програми користувача, логи компіляції чекера та розв’язку, а також результати роботи черека по кожному тесту.

Крок 6. Скомпілювати чекер відповідним компілятором та перемістити його до робочої директорії.

Крок 7. Перевірка результату компіляції чекера. Якщо компіляція неуспішна – перейти до кроку 16.

Крок 8. Скомпілювати розв’язок відповідним компілятором та перемістити його до робочої директорії.

Крок 9. Перевірка результату компіляції розв’язку. Якщо компіляція неуспішна – перейти до кроку 17.

Крок 10. Цикл по списку отриманих тестових пар. Після циклу перейти до кроку 18.

Крок 11. Скопіювати вхідні дані поточного тесту до робочої директорії.

Крок 12. Запустити розв’язок користувача у ізольованому середовищі та надати йому доступ до вхідних даних.

Крок 13. Скопіювати відповіді поточного тесту до робочої директорії.

Крок 14. Запустити чекер у ізольованому середовищі та надати йому доступ до вхідних даних, відповідей на тест і результатів роботи програми користувача.

Крок 15. Записати результат роботи чекера у список результатів results.

Крок 16. Записати у список результатів results інформацію про критичну помилку в зв’язку з неуспішністю компіляції чекера.

Крок 17. Записати у список результатів results інформацію про помилку в компіляції розв’язку.

Крок 18. Видалити тимчасову робочу директорію.

Крок 19. Передати результати тестування зі списку results до основного модулю програми.

Крок 20. Кінець.

Усі операції модулю логуються, що в подальшому дозволить виявити неочікувані помилки та частини алгоритму, які можна було б покращити. Робота модулю відбувається в робочих директоріях, що дозволяє ізолювати розв’язки між собою та дає можливість виконувати паралельне тестування.

2.5 Розробка методу використання процесорних тактів для оцінки часу тестування програми

Традиційним методом обрахунку часу роботи програми є вирахування різниці у часі між початком і кінцем роботи програми. Також використовують показники операційної системи. Головним недоліком такого підходу є некоректність результату у випадку, коли програма використовує декілька ядер процесора, та врахування часу, що необхідний процесору на переключення контекстів для роботи з іншими програмами – в цей час програма, що тестується, не виконує жодних операцій.

Іншим підходом до підрахунку часу роботи програм-рішень є аналіз кількості затрачених на їх роботу тактів процесора. Цей метод дозволяє покращити точність результатів оцінювання. Проте такі показники є незручними для користувача. Кількість затрачених тактів можна перевести у секунди, за умови, що тактова частота процесора є незмінною. Цей підхід вносить деякі неточності в зв’язку з тим, що у різних процесорів частота є різною і отримані дані є справедливими лише для конкретної системи. Тому, при заданні обмежень, бажано вказувати обмеження у тактах.

Для користувача зручними є загальноприйняті одиниці виміру часу, такі як секунди та мілісекунди. Тому кількість затрачених тактів необхідно перевести у ці одиниці. Знаючи тактову частоту процесора (за умови, що вона фіксована), можна перевести кількість тактів у мілісекунди за формулою:

$T = \frac{n}{\upsilon} \cdot 1000$, де T – час роботи програми, мс; n – кількість тактів процесора, що затрачені програмою; υ – тактова частота роботи процесора, Гц.

Запропонований метод реалізовано у розробленому веб-ресурсі в експериментальному режимі. Серверна частина містить модуль безпечної перевірки вихідного коду розв'язку програми, розроблений за допомогою технології Windows API на мові C++.

Порівняння значень часу роботи програми різними методами зображено на рисунку 2.5.

Рисунок 2.5 – Порівняння значень часу роботи програми різними методами [3]

Як можна побачити з рисунку 2.5, середній час, виміряний за класичним методом, є дещо більшим та менш стабільним. Це зумовлено наявністю інших працюючих на комп'ютері програм. Отже, запропонований метод має більшу точність оцінювання та стабільність результатів, що обумовлює перспективність його використання в програмах оцінювання розв'язків задач зі

спортивного програмування, де часовий параметр набуває особливо важливого значення.

2.6 Розробка інтерфейсу файлу конфігурації програмного додатку

Для спрощеного використання сервісу було вирішено створити файл конфігурації, що буде містити загальні налаштування програми. В файлі конфігурації перелічено всі необхідні налаштування для коректної роботи програми.

Файл конфігурації має назву «TestLib.WorkerService.exe.config». Він представляє собою текстовий файл у форматі XML. Файл має стандарту структуру згідно з рекомендаціями Microsoft для C# додатків[14]. У випадку відсутності деяких налаштувань, сервіс буде використовувати значення за замовчуванням, що вказані у вихідному коді. Однак деякі налаштування є обов’язковими, тому що неможливо вказати їхні значення наперед. Без таких налаштувань робота сервісу не є можливою, тому під час запуску перевіряється їх наявність та в разі відсутності хоча б одного з обов’язкових значень програма зупиняє свою роботу й записує в лог-файл відповідне повідомлення.

Відкрити файл конфігурації та внести в нього зміни відповідно до користувацьких побажань можна за допомогою вбудованого текстового редактора «Блокнот». Для більш зручного редагування варто використати редактори, що підтримують автоматичний парсинг та форматування файлів у форматі XML, наприклад Notepad++ , Microsoft Visual Studio або Microsoft Visual Studio Code [15].

Файл конфігурації, відкритий за допомогою програми «Блокнот», представлений на рисунку 2.6.

Файл конфігурації містить такі поля:

– worker_name – використовується для ідентифікації сервісу у клієнтській частині. Це поле є обов’язковим;

– cache_folder – вказує шлях до директорії, що буде використовуватися для кешування інформації про задачу. Це поле є опціональним, за

замовчуванням кеш директорія буде створена в поточній директорії запуску сервісу;

– compilers_config_folder – вказує шлях до директорії з налаштуваннями усіх доступних на сервері компіляторів. Це поле є опціональним, за замовчуванням ця директорія буде шукатися в поточній директорії запуску сервісу;

– worker_slot_count – це налаштування використовується, щоб налаштувати ступінь багатонитковості (англ. multi-threading) додатку. Це поле є опціональним та за замовчуванням додаток буде працювати в одному потоці. Дане поле додано з метою можливості керування навантаженнм на систему.

– result_sending_cache_size – це налаштування вказує кількість результатів, які одночасно можуть зберігатися у кеші до надсилання їх на клієнтську частину. Це поле є опціональним та за замовчуванням додаток буде кешувати 2048 результатів;

– problems_cache_size – це налаштування вказує кількість задач, що можна одночасно помістити у кеш. Це поле є опціональним та за замовчуванням додаток буде кешувати 16 задач;

– workarea – вказує шлях до директорії, де будуть відбуватися усі тестування. Це поле є опціональним, за замовчуванням вона буде створена в поточній директорії запуску сервісу;

– base_address – вказує схему комунікації та хост (доменне ім’я) клієнтської частини. Це поле є обов’язковим, оскільки адреса системи невідома заздалегідь.

– api_path – вказує шлях до базового ресурсу клієнтської частини. Це поле є опціональним, за замовчуванням буде використано /api;

– api_token – вказує токен авторизації, що буде використано під час комунікації з клієнтською частиною. Це поле є обов’язковим;

– get_submission_delay – вказує час у мілісекундах між двома запитами до клієнтської частини для отримання списку нових надісланих рішень. Це поле є опціональним, за замовчуванням запити надсилаються кожної секунди;

– update_latest_version_url – вказує URL ідентифікатор ресурсу, що надає останню версію сервісу для оновлення поточної.

TestLib.WorkerService.exe.config — Блокнот

Файл Правка Формат Вид Справка

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
  </startup>
  <appSettings>
    <add key="worker_name" value="Tester7-vm-2s" />

    <add key="cache_folder" value="C:\TestLib.Worker\cache\" />
    <add key="compilers_config_folder" value="C:\TestLib.Worker\compilers\" />
    <add key="worker_slot_count" value="1" />
    <add key="problems_cache_size" value="3" />
    <add key="result_sending_cache_size" value="2048" />

    <add key="workarea" value="C:\TestLib.Worker\workarea\" />
    <add key="base_address" value="https://codelabs.site" />
    <add key="api_path" value="/api" />

    <add key="api_token" value="488" />
    <add key="stage_api_token" value="752" />

    <add key="get_submission_delay" value="5000" />

    <add key="update_latest_version_url" value="http://update.codelabs.site/api/version" />
    <add key="update_latest_program_url" value="http://update.codelabs.site/api/update?version=latest" />
  </appSettings>
</configuration>
```

Рисунок 2.6 – Файл конфігурації програмного додатку

2.7 Розробка комбінованого методу підвищення комп’ютерної безпеки при запуску недовіреного коду на сервері

Запуск стандартної віртуальної машини вимагає значних системних ресурсів для повної віртуалізації усіх пристроїв. Якщо потрібно виконати тільки одну програму, все одно необхідно завантажувати операційну систему та робити усі підготовчі етапи у віртуальному середовищі. Крім накладання витрат системних ресурсів, на це також потрібен додатковий час.

Ще одним недоліком повноцінних віртуальних машин є те, що вони орієнтовані на запуск операційних систем, а тому, щоб автоматично запустити цільову програму, необхідна додаткова конфігурація.

Також для запуску окремих процесів використовують технології контейнеризації. Вони не вимагають великих ресурсів, проте не забезпечують

такого рівня ізоляції, як повноцінна віртуальна машина. Тому запуск недовіреного коду в контейнерах також може нести ризики безпеки.

Легковісні віртуальні машини виглядають і працюють як контейнери. Але вони забезпечують ізоляцію робочого навантаження і мають переваги безпеки віртуальних машин. Разом з цим вони, як і контейнери, дозволяють запустити лише одну програму чи процес без накладних витрат на віртуалізацію опційного обладнання та запуску операційної системи.

Libkrun – це динамічна бібліотека, яка дозволяє програмам легко запускати дочірні процеси в частково ізольованому середовищі за допомогою віртуалізації на рівні операційної системи. Вона інтегрує монітор віртуальної машини, тобто частину користувальницького простору гіпервізора, з мінімальною кількістю емульованих пристроїв, які є необхідними для забезпечення працездатності запуску процесу. При цьому більша частина ускладнень, які виникають від керування віртуальною машиною, абстрагуються, а користувачам пропонується простий API.

Використання легковісних віртуальних машин вирішує поставлену задачу безпечного запуску недовіреного коду на боці серверу. Також вирішується проблема надмірного використання ресурсів класичної віртуальної машини [5]. Для спрощення керування такими машинами доцільно використовувати бібліотеку libkrun. З її допомого можна загорнути недовірений вихідний код програми злонаміреного студента у невелику підпрограму, що самостійно завантажить віртуальне середовище, у якому запуститься головний процес.

Перевагою є те, що бібліотека libkrun бере на себе усі процеси перенаправлення потоків введення/виведення, виділення ресурсів та запуску легковісної віртуальної машини. Також ця бібліотека є простою і зручною в інтеграції в існуючі рішення компіляції вихідного коду та запуску програми. Для цього необхідна мінімальна модифікація процесу компіляції шляхом додавання мітки лінкування із файлом бібліотеки.

Недоліком такого методу є складність оцінювання реального часу виконання програми, оскільки час запуску віртуалізованого середовища є

змінним. Також такий підхід досить складно застосувати для запуску програм на деяких типах мов програмування, таких як інтерпретовані. Хоча у загальному випадку бібліотека libkrun функціонує, вона все ще знаходиться на дуже ранній стадії розробки, а тому необхідно добре протестувати працездатність такого рішення у реальних умовах.

Таким чином, опишемо покроковий алгоритм комбінованого методу підвищення комп'ютерної безпеки при запуску недовіреного коду на сервері:

1. Початок.
2. Завантаження недовіреного вихідного коду розв'язку користувача, вихідного коду програми чекеру та тестів до задачі.
3. Запуск компіляції вихідних кодів у виконувані файли з використанням додаткових флагів, що оповіщують компілятор про необхідність використання бібліотеки libkrun.
4. Для кожного тесту:
    a. запуск виконуваного файлу розв'язку у легковісній віртуальній машині із завантаженням у неї вхідних даних тесту;
    b. аналіз затрачених при роботі часу та оперативної пам'яті з урахуванням обмежень задачі;
    c. запуск програми чекеру у легковісній віртуальній машині із передачею вихідних даних роботи програми для отримання вердикту перевірки для поточного тесту.
5. Кінець.

2.8 Висновки

У другому розділі було розроблено та описано алгоритми роботи серверної частини веб-ресурсу «Codelabs». Представлено блок схеми основного алгоритму роботи Windows сервісу та модулю відповідального за тестування розв'язків на заданому наборі тестів «Tester».

Проаналізовано особливості використання процесорних тактів при оцінюванні часу роботи програм. Було розроблено модуль, що відповідає за

логування дій, що виконуються у всій програмі з можливостями конфігурації, що дає змогу налаштувати детальність інформації у лог файлі. При налаштуванні найдетальнішого звітування кожен логічний етап програми буде супроводжуватись відповідним записом у лог файлі. Цей файл містить важливу інформацію, що в подальшому допоможе швидко знаходити помилки та отримати повну інформацію о роботі системи, для її покращення. Також побудовано загальну схему роботи веб-ресурсу та діаграми компонентів, що відображають взаємозв'язок між різними модулями серверної частини. А також розроблено комбінований метод компіляції для використання бібліотеки libkrun, для отримання програми, що автоматично запускається у легковісній віртуальній машині, яка має більший рівень ізоляції, ніж контейнери, проте не потребує таких ресурсів, як повноцінна віртуальна машина.

Розроблено конфігураційний файл, що дає можливість змінювати головні налаштування сервісу. Він має зручний формат та може бути відредагований вбудованими засобами OS Windows. Частина налаштувань є обов’язковими, для зв’язку з клієнтською частиною, інші можуть бути налаштовані за необхідністю.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СЕРВЕРНОЇ ЧАСТИНИ

3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації програмного засобу.

Перед початком розробки серверної частини веб-ресурсу для розміщення і автоматизованої перевірки олімпіадних задач необхідно обрати мову програмування, яку краще для цього застосувати. В зв’язку з необхідністю створити програму, що використовую функції Windows API для безпечного запуску користувацьких рішень, в якості мови програмування необхідно обрати ту, що підтримує такі виклики. Також мова програмування повинна мати зручний інтерфейс для роботи з протоколами мережі Інтернет для забезпечення комунікації з клієнтською частиною.

C++ – мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм, таких як відеоігри. Інтерфейс Windows API написаний на мові програмування C. Так як мова C++ є нащадком мови C, вона дозволяє нативно викликати функції мови C, зокрема Windows API. Стандартна бібліотека включає загальновживані контейнери і алгоритми. C++ поєднує властивості як низькорівневих, так і високорівневих мов програмування. C++ продовжує розвиватися, щоб відповідати сучасним вимогам. Останнім часом з'явилася спроба об'єднання ефективності C++, безпеки і швидкості розробки, як в Java і C# – були запропоновані такі мови як D та Rust , проте поки що вони не отримали широкого визнання[16].

C# – об'єктно-орієнтована мова програмування для платформи .NET. Перевагою даної мови є те, що розмір скомпільованих програмних продуктів є досить малим. Синтаксис C# близький до C++ і Java. Мова має жорстку статичну типізацію, підтримує основні засади об’єктно-орієнтованого

програмування (поліморфізм, наслідування, інкапсуляцію), заміщення операторів, посилання на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників — мов C++, Delphi, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++). Якщо програми на різних мовах виконуються на платформі .NET, .NET бере на себе клопіт щодо сумісності програм (тобто типів даних, за кінцевим рахунком)[17].

Java – об'єктно-орієнтована мова програмування, що спочатку розроблялася для побутової електроніки, але згодом стала використовуватися для написання аплетів, додатків і серверного програмного забезпечення. Розроблена в 1995 році компанією Sun Microsystems і з самого початку проектувалась як об’єктно-орієнтована мова. В цьому вона вигідно відрізняється від C++, який, з міркувань сумісності, зберігає багато елементів процедурної мови C. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як кросплатформенна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано[18].

Для вибору розглянуто такі критерії, як наявність об’єкто-орієнтовної парадигми, реалізація динамічних масивів, застосування багатонитковості (англ. multi-threading), кросплатформеність, статична типізація, швидкість виконання розробленого програмного продукту, можливість нативного виклику функцій бібліотеки Windows API і можливість роботи з протоколами веб комунікації, оскільки вони є найактуальнішими при виборі мови.

Для об’єктивного порівняння вище наведених мов програмування по визначеним критеріям зведемо усю інформацію у таблицю 3.1.

Таблиця 3.1 – Порівняння мов програмування

Характеристики	Мова програмування		
	C++	C#	Java
Об’єктно-орієнтована	+	+	+
Динамічні масиви	+	+	+
Багатонитковість	+	+	+/–
Розширення сторонніх об’єктів	–	+	–
Кросплатформеність	+/–	+/–	+
Статична типізація	+	+	+
Швидкість виконання	+	+	–
Нативний виклик Windows API	+	–	–
Роботи з протоколами веб комунікації	–	+	+

Мова програмування C# має перевагу над Java і C++, тому для реалізації Windows сервісу було обрано саме її, адже вона задовольняє всі необхідні вимоги, окрім нативного виклику Windows API. Для роботи з бібліотекою Windows API необхідно буде розробити модуль на мові C++ та завантажити його за допомогою інструменту C++/CLI у програму на C#.

Для вибору середовища розробки створення програмного додатку було проаналізовано такі середовища розробки, як Microsoft Visual Studio, MonoDevelop та C++ Builder.

Microsoft Visual Studio — серія продуктів фірми Microsoft для програмування на таких мовах, як C++, C#, J# та F#. Дане IDE включає інтегроване середовище розробки програмного забезпечення з вбудованою технологією IntelliSense та ряд інших інструментальних засобів. IntelliSense – технологія автодоповнення Microsoft[19]. Дописує назву функції при введенні

початкових літер. Крім прямого призначення, IntelliSense використовується для доступу до документації та для усунення неоднозначності в іменах змінних, функцій і методів, використовуючи рефлексію. Також дане середовище дозволяє розробити зручний інтерфейс користувача, прослідкувати хід виконання програми в режимі реального часу, контролювати використання апаратних ресурсів, провести модульне та повне тестування. Вона має вбудовані можливості для командної роботи з репозиторіями вихідного коду, дозволяє під'єднуватися та переглядати бази даних, тощо. Інтерфейс Microsoft Visual Studio зображено на рисунку 3.1.

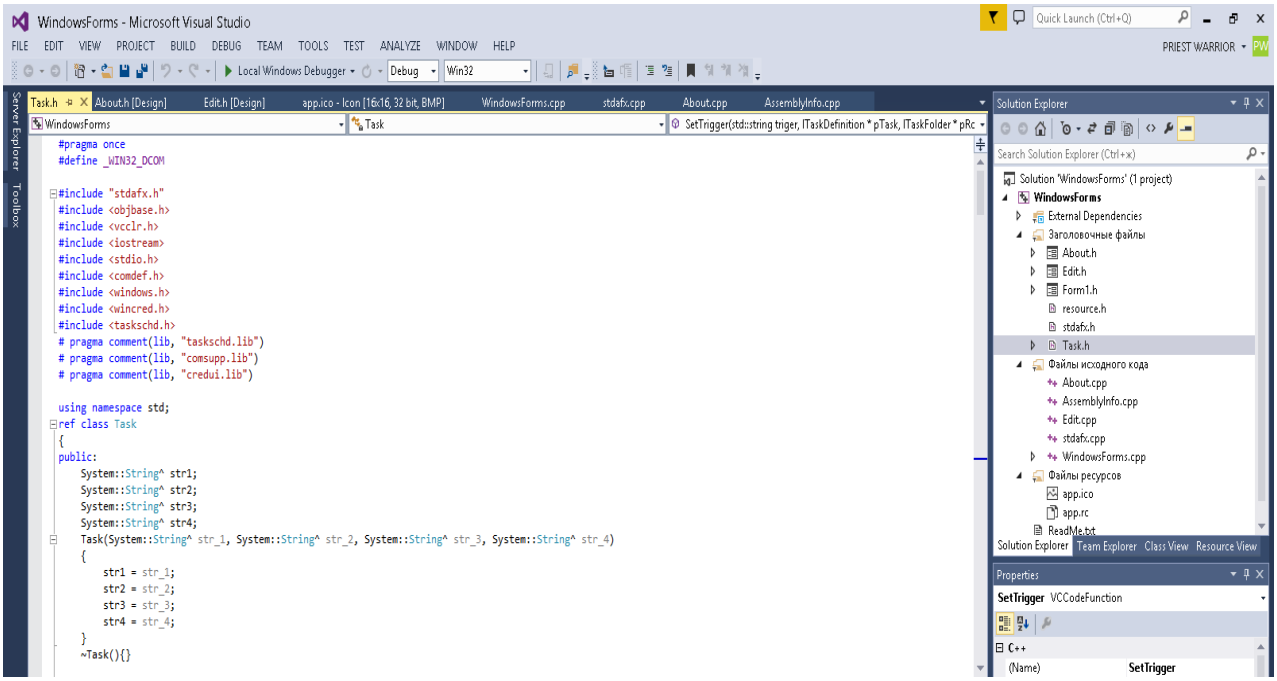


Рисунок 3.1 – Приклад роботи програми Microsoft Visual Studio

MonoDevelop — дозволяє розробникам швидко писати настільні та веб-додатки на Linux, Windows та macOS. Це також спрощує розробникам портування .NET-додатків, створених за допомогою Visual Studio для Linux та macOS, підтримуючи єдину кодову базу для всіх платформ. MonoDevelop включає можливості подібні до NetBeans та Microsoft Visual Studio, такі як автоматичне доповнення, інтеграція контролю коду, графічний користувацький інтерфейс і веб-дизайнер[20]. В MonoDevelop інтегрований Gtk# GUI дизайнер

під назвою Stetic. З часом проект MonoDevelop був занурений в решту частин проекту Mono і активно підтримується Novell і спільнотою Mono. Починаючи від Mono 1.0 Beta 2, MonoDevelop включений в комплект релізів Mono. Є можливість інтеграції з Microsoft Visual Studio і. NET Framework (в середовищі Microsoft Windows). MonoDevelop зображено на рисунку 3.2.

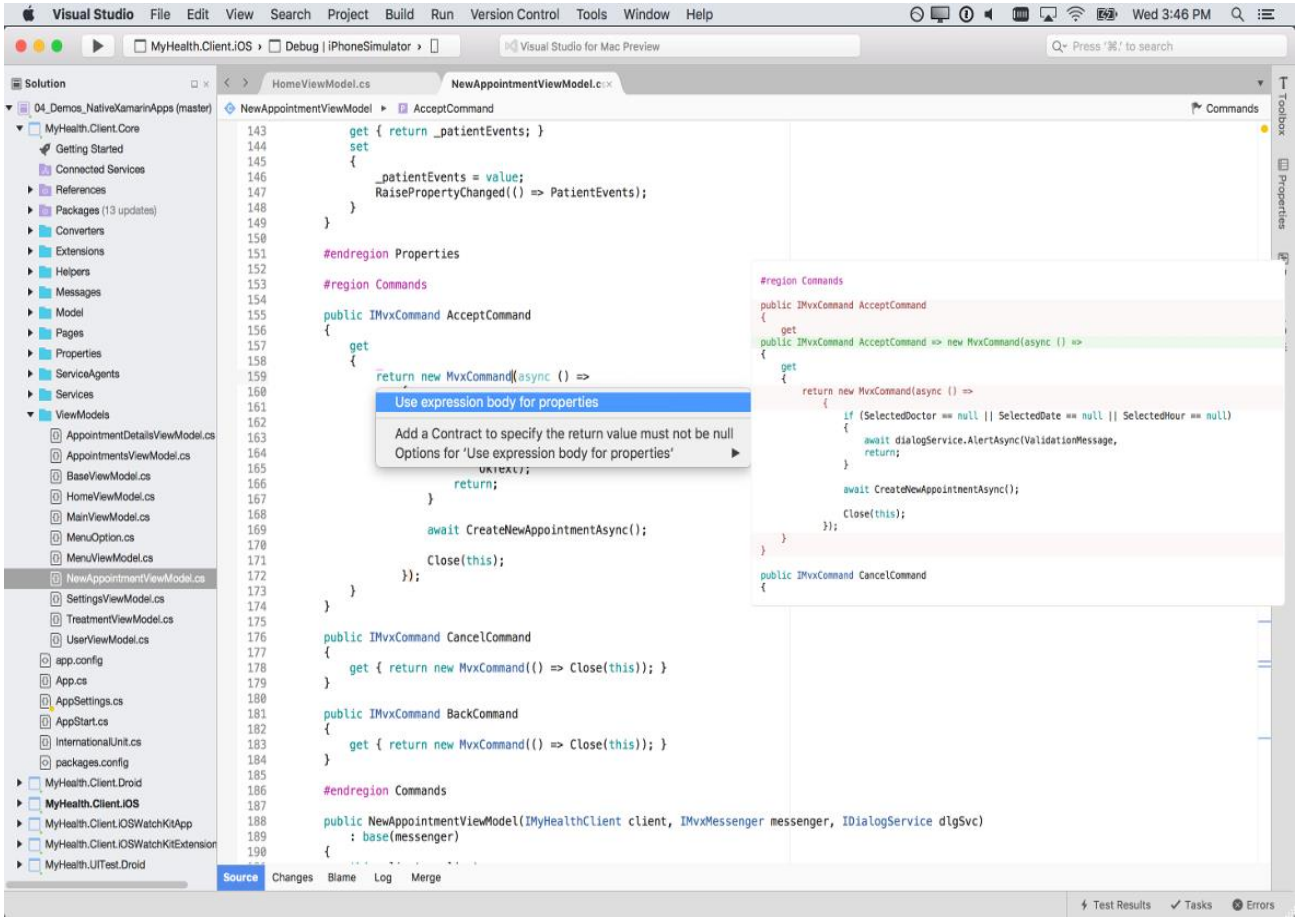


Рисунок 3.2 – Приклад роботи програми MonoDevelop

C++ Builder – середовище розробки, що розробляється компанією Codegear. Призначений для розробки програм на мові програмування C++. C++ Builder об'єднує Бібліотеку візуальних компонентів і середовище програмування, написане на Delphi з компілятором C++. C++ Builder містить інструменти, які дозволяють здійснювати візуальну розробку Windows-програм методом drag-and-drop, спрощуючи програмування завдяки WYSIWYG редакторові інтерфейсу, вбудованому в його середовище розробки[21]. Приклад роботи зображено на рисунку 3.3.

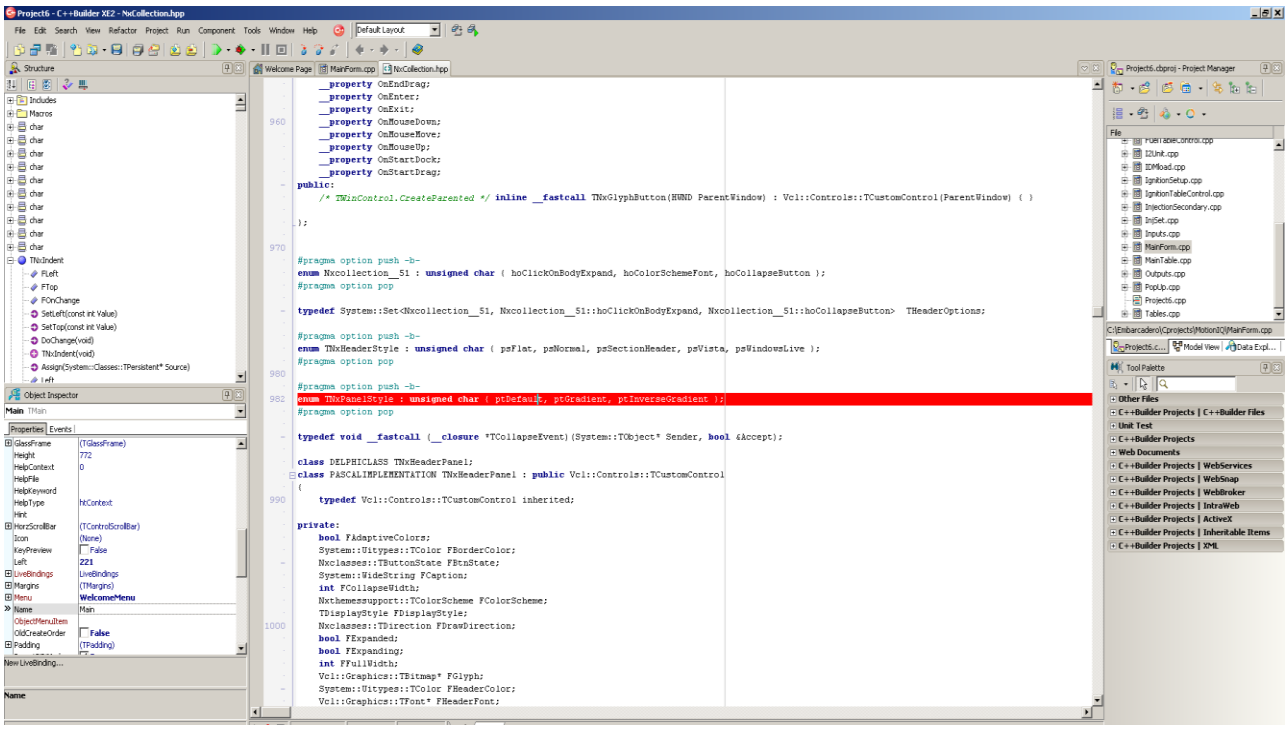


Рисунок 3.3 – Приклад роботи програми C++ Builder

Порівняння розглянутих середовищ програмування за обраними критеріями наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння середовищ програмування

Функції	Середовище програмування		
	C++ Builder	MonoDevelop	Visual Studio
Одночасна робота з C++ та C#	–	–	+
Режим відлагодження Windows сервісів	–	–	+
Робота з GIT репозиторієм	–	+	+
Підтримка XML файлів конфігурації	+	+	+
Кросплатформеність	–	+	–
Кросплатформенна компіляція	+	–	+

Серед розглянутих середовищ було обрано Microsoft Visual Studio, оскільки це середовище, на відміну від інших, підтримує одночасну робота з C++ та C#, дозволяє відлагодження Windows сервісів і дає можливість командної роботи з використанням GIT репозиторіїв.

3.2 Розробка головного модуля Windows сервісу

Програмне забезпечення розроблялося за принципами об’єктно-орієнтованого програмування, тому модуль складається з таких класів, як: Application, Configuration та WorkerTaskManager.

Клас Application є точкою входу сервісу. З цього класу починається запуск сервісу, він містить функції для ініціалізації інших модулів та виконує головний цикл обробки усіх повідомлень, що надходять з різних компонентів системи. Функціями цього класу є Init, Start та Stop.

Вхідними даними для класу є змінні середовища запуску, що описують початкові параметри.

Функція Init – виконує первинну ініціалізацію програми. Вона виконує логування параметрів запуску (рисунок 3.4), після чого відбувається ініціалізація і очищення структури директорій (рисунок 3.5) та завантаження конфігурації компіляторів (рисунок 3.6). Далі виконується під’єднання серверу оцінювання робіт до клієнтського додатку для отримання завдань через відповідне АПІ (рисунок 3.7). Вхідні дані – зчитана конфігурація програми. Вихідні дані – статус ініціалізації системи.

У випадку помилки первинної ініціалізації, ця помилка буде записана у лог файл та програма завершить свою роботу. У випадку успішної ініціалізації програма передає керування менеджеру сервісів Windows.

Функція Start, вихідний код якої зображено на рисунку 3.8, виконує запуск усіх модулів програми після їх ініціалізації, а саме запуск головної черги команд для обробки задач (рисунок 3.9) і черги для надсилання результатів (рисунок 3.10). Вхідні дані – проініціалізовані модулі. Вихідні дані – статус головної черги.

```
public bool Init() => Initialized = init();
private bool init()
{
        LoggerManaged.InitNativeLogger(new LoggerManaged.LogEventHandler(LogManager.GetLogger("Native").Log));

        logger.Debug("Current directory is {0}", Directory.GetCurrentDirectory());
        logger.Debug("AppData folder is {0}", Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
        logger.Debug("Current user is {0}", System.Security.Principal.WindowsIdentity.GetCurrent().Name);
```

Рисунок 3.4 – Запис параметрів запуску у лог-файл

```
        private static void CleanDirectory(string directory)
        {
                DirectoryInfo dirInfo = new DirectoryInfo(directory);

                if (dirInfo.Exists)
                {
                        foreach (FileInfo file in dirInfo.GetFiles())
                                DeleteFile(file.FullName);

                        foreach (DirectoryInfo dir in dirInfo.GetDirectories())
                                DeleteDirectory(dir.FullName);
                }
                else
                        Directory.CreateDirectory(dirInfo.FullName);
        }
```

Рисунок 3.5 – Ініціалізація директорії

```
var files = Directory.GetFiles(directory);
foreach (var file in files)
{
        var info = new FileInfo(file);
        if (info.Extension != ".cfg")
        {
                logger.Warn("Incorect compiler configuration file extension. File {0} was skipped", info.Name);
                continue;
        }
        else
                logger.Debug("Try to load configuration from file {0}", info.Name);

        try
        {
                Compiler compiler;
                XmlSerializer serializer = new XmlSerializer(typeof(Compiler));
                using (FileStream fs = info.OpenRead())
                using (XmlReader reader = XmlReader.Create(fs))
                        compiler = (Compiler)serializer.Deserialize(reader);

                compiler.Commands.Sort((x, y) => x.Order - y.Order);
                compilers.Add(compiler.Id, compiler);
        }
        catch (Exception ex)
        {
                logger.Warn(ex, "Load configuration failed");
        }
}
```

Рисунок 3.6 – Завантаження конфігурації компіляторів

```
if (Configuration.WorkerId == Guid.Empty)
{
        if (!signUp())
        {
                logger.Error("Application sign up failed");
                return false;
        }
}

var status = apiClient.SignIn(Configuration.WorkerId);
if (status == UpdateWorkerStatus.LoginIncorrect)
{
        logger.Error("Application sign in failed. Worker id is incorrect. Trying re-sign up");

        if (!signUp())
        {
                logger.Error("Application sign up failed");
                return false;
        }
}
else if (status != UpdateWorkerStatus.Ok)
{
        logger.Error("Application sign in failed");
        return false;
}

if (!workerTasks.Init(Configuration))
{
        apiClient.SignOut(Configuration.WorkerId);
        return false;
}
```

Рисунок 3.7 – Під’єднання до клієнтського додатку

Сервер оцінювання робіт отримує команди від клієнтського додатку через HTTP API. Серед команд, опрацювання яких відбувається сервером оцінювання робіт, можна виділити:

– тестування нового розв’язку (рисунок 3.11);
– завантаження тестових даних для певної задачі (рисунок 3.12);
– надсилання результатів до клієнтської частини.

Ці команди необхідно виконувати у заданому порядку, саме тому для їх організації використовується така структура даних, як черга. При цьому надсилання результатів можна виконувати паралельно іншим задачам, тому

програма має дві черги: головну – для роботи з усіма командами, і додаткову – для відсилання результатів тестування назад до клієнтського додатку.

```
public void Start()
{
        if (started)
                return;

        StartSendingTestingResult();
        StartSlots();

        started = true;
}
```

Рисунок 3.8 – Функція Start

```
private void StartSlot(uint id)
{
        slots[id - 1] = slots[id - 1] ?? new Slot(id);
        workerTasks[id] =
                Task.Run(() => slots[id - 1].Do(slotsCancellationTokenSource.Token), slotsCancellationTokenSource.Token);
}
```

Рисунок 3.9 – Запуск головної черги команд

```
private void SendTestingResult()
{
        Application app = Application.Get();
        IApiClient client = new HttpCodelabsApiClient();
        var logger = LogManager.GetCurrentClassLogger();

        sendingCancellationTokenSource.Token.Register(() => app.RequestMessages.Enqueue(null));

        while (true)
        {
                var request = app.RequestMessages.Dequeue();
                if (request is null)
                        break;

                try
                {
                        client.SendRequest(request);
                }
                catch (Exception ex)
                {
                        logger.Error("Error sending request {0} to server: {1}. Some data will be lose", request.RequestUri, ex);
                }
        }

        sendingCancellationTokenSource.Token.ThrowIfCancellationRequested();
}
```

Рисунок 3.10 – Обробник додаткової черги для надсилання результатів

```
var submissions = client.GetSuitableSubmissions(app.Compilers.GetCompilers());
if (!submissions.Any())
{
        Thread.Sleep(app.Configuration.GetSubmissionDelayMs);
        continue;
}

var submission = submissions.First();
if (!client.SendRequest(client.GetTakeSubmissionsRequestMessage(submission.Id)))
{
        continue;
}

logger.Info("Testing slot {0} taken submission with id {1}", slotNumber, submission.Id);
logger.Debug("Submission: {0}", submission);
```

Рисунок 3.11 – Обробник запиту на тестування нового розв’язку.

```
Problem problem = app.Problems.FetchProblem(
        submission.ProblemId, submission.ProblemUpdatedAt,
        () =>
        {
                logger.Debug("Need download problem with id {0}", submission.ProblemId);
                var downloadedProblem = client.DownloadProblem(submission.ProblemId);
                if (downloadedProblem is null)
                {
                        logger.Error("Failed to download problem with id {0}", submission.ProblemId);
                        result = WorkerResult.TestingError;
                }

                return downloadedProblem;
        }
);
```

Рисунок 3.11 – Обробник запиту на завантаження тестових даних задачі.

Власне запуск тестування певної роботи користувача відбувається після завантаження даних цього розв’язку та тестових даних задачі. Вихідний код відповідної функції зображено на рисунку 3.12. Для цього використовується додатковий клас Worker із такими функціями, як: compileSolution для компіляції вихідного коду програми розв’язку (рисунок 3.13), compileChecker для компіляції вихідного коду програми перевірки результату роботи (рисунок 3.14), функція для запуску програми користувача (рисунок 3.15), функція для

перевірки лімітів після роботи програми (рисунок 3.16), функція для запуску програми чекера з метою отримання вердикту (рисунок 3.17).

```
Worker worker = new Worker(slotNumber, client);

if (result != WorkerResult.TestingError)
{
        try
        {
                result = worker.Testing(submission, problem, solution);
        }
        catch (Exception ex)
        {
                logger.Error("Slot {0} worker testing failed with exception {1}. Error {2}", slotNumber, ex.GetType().Name, ex);
                result = WorkerResult.TestingError;
        }
}

switch (result)
{
        case WorkerResult.Ok:
        case WorkerResult.CompilerError:
                app.RequestMessages.Enqueue(client.GetReleaseSubmissionsRequestMessage(submission.Id, result));
                break;
        case WorkerResult.TestingError:
                app.RequestMessages.Enqueue(client.GetFailSubmissionsRequestMessage(submission.Id));
                break;
}
```

Рисунок 3.12 – Тестування розв’язку користувача

```
private bool compileSolution(string workdir, ProblemFile solution, Compiler compiler)
{
        string sourceFilename = $"solution{compiler.FileExt}";
        string sourceFullPath = Path.Combine(workdir, sourceFilename);
        string compilerLogFullPath = Path.Combine(workdir, Application.Get().Configuration.CompilerLogFileName);

        Application.Get().FileProvider.Copy(solution, sourceFullPath);

        var replacement = GenerateReplacementDictionary(
                        sourceFullPath: sourceFullPath,
                        sourceFilename: sourceFilename,
                        binaryFullPath: Path.Combine(workdir, solutionBinaryFilename),
                        binaryFilename: solutionBinaryFilename,
                        workDirecrory: workdir,
                        compilerLogFilename: Application.Get().Configuration.CompilerLogFileName,
                        compilerLogFullPath: compilerLogFullPath
                );

        return compile(workdir, compiler, replacement, compilerLogFullPath, false);
}
```

Рисунок 3.13 – Компіляція вихідного коду розв’язку

```
private bool compileChecker(string workdir, ProblemFile checker, Compiler compiler)
{
        string sourceFilename = $"checker{compiler.FileExt}";
        string sourceFullPath = Path.Combine(workdir, sourceFilename);
        string compilerLogFullPath = Path.Combine(workdir, Application.Get().Configuration.CompilerLogFileName);

        Application.Get().FileProvider.Copy(checker, sourceFullPath);

        var replacement = GenerateReplacementDictionary(
                        sourceFullPath: sourceFullPath,
                        sourceFilename: sourceFilename,
                        binaryFullPath: Path.Combine(workdir, checkerBinaryFilename),
                        binaryFilename: checkerBinaryFilename,
                        workDirecrory: workdir,
                        compilerLogFilename: Application.Get().Configuration.CompilerLogFileName,
                        compilerLogFullPath: compilerLogFullPath
                );

        return compile(workdir, compiler, replacement, compilerLogFullPath, true);
}
```

Рисунок 3.14 – Компіляція вихідного коду програми перевірки

```
Tester tester = new Tester();

var replacement = GenerateReplacementDictionary(
        binaryFullPath: Path.Combine(workdir, solutionBinaryFilename),
        binaryFilename: solutionBinaryFilename,
        workDirecrory: workdir);

string program = solutionCompiler.RunCommand.Program.ReplaceByDictionary(re, replacement);
string args = solutionCompiler.RunCommand.Arguments.ReplaceByDictionary(re, replacement);

tester.SetProgram(program, $"\"{program}\" {args}");

tester.SetWorkDirectory(workdir);
tester.SetRealTimeLimit(submission.RealTimeLimit);
tester.SetMemoryLimit(submission.MemoryLimit);
tester.RedirectIOHandleToFile(IOHandleType.Input, inputFileFullPath);
tester.RedirectIOHandleToFile(IOHandleType.Output, outputFileFullPath);
tester.RedirectIOHandleToHandle(IOHandleType.Error, tester.GetIORedirectedHandle(IOHandleType.Output));

logger.Info("Slot {0}: Run solution", slotNum);
if (tester.Run(true))
{
        logger.Info("Slot {0}: Solution run successfully", slotNum);
}
else
{
        logger.Error("Slot {0}: Can't run solution", slotNum);
        return WorkerResult.TestingError;
}
```

Рисунок 3.15 – Запуск програми користувача

```
if (testResult.WorkTime > submission.TimeLimit)
{
        testResult.Result = TestingResult.TimeLimitExceeded;

        Application.Get().RequestMessages.Enqueue(apiClient.GetSendTestingResultRequestMessage(testResult));

        logger.Info("Slot {0}: Solution work {1}ms and time limit {2}ms",
                slotNum, testResult.WorkTime, submission.TimeLimit);

        //Not start checker
        continue;
}

if (testResult.UsedMemmory > submission.MemoryLimit)
{
        testResult.Result = TestingResult.MemoryLimitExceeded;
        Application.Get().RequestMessages.Enqueue(apiClient.GetSendTestingResultRequestMessage(testResult));

        logger.Info("Slot {0}: Solution used {1}kb memory and memory limit {2}kb",
                slotNum, testResult.UsedMemmory, submission.MemoryLimit);

        //Not start checker
        continue;
}
```

Рисунок 3.16 – Перевірка лімітів після роботи програми

```
if (tester.Run())
{
        logger.Info("Slot {0}: Checker run successfully", slotNum);
}
else
{
        logger.Error("Slot {0}: Can't run checker", slotNum);
        return WorkerResult.TestingError;
}

if (tester.Wait() == WaitingResult.Ok)
{
        logger.Debug("Slot {0}: Waited successfully", slotNum);
}
else
{
        logger.Error("Slot {0}: Wait failed", slotNum);
        return WorkerResult.TestingError;
}

uint exitCode = tester.GetExitCode();
logger.Info("Slot {0}: Checker exit with code {1}", slotNum, exitCode);

testResult.Result = (TestingResult)exitCode;
testResult.Log = File.ReadAllText(reportFileFullPath);

Application.Get().RequestMessages.Enqueue(apiClient.GetSendTestingResultRequestMessage(testResult));
```

Рисунок 3.17 – Запуск програми чекера для отримання вердикту

Функція Stop, зображена на рисунку 3.18, виконує зупинку усіх запущених модулів програми. Перед зупинкою усі команди з головної черги повинні бути виконані, але нові команди не можуть бути додані. Тому черга переходить у режим лише для читання. Для цього використовуються відповідні функції для опрацювання головної черги (рисунок 3.19) та додаткової черги (рисунок 3.20).

Вхідні дані – команди з головної черги.

Вихідні дані – статус системи.

Функції Start та Stop оброблюють відповідні події запуску чи зупинки сервісу від менеджеру сервісів Windows.

```
public void Stop()
{
        if (!started)
                return;

        slotsCancellationTokenSource.Cancel();
        sendingCancellationTokenSource.Cancel();

        StopSlots();
        StopSendingTestingResult(true);

        started = false;
}
```

Рисунок 3.18 – Функція Stop

```
private void StopSlots()
{
        slotsCancellationTokenSource.Cancel();

        try
        { Task.WaitAll(new ArraySegment<Task>(workerTasks, 1, workerTasks.Length - 1).Array); }
        catch { }
}
```

Рисунок 3.19 – Зупинка головної черги

```
private void StopSendingTestingResult(bool flushQuery)
{
        sendingCancellationTokenSource.Cancel();

        try
        { Task.WaitAll(workerTasks[0]); }
        catch { }
}
```

Рисунок 3.20 – Зупинка додаткової черги

Клас Configuration виконує функції менеджера конфігурації. Він читає наявний файл конфігурації та дозволяє оновити конфігурацію у файлі. Функціями цього класу є Get та Set, що реалізовані для кожного з параметрів конфігурації подібним чином (рисунок 3.21). Вхідними даними для класу є шлях до файлу конфігурації.

Функція Get призначена для отримання поточної значення певного поля, що вказане у відповідному файлі конфігурації.

Вхідні дані – найменування поля, шлях до файлу конфігурації.

Вихідні дані – значення даного поля.

Функція Set призначена для зміни поточної значення певного поля на нове, що вказане у відповідному файлі конфігурації.

Вхідні дані – найменування поля, нове значення, шлях до файлу конфігурації.

Вихідні дані – статус запису інформації у файл.

З метою зменшення навантаження на файлову систему операційної системи, читання усіх полів відбувається один раз під час ініціалізації. Після цього функція Get працює з кешованими у оперативній пам’яті даними. Функція Set одразу записує значення у файл, оскільки зміна відбувається не так часто, як читання поточних значень.

```
public Guid WorkerId
{
        get => _workerId;
        set => File.WriteAllText(fid, (_workerId = value).ToString());
}
```

Рисунок 3.21 – Функції Get та Set для параметру ідентифікатора серверу

Разом з цим, клас Configuration має функцію завантаження збереженої раніше конфігурації із файлу, вихідний код якої наведено на рисунку 3.22. Така функція не виконує читання даних з файлу напряму, а використовує для цього бібліотеку для керуваннями конфігурації додатків C#/.NET. Вхідними даними цієї функції є колекція прочитаних із файлу параметрів у вигляді класу NameValueCollection. Вихідними даними цієї функції є проініціалізований клас для збереження конфігурації.

Клас WorkerTaskManager призначений для керування процесом тестування розв’язків. Його задачею є отримання усіх даних, таких як вихідний код, набір тестових даних для задачі та обмеження по часу та пам’яті і виклик модулю «Tester» для подальшого тестування рішення. Функціями класу є GetProblemInformation, GetSolutionInformation, RunTester. Ініціалізацію класу наведено на рисунку 3.23.

Вхідними даними для класу є команда на тестування рішення. Тестування може відбуватися паралельно, це залежить від налаштувань у файлі конфігурації. Відповідно до цих значень, клас буде запускати певну кількість ниток для модулю «Tester».

Функція GetSolutionInformation – виконує завантаження вихідного коду рішення, що необхідно протестувати, використовуючи протокол комунікації з клієнтською частиною ресурсу. Вхідними даними є унікальний числовий ідентифікатор рішення для завантаження. Вихідними даними є вихідний код рішення, унікальний числовий ідентифікатор компілятора, який необхідно використати при компіляції цього рішення та обмеження по часу й пам’яті, які є корегованими під обрану користувачем мову програмування.

```
public Configuration(NameValueCollection config)
{
        Update = new UpdateConfiguration();

        WorkerName = config.Get("worker_name") ?? "TestLib.Worker";

        TestingWorkDirectory = config.Get("workarea") ?? ".\\workarea\\";

        FileCacheFolder = config.Get("cache_folder") ?? ".\\cache\\";
        CompilersConfigFolder = config.Get("compilers_config_folder") ?? ".\\compilers\\";
        ProblemsCacheSize = config.Get("problems_cache_size").ToUInt32OrDefault(1);
        ResultSendingCacheSize = config.Get("result_sending_cache_size").ToUInt32OrDefault(2048);

        BaseAddress = new Uri(config.Get("base_address") ?? "http://localhost:8080/");
        BaseApiAddress = new Uri(BaseAddress, config.Get("api_path") ?? "/api");
        ApiAuthToken = config.Get("api_token");

        InputFileName = config.Get("input_file_name") ?? "input.txt";
        OutputFileName = config.Get("output_file_name") ?? "output.txt";
        AnswerFileName = config.Get("answer_file_name") ?? "answer.txt";
        ReportFileName = config.Get("report_file_name") ?? "report.txt";
        CompilerLogFileName = config.Get("compiler_log_file_name") ?? "compiler_log.txt";

        GetSubmissionDelayMs = config.Get("get_submission_delay").ToInt32OrDefault(500);
        WorkerSlotCount = config.Get("worker_slot_count").ToUInt32OrDefault(1);

        Update.LatestVersionUrl = config.Get("update_latest_version_url") ?? "http://localhost:8081/api/version";
        Update.LatestProgramUrl = config.Get("update_latest_program_url") ?? "http://localhost:8081/api/update?version=latest";

        fid = Path.Combine(
                Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
                Process.GetCurrentProcess().ProcessName);

        if (!Directory.Exists(fid))
                Directory.CreateDirectory(fid);

        fid = Path.Combine(fid, "fid");

        if (File.Exists(fid))
                if (!Guid.TryParse(File.ReadAllText(fid), out _workerId))
                {
                        File.Delete(fid);
                        _workerId = Guid.Empty;
                }
}
```

Рисунок 3.22 – Функція завантаження збереженої конфігурації

Функція FetchProblem – виконує завантаження інформації про задачу, рішення якої необхідно перевірити (рисунок 3.24). Ця інформація містить вихідний код чекеру та набір тестових даних. Для пришвидшення тестування та зменшення навантаження на мережу виконується кешування завантаженої інформації у файловій системі. коли така задача вже перевірялася системою та не відбулося її редагування на клієнтській частині, інформація про задачу зчитується з відповідних файлів. Вхідні дані – унікальний ідентифікатор задачі та час редагування. Вихідні дані – вихідний код чекеру, набір тестових даних.

```
public WorkerTaskManager()
{
        logger = LogManager.GetCurrentClassLogger();

        started = false;
}

public bool Init(Configuration configuration)
{
        slots = new Slot[configuration.WorkerSlotCount];
        workerTasks = new Task[configuration.WorkerSlotCount + 1];

        aliveStatusSenderCancellationTokenSource = new CancellationTokenSource();

        try
        {
                aliveStatusSenderTask = Task.Run(
                        () => sendAliveStatus(), aliveStatusSenderCancellationTokenSource.Token);

                return true;
        }
        catch (Exception ex)
        {
                logger.Error(ex, "Can't initalize WorkerTaskManager. Start is not avaliable");

                return false;
        }
}
```

Рисунок 3.23 – Ініціалізація класу WorkerTaskManager

```
public Problem FetchProblem(ulong id, DateTime updatedAt, Func<Problem> func)
{
        var contains = problems.TryGetValue(id, out var problem);

        if (contains && problem.LastUpdate == updatedAt)
        {
                return getProblem(id);
        }
        else
        {
                lock (sync)
                {
                        contains = problems.TryGetValue(id, out problem);

                        if (contains && problem.LastUpdate == updatedAt)
                        {
                                return getProblem(id);
                        }
                        else
                        {
                                problem = func();
                                addOrUpdateProblem(problem, contains);
                                return getProblem(problem.Id);
                        }
                }
        }
}
```

Рисунок 3.24 – Функція FetchProblem

Функція RunTester – виконує запуск модулю «Tester». Для запуску йому передається інформація про розв’язок та задачу. Спочатку запускається компіляція вихідних файлів. Далі, відповідно до налаштувань, відбувається запуск потрібної кількості процесів програм розв’язків з записом їх виведення та параметрів роботи. Після цього запускається чекер, що видає вердикт тестування. Інформація кожного з етапів, логи компілятора та чекера, кінцева оцінка надсилаються до клієнтської частини.

Вхідні дані – інформація про розв’язок та задачу.

Вихідні дані – результати компіляції та тестування по кожному з тестів.

3.3 Розробка модуля «Tester»

Цей модуль написаний на мові C++ та виконує нативні виклики бібліотеки Windows API. Він не має точки входу, оскільки надає можливості іншим модулям зробити виклики системних функцій [22]. Цей модуль складається з файлів Tester та WrapperTester.

Клас WrapperTester є «обгорткою» для класу Tester, що дає можливість запускати клас, написаний на мові C++, з інших модулів програми, написаних на мові C#. Усі функції класу дублюють функції класу Tester та виконують перетворення даних з типів мови C# у типи мови C++ і навпаки. Наприклад System.String перетворюється у wchar_t* за допомогою допоміжних функцій Marshal::StringToHGlobalUni, що необхідно зробити для виклику SetProgram (рисунок 3.25), SetUser (рисунок 3.26), SetWorkDirectory (рисунок 3.27) та RedirectIOHandleToFile (рисунок 3.29). Також необхідно обробити передачу вказівників, так, функція GetIORedirectedHandle виконує перетворення вказівника на ідентифікатор потоку введення-виведення із мови C++ у середовище .NET, для чого потрібно сповістити збирач сміття рантайму про цей вказівник за допомогою оператора gcnew (рисунок 3.28).

Клас Tester виконує компіляцію вихідного коду файлів розв’язку та чекеру, запуск програм в ізольованому середовищі з урахуванням атрибутів

безпеки і отримує параметри їх роботи. Функціями класу RedirestStdStream, SetLimits, ExecuteProgram.

```
void SetProgram(String ^ program, String ^ args)
{
        using namespace Runtime::InteropServices;

        const wchar_t* program_native = (const wchar_t*)(Marshal::StringToHGlobalUni(program)).ToPointer();
        const wchar_t* args_native = (const wchar_t*)(Marshal::StringToHGlobalUni(args)).ToPointer();

        tester->SetProgram(program_native, args_native);

        Marshal::FreeHGlobal(IntPtr((void*)program_native));
        Marshal::FreeHGlobal(IntPtr((void*)args_native));
}
```

Рисунок 3.25 – Функція SetProgram

```
void SetUser(String ^ userName, String ^ domain, String ^ password)
{
        using namespace Runtime::InteropServices;

        const wchar_t* userName_native = (const wchar_t*)(Marshal::StringToHGlobalUni(userName)).ToPointer();
        const wchar_t* domain_native = (const wchar_t*)(Marshal::StringToHGlobalUni(domain)).ToPointer();
        const wchar_t* password_native = (const wchar_t*)(Marshal::StringToHGlobalUni(password)).ToPointer();

        tester->SetUser(userName_native, domain_native, password_native);

        Marshal::FreeHGlobal(IntPtr((void*)userName_native));
        Marshal::FreeHGlobal(IntPtr((void*)domain_native));
        Marshal::FreeHGlobal(IntPtr((void*)password_native));
}
```

Рисунок 3.26 – Функція SetUser

```
void SetWorkDirectory(String ^ directory)
{
        using namespace Runtime::InteropServices;

        const wchar_t* directory_native = (const wchar_t*)(Marshal::StringToHGlobalUni(directory)).ToPointer();
        tester->SetWorkDirectory(directory_native);

        Marshal::FreeHGlobal(IntPtr((void*)directory_native));
}
```

Рисунок 3.27 – Функція SetWorkDirectory

```
IntPtr ^ GetIORedirectedHandle(IOHandleType handleType)
{
        return gcnew IntPtr(tester->GetIORedirectedHandle(handleType));
}
```

Рисунок 3.28 – Функція GetIORedirectedHandle

```
void RedirectIOHandleToFile(IOHandleType handleType, String ^ fileName)
{
        if (fileName == nullptr)
                throw gcnew ArgumentNullException(nameof(fileName));

        using namespace Runtime::InteropServices;

        const wchar_t* fileName_native = (const wchar_t*)(Marshal::StringToHGlobalUni(fileName)).ToPointer();

        tester->RedirectIOHandleToFile(handleType, fileName_native);

        Marshal::FreeHGlobal(IntPtr((void*)fileName_native));
}
```

Рисунок 3.29 – Функція RedirectIOHandleToFile

Функція RedirectIOHandleToFile виконує підміну стандартних потоків введення/виведення перед запуском програми. Це відбувається в декілька етапів: налаштування атрибутів безпеки для файлу, щоб дочірній процес мав доступ до нього (рисунок 3.30); задання параметрів відкриття файлу (рисунок 3.32); власне відкриття файлу (рисунок 3.31); збереження отриманого ідентифікатору файлу у відповідне поле, в залежності від вхідних даних (рисунок 3.33). Це необхідно для передачі вхідних даних тесту до програми-розв’язку та запису її виведення.

Вхідні дані – ідентифікатор системної задачі, який буде використано для подальшого запуску програми, ідентифікатор потоку та шлях до файлу, що необхідно подати на вказаний потік, тип відкриваємого потоку.

Вихідні дані – статус виконаного налаштування.

```
SECURITY_ATTRIBUTES attr;
attr.nLength = sizeof(SECURITY_ATTRIBUTES);
attr.lpSecurityDescriptor = nullptr;
attr.bInheritHandle = TRUE;
```

Рисунок 3.30 – Налаштування атрибутів безпеки файлу

```
HANDLE h = CreateFileW(_fileName, rwMode, 0, &attr, openMode, FILE_ATTRIBUTE_NORMAL, nullptr);
if (h == INVALID_HANDLE_VALUE)
{
        Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. CreateFileW failed error %s. File name = %s\n",
                __LINE__, GetErrorMessage().get(), _fileName);

        return false;
}
```

Рисунок 3.31 – Відкриття файлу із параметрами

```
switch (_handleType)
{
case TestLib::IOHandleType::Input:
        rwMode = GENERIC_READ;
        openMode = OPEN_EXISTING;
        break;
case TestLib::IOHandleType::Output:
case TestLib::IOHandleType::Error:
        rwMode = GENERIC_WRITE;
        openMode = CREATE_ALWAYS;
        break;
        /*case RedirectFileHandleMode::Rewrite:
        rwMode = GENERIC_WRITE;
        openMode = CREATE_ALWAYS;
        break;
        */
default:
        Internal::logger->Error(__FUNCTION__ L"IOHandleType incorrect _handleType = %hhu\n"
                (uint8)_handleType);

        return false;
}
```

Рисунок 3.32 – Задання параметрів відкриття файлу

```
switch (_handleType)
{
case TestLib::IOHandleType::Input:
        SafeCloseHandle(&IoHandles.input);
        IoHandles.input = h;
        break;
case TestLib::IOHandleType::Output:
        SafeCloseHandle(&IoHandles.output);
        IoHandles.output = h;
        break;
case TestLib::IOHandleType::Error:
        SafeCloseHandle(&IoHandles.error);
        IoHandles.error = h;
        break;
default:
        Internal::logger->Error(__FUNCTION__ L"IOHandleType incorrect _handleType = %hhu\n",
                (uint8)_handleType);
        return false;
}

return true;
```

Рисунок 3.33 – Збереження ідентифікатору файлу

Функція applyMemoryLimit (рисунок 3.34) задає обмеження на використання програмою ресурсів системної пам’яті перед її запуском для накладання обмежень відповідно до умов задачі. Вхідні дані – ідентифікатор системної задачі, який буде використано для подальшого запуску програми, список обмежень. Вихідні дані – статус виконаного налаштування.

```
bool applyMemoryLimit()
{
	if (limits.memoryLimitKb <= 0)
	{
		Internal::logger->Warning(L"Can't set not positive memory limit. MemoryLimitKb = %lu\n",
			limits.memoryLimitKb);

		return false;
	}

	JOBOBJECT_EXTENDED_LIMIT_INFORMATION jobExtendedLimits = { 0 };

	if (!QueryInformationJobObject(startupHandles.job, JobObjectExtendedLimitInformation,
		&jobExtendedLimits, sizeof(jobExtendedLimits), nullptr))
	{
		Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. QueryInformationJobObject failed error %s",
			__LINE__, GetErrorMessage().get());

		return false;
	}

	jobExtendedLimits.BasicLimitInformation.LimitFlags |= JOB_OBJECT_LIMIT_PROCESS_MEMORY;
	jobExtendedLimits.ProcessMemoryLimit = static_cast<SIZE_T>(1.5 * limits.memoryLimitKb * 1024); // 1.5X reserve;

	if (!SetInformationJobObject(startupHandles.job, JobObjectExtendedLimitInformation,
		&jobExtendedLimits, sizeof(JOBOBJECT_EXTENDED_LIMIT_INFORMATION)))
	{
		Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. SetInformationJobObject failed error %s. Memory limit = %lu\n",
			__LINE__, GetErrorMessage().get(), limits.memoryLimitKb);

		return false;
	}

	return true;
}
```

Рисунок 3.34 – Функція applyMemoryLimit

Функції applyMandatoryLevel (рисунок 3.35) та applyStartupAttribute (рисунок 3.36) задають обмеження на використання програмою різноманітних системних можливостей, таких як відключення певних системних викликів, задання ізоляції на рівні ОС, заборону запуску інших виконуваних файлів, тощо.

Все це необхідно для забезпечення безпеки процесу тестування та ізоляції виконуваного недовіреного коду користувача. Вхідні дані – ідентифікатор системної задачі, який буде використано для подальшого запуску програми, список обмежень. Вихідні дані – статус виконаного налаштування.

```
bool applyMandatoryLevel(HANDLE hProcessCreationToken)
{
        if (!SetTokenIntegrityLevel(hProcessCreationToken, SECURITY_MANDATORY_LOW_RID))// SECURITY_MANDATORY_UNTRUSTED_RID))
        {
                Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. Can't set container process creation token
                        __LINE__, GetErrorMessage().get());

                SafeCloseHandle(&hProcessCreationToken);

                return false;
        }

        return true;
}
```

Рисунок 3.35 – Функція applyMandatoryLevel

```
DWORD64 mitigationPolicy =
        PROCESS_CREATION_MITIGATION_POLICY_DEP_ENABLE |
        PROCESS_CREATION_MITIGATION_POLICY_DEP_ATL_THUNK_ENABLE |
        PROCESS_CREATION_MITIGATION_POLICY_SEHOP_ENABLE |
        PROCESS_CREATION_MITIGATION_POLICY_HEAP_TERMINATE_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_BOTTOM_UP_ASLR_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_HIGH_ENTROPY_ASLR_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_STRICT_HANDLE_CHECKS_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_WIN32K_SYSTEM_CALL_DISABLE_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_EXTENSION_POINT_DISABLE_ALWAYS_ON;
10
mitigationPolicy |=
        PROCESS_CREATION_MITIGATION_POLICY_FONT_DISABLE_ALWAYS_ON |
        PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON;


if (!UpdateProcThreadAttribute(_startupInfoEx->lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY,
        &mitigationPolicy, sizeof(mitigationPolicy), nullptr, nullptr))
{
        Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. UpdateProcThreadAttribute for PR
                __LINE__, GetErrorMessage().get());

        DeleteProcThreadAttributeList(_startupInfoEx->lpAttributeList);
        free(_startupInfoEx->lpAttributeList);

        return false;
}

10
DWORD childProcessPolicy = PROCESS_CREATION_CHILD_PROCESS_RESTRICTED;
if (!UpdateProcThreadAttribute(_startupInfoEx->lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_CHILD_PROCESS_POLICY,
        &childProcessPolicy, sizeof(childProcessPolicy), nullptr, nullptr))
{
        Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. UpdateProcThreadAttribute for PR
                __LINE__, GetErrorMessage().get());

        DeleteProcThreadAttributeList(_startupInfoEx->lpAttributeList);
        free(_startupInfoEx->lpAttributeList);

        return false;
}
```

Рисунок 3.36 – Функція applyStartupAttribute

Функція applyUIRestrictions (рисунок 3.37) задає обмеження на роботу із різноманітними елементами та функціями користувацького інтерфейсу на рівні ОС, що теж необхідно для забезпечення безпеки процесу тестування. Вхідні дані – ідентифікатор системної задачі, який буде використано для подальшого запуску програми, список обмежень. Вихідні дані – статус виконаного налаштування.

```
bool applyUIRestrictions()
{
        JOBOBJECT_BASIC_UI_RESTRICTIONS jobUILimits = { 0 };

        DWORD restrictionsClass =
                JOB_OBJECT_UILIMIT_EXITWINDOWS |
                JOB_OBJECT_UILIMIT_DESKTOP |
                JOB_OBJECT_UILIMIT_GLOBALATOMS |
                JOB_OBJECT_UILIMIT_DISPLAYSETTINGS |
                JOB_OBJECT_UILIMIT_SYSTEMPARAMETERS |
                JOB_OBJECT_UILIMIT_WRITECLIPBOARD |
                JOB_OBJECT_UILIMIT_READCLIPBOARD |
                JOB_OBJECT_UILIMIT_HANDLES |
                JOB_OBJECT_UILIMIT_ALL;

        jobUILimits.UIRestrictionsClass = restrictionsClass;
        if (!SetInformationJobObject(startupHandles.job, JobObjectBasicUIRestrictions,
                &jobUILimits, sizeof(JOBOBJECT_BASIC_UI_RESTRICTIONS)))
        {
                Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
                        __LINE__, GetErrorMessage().get());

                return false;
        }

        return true;
}
```

Рисунок 3.37 – Функція applyUIRestrictions

Функція applyAffinity (рисунок 3.38) дозволяє програмі використовувати лише одне ядро центрального процесору, оскільки необхідно правильно оцінити час роботи алгоритму без врахування його можливої багатонитковості. Вхідні дані ідентифікатор задачі, вихідні дані статус виконаного налаштування.

```
bool applyAffinity()
{
` > _WIN32_WINNT_VISTA
        if (!SetProcessAffinityUpdateMode(startupHandles.process, 0))
        {
                Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. Can't di
                        __LINE__, GetErrorMessage().get());

                return false;
        }

        ULONG proc = InterlockedIncrement(&current_processor);
        if (!SetProcessAffinityMask(startupHandles.process, 1ull << (proc % processor_count)))
        {
                Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d. Can't up
                        __LINE__, GetErrorMessage().get());

                return false;
        }

        return true;
}
```

Рисунок 3.38 – Функція applyUIRestrictions

Функція LoginUser (рисунок 3.39) виконує налаштування для запуску програми, яку потрібно протестувати, від імені певного користувача, якого було заздалегідь налаштовано із максимальними обмеженнями із метою безпеки. Вхідні дані ідентифікатор задачі, вихідні дані статус виконаного налаштування.

```
HANDLE LoginUser()
{
        HANDLE hToken;

        BOOL res = LogonUserW(userInfo.userName, userInfo.domain, userInfo.domain,
                LOGON32_LOGON_INTERACTIVE, LOGON32_PROVIDER_DEFAULT, &hToken);

        if (res == 0)
        {
                Internal::logger->Error(__FUNCTIONW__ L" failed. Login with username %s failed. Error: %s",
                        userInfo.userName, GetErrorMessage().get());

                return INVALID_HANDLE_VALUE;
        }

        return hToken;
}
```

Рисунок 3.38 – Функція LoginUser

Функція ExecuteProgram – виконує запуск заданої програми з усіма налаштуваннями, що були створені з використанням інших функцій модулю.

Вхідні дані – повний шлях до виконуваного файлу, ключі для його запуску та ідентифікатор системної задачі, що містить налаштування.

Вихідні дані – параметри роботи програми.

Функція CloseIORedirectionHandles використовується для безпечного закриття усіх файлів, що були використані для підміни стандартних потоків введення/виведення перед запуском програми. Вихідний код зображено на рисунку 3.39.

```
void Tester::CloseIORedirectionHandles()
{
        if (IoHandles.input != INVALID_HANDLE_VALUE)
                FlushFileBuffers(IoHandles.input);
        if (IoHandles.output != INVALID_HANDLE_VALUE)
                FlushFileBuffers(IoHandles.output);
        if (IoHandles.error != INVALID_HANDLE_VALUE)
                FlushFileBuffers(IoHandles.error);

        SafeCloseHandle(&IoHandles.input);
        SafeCloseHandle(&IoHandles.output);
        SafeCloseHandle(&IoHandles.error);
}
```

Рисунок 3.39 – Функція CloseIORedirectionHandles

3.4 Висновки

У третьому розділі було проведено аналіз різних мов програмування для розробки Windows орієнтованого сервісу, що виконує функції серверної частини веб-ресурсу, і середовищ програмування, розглянуто питання комп’ютерної безпеки в сценарії запуску користувацького коду на власному сервері з метою його тестування.

У якості мови програмування було обрано мову C# для розробки головних модулів програмного додатку та мови C++ для взаємодії з Windows API. Ці мови є об’єктно-орієнтованими, забезпечують достатньо високу

швидкодію програмного продукту та дозволяють напряму взаємодіяти з системними функціями без додаткових обгорток. Це дозволить не втрачати продуктивність роботи програми.

Також було прийнято рішення використовувати Microsoft Visual Studio в якості IDE, адже дане середовище розробки відповідає всім необхідним вимогам, а головне дозволяє писати код одночасно на мовах C++ та C#. Воно має зручний редактор для написання, редагування та форматування коду. Вбудований препроцесор, що полегшить написання коду завдяки знаходженню синтаксичних помилок під час написання коду програмного додатку. Також Microsoft Visual Studio дозволяє швидко та зручно обирати цільовий формат побудови виконуваного файлу, а також налаштування побудови, такі як розрядність системи, платформу тощо.

Було розроблено головний модуль серверної частини, що є вхідною точкою роботи Windows сервісу, модуль «Tester», що виконує нативні виклики бібліотеки Windows API.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис методів тестування програмного забезпечення

Тестування програмного забезпечення призначено для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Аналіз результатів розробки ПЗ називається статичним тестуванням, а експлуатація програмного продукту називається динамічним тестуванням.

Розглянемо кілька типів тестування, які відрізняються рівнем знань про структуру програмного додатку.

Чорна скринька – це метод тестування, який не передбачає жодного знання про внутрішню архітектуру тестованого компонента або системи. Тестування з використанням техніки чорної скриньки передбачає написання тест-кейсів на основі аналізу функціональної чи нефункціональної специфікації компонента або системи, без знання її внутрішнього будови [23].

Метою цієї техніки є пошук помилок в таких категоріях:

– неправильно реалізовані, відсутні функції;

– помилки в структурах даних чи в доступі до зовнішніх ресурсів;

– помилки інтерфейсу;

– помилки поведінки, ініціалізації та завершення або недостатні характеристики системи.

Таким чином, користувач не має уявлення про структуру та внутрішній устрій системи. Потрібно концентруватися на тому, що програма робить, а не на тому, як вона це робить.

Переваги:

– тестування проводиться з позиції кінцевого користувача;

– той, хто проводить тестування, може не знати мов програмування і може не заглиблюватись у внутрішню реалізацію додатку;

– тест-кейси пишуться лише на основі готової специфікації.

Недоліки:

– тестування обмежене незнанням реалізації внутрішньої реалізації, важко підібрати критичні вхідні значення;

– без чіткої специфікації важко скласти ефективні тест-кейси;

– деякі тести можуть виявитися надмірними.

Біла скринька – це метод тестування програмного забезпечення, який передбачає, що внутрішня структура та реалізація системи відомі для тих, хто проводить тестування і використовуються для створення тестів. Вхідні значення вибираються, ґрунтуючись на знанні коду, який буде їх обробляти[24].

Переваги:

– тестування може виконуватись до створення інтерфейсу користувача;

– можливо підібрати більш ретельні тест-кейси, з покриттям великої кількості шляхів виконання програми;

– покриття саме логіки (вихідний текст) програми.

Недоліки:

– для виконання тестування необхідні спеціальні знання;

– не можливо складати тест-кейси на основі лише специфікації;

– при автоматизації тестування підтримка тестових скриптів може виявитися важкою, особливо при зміні тестованої програми.

Також можлива комбінація підходів білої та чорної скриньки. Такий метод називають сіра або напівпрозора скринька. Тобто, нам відомо внутрішній устрій програми, але саме тестування проводиться лише з позиції користувача.

4.2 Тестування роботи серверної частини веб-ресурсу

Виконаємо тестування методом чорної скриньки, оскільки це допоможе виявити помилки, що не можуть бути покрити автоматизованим тестуванням. А саме перевіримо можливість керування серверною частиною через доступний для користувача інтерфейс. Для запуску та зупинки сервісу використовується менеджер сервісів Windows [25].

Щоб увімкнути сервіс, необхідно відкрити вікно його властивостей (рисунок 4.1), натиснути на кнопку «Запустити», після чого з’явиться індикація процесу запуску (рисунок 4.2). Час запуску залежить не тільки від характеристик серверного комп’ютеру, а й від якості Інтернет доступу до клієнтської частини.

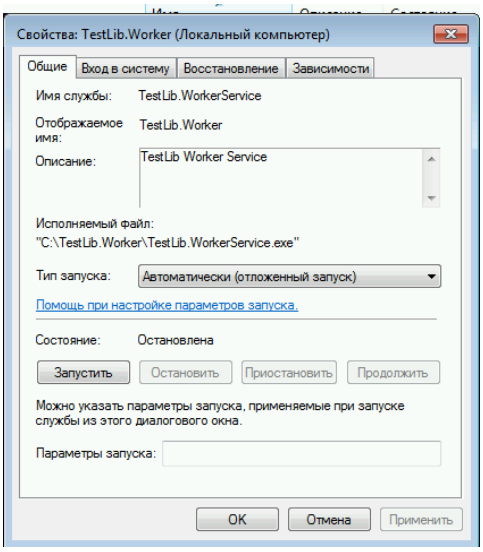


Рисунок 4.1 – Властивості зупиненого сервісу

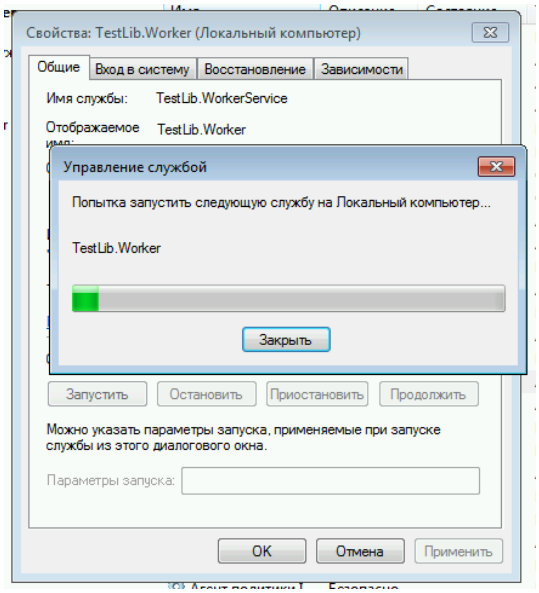


Рисунок 4.2 – Індикація запуску сервісу на комп’ютері

Після успішного запуску стан сервісу буде змінено на «Працює» (рисунок 4.3), а у разі помилки буде відображено відповідне повідомлення. Для

зупинки запущеного сервісу також скористаймося його вікном властивостей. Потрібно натиснути на кнопку «Зупинити», після чого з’явиться індикація процесу зупинки (рисунок 4.4).

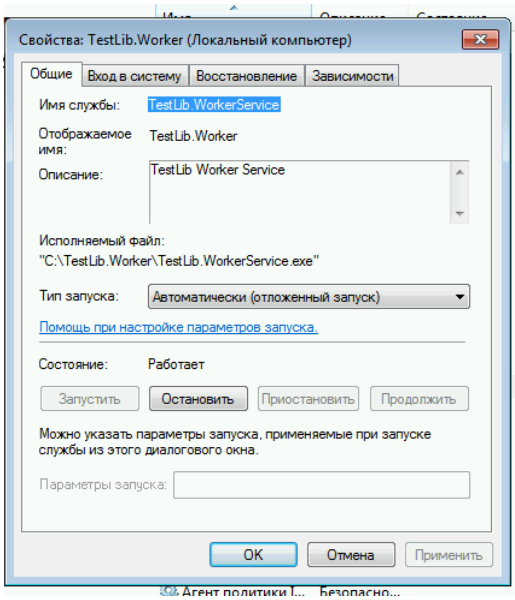


Рисунок 4.3 – Властивості запущеного сервісу

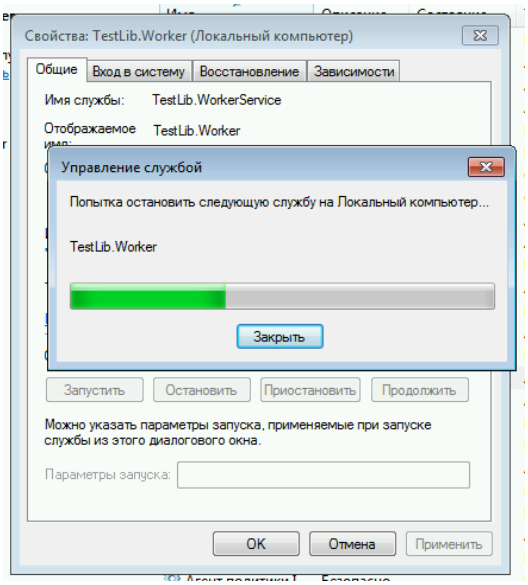


Рисунок 4.4 – Властивості зупиненого сервісу

Іншим інтерфейсом взаємодії є файл логу, у якому міститься уся інформація про поточну роботу серверної частини.

Тому, використовуючи клієнтську частину, додамо новий розв’язок для задачі (рисунок 4.5), який буде перевірений запущеним сервісом серверної частини.

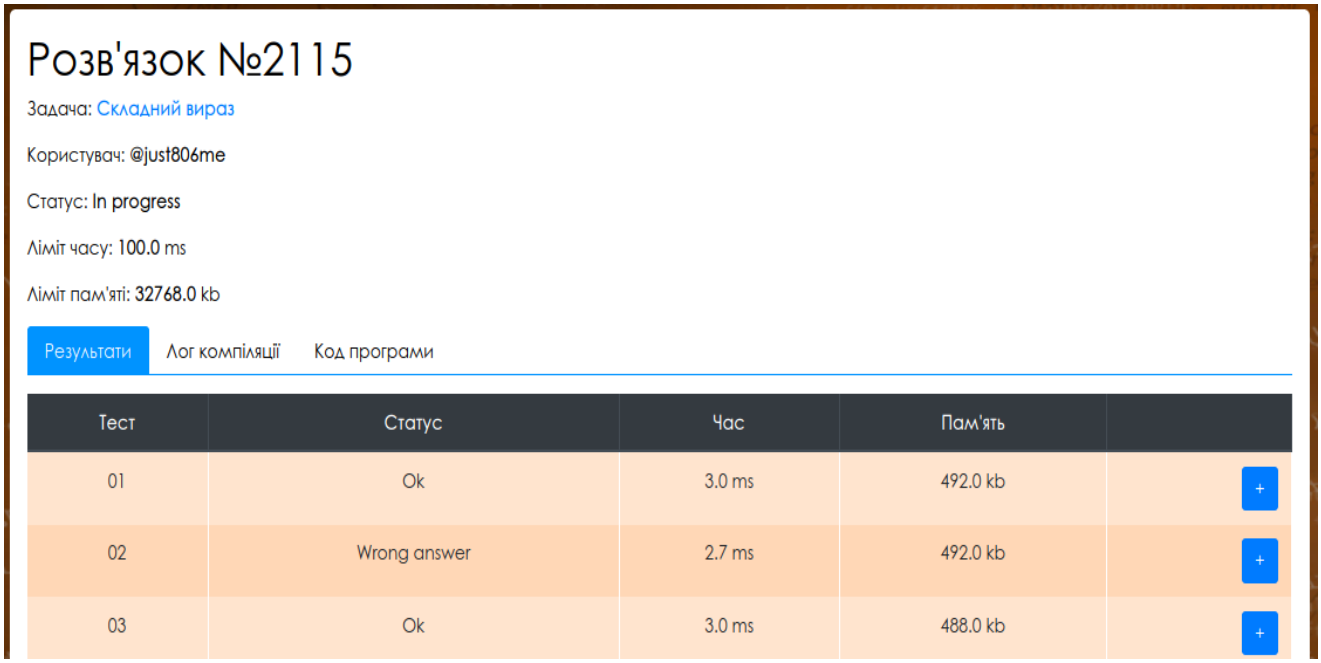


Рисунок 4.5 – Розв’язок задачі у процесі перевірки

Розглянемо результати, що будуть записані у файл логу та інформацію, що буде показана користувачу.

Як видно з рисунку 4.6, серверна частина отримала відправлення з номером 2115 на перевірку.

Серверна частина почала процес завантаження його вихідного коду та тестів до задачі.

Результат тестування відображено на рисунку 4.6.

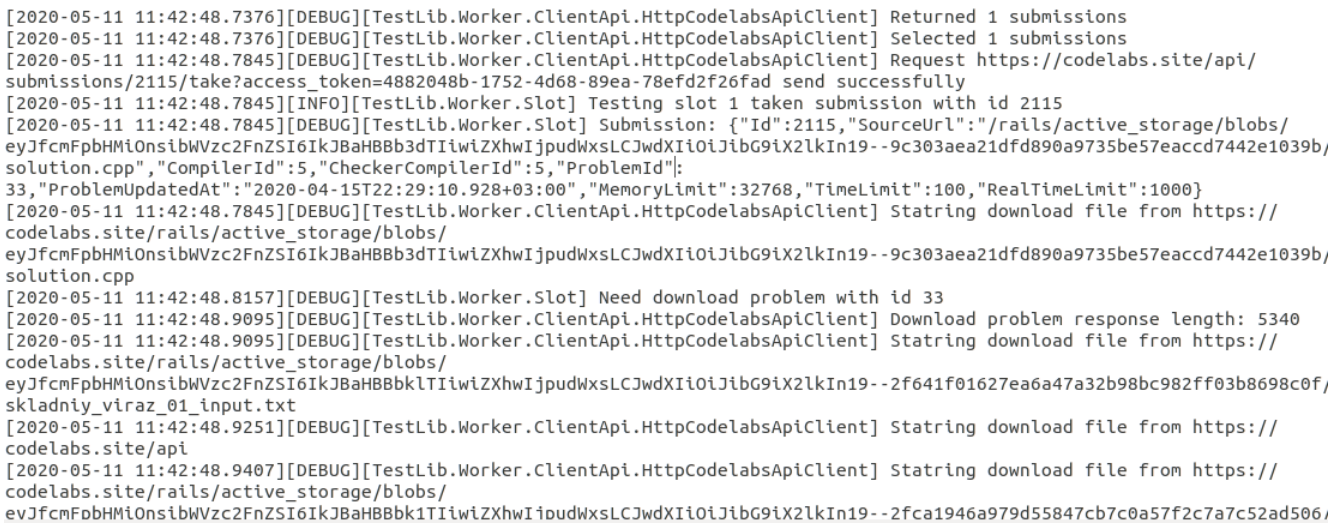


Рисунок 4.6 – Лог початку тестування рішення

На рисунку 4.7 видно інформацію отриману від чекеру про результати перевірки перших двох тестів. Тест номер 01 пройшов перевірку успішно, а тест номер 02 завершився з помилкою. Ці відомості збігаються з зображеними на сайті клієнтської частини (рисунок 4.5).

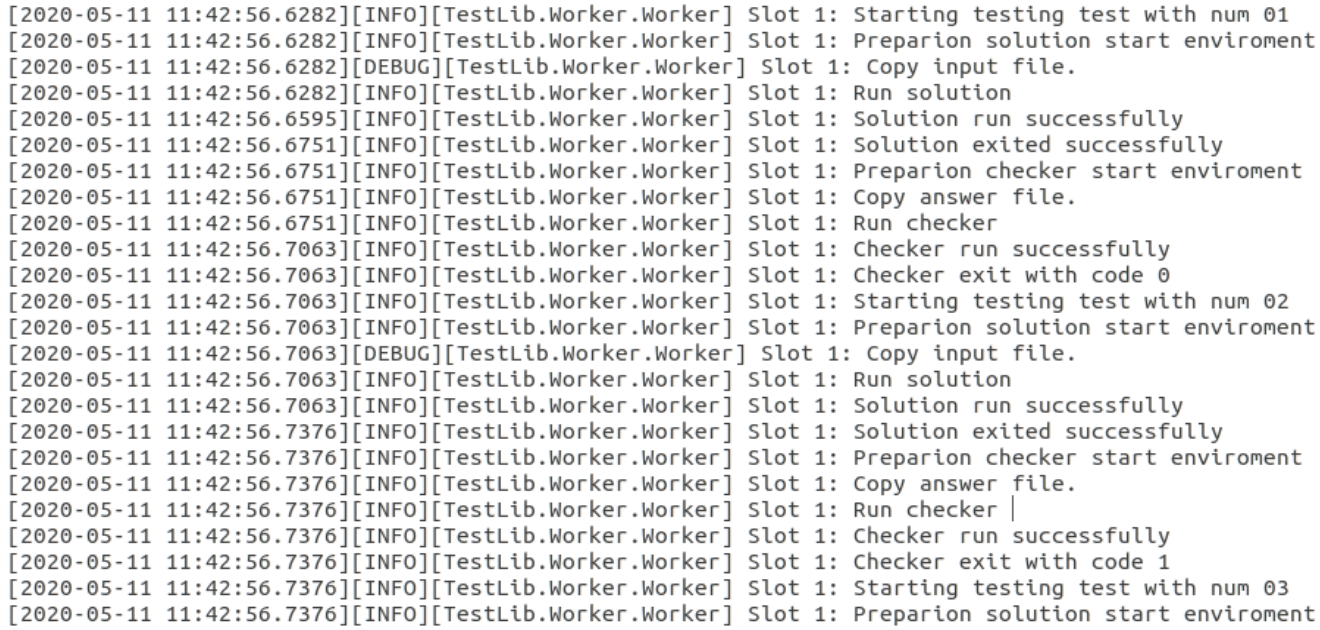


Рисунок 4.7 – Лог тестування перших двох тестів

Усі зроблені перевірки показують коректність роботи програми, відповідність та цілісність даних між клієнтською та серверною частинами. Компіляція, запуск та перевірка рішень виконуються успішно. Отже, розроблений програмний продукт виконує поставленні задачі.

4.3 Висновки

Одним з важливих етапів розробки додатку є тестування, так як воно є гарантією якості розроблюваного додатку.

У четвертому розділі було проаналізовано основні методи тестування, описано методику тестування програмного додатку методом «чорної скриньки». Наведено результати тестування за таким методом, що не виявили помилок та підтвердили повну працездатність програмного продукту та відповідність технічному завданню.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету: Войтко Вікторія Володимирівна (к.т.н., доц. кафедри ПЗ ВНТУ), Черноволик Галина Олександрівна (к.т.н., доц. кафедри ПЗ ВНТУ), Бурбело Сергій Михайлович (к.т.н., ст.в. кафедри ПЗ ВНТУ). Для проведення технологічного аудиту було використано таблицю 5.1, в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу розробки.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 5.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

У таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Войтко В.В	Бурбело С.М.	Черноволик Г.О.
	Бали, виставлені експертами:		
1	4	4	4
2	2	2	2
3	3	2	3
4	3	4	3
5	2	2	2
6	4	3	3
7	3	3	3
8	4	4	4
9	2	2	1
10	3	2	3
11	3	4	4
12	4	3	3
Сума балів	$СБ_1$=37	$СБ_2$=35	$СБ_3$=35
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{37+35+35}{3} = 35.6$		

Розрахована нами на основі висновків експертів середньоарифметична сума балів склала 35.6 бали. Згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Методи і програмні засоби системи тренування і оцінювання робіт зі спортивного програмування, що розробляються в магістерській роботі, будуть цікаві школам та університетам, що приймають участь у міжнародних олімпіадах зі спортивного програмування, для підготовки учнів та студентів і навчанню їх основним принципам вирішення олімпіадних задач. Враховуючи особливості цільової аудиторії та освітню спрямованість проекту, дана розробка є некомерційною.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий	Новий		
Точність обмежень для різних мов програмування, %	60	90	1,5	30%
Детальність результатів тестування, бали	5	8	1,6	30%
Середній час відповіді на запит *(менше – краще)*, мс	500	280	1,79	20%
Кількість підтримуваних мов програмування, шт	24	6	0,25	10%
Використання ресурсів комп’ютера, %	20	20	1	10%

Порівняємо нову розробку, що розробляється в магістерській роботі, з аналогом, який існує на ринку.

В якості аналога для розробки було обрано інтернет-портал організаційно-методичного забезпечення дистанційних олімпіад з

програмування для обдарованої молоді навчальних закладів України – E-Olymp. Основним недоліком аналога є використання однакових обмежень по часу та пам’яті для різних мов програмування й недостатня детальність результатів тестування, що ускладнює навчальний процес.

У розробці дані проблеми вирішуються за рахунок введення вагових коефіцієнтів та модернізації програми «чекера» для збереження детального ходу тестування.

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{90}{60} = 1{,}5;$$

$$q_2 = \frac{8}{5} = 1{,}6;$$

$$q_3 = \frac{500}{280} = 1{,}79;$$

$$q_4 = \frac{6}{24} = 0{,}25;$$

$$q_5 = \frac{20}{20} = 1.$$

Відносний рівень якості нової розробки визначаємо за формулою 5.3:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i \tag{5.3}$$

$$К_{я.в.} = 1{,}5 \cdot 0{,}3 + 1{,}6 \cdot 0{,}3 + 1{,}79 \cdot 0{,}2 + 0{,}25 \cdot 0{,}1 + 1 \cdot 0{,}1 = 1{,}413$$

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою 5.4:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів;

$I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5):

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Нei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{5.5}$$

де $P_{Нei}$, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Зважаючи на некомерційний характер розробки і аналогу, приймемо індекс економічних параметрів за 1.

$$I_{е.п.} = 1;$$

$$К = \frac{1{,}413}{1} = 1{,}413.$$

Виходячи з розрахунків можна зробити висновок, що нова розробка буде більш конкурентоспроможною, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_0$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою 5.6:

$$З_0 = \frac{М}{Т_р} * t(грн) \qquad (5.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_Р$ – число робочих днів в місяці; приблизно $T_Р \approx 21…23$ дні;

t – число робочих днів роботи дослідника.

Для розробки програмних засобів системи тренування і оцінювання робіт зі спортивного програмування необхідно залучити двох програмістів: розробник C++ з посадовим окладом 120000 грн та розробник C# з посадовим окладом 125000 грн; одного QA-інженера з посадовим окладом 115000 грн; одного системного інженера з посадовим окладом 135000 грн [33], а також керівника роботи.

Кількість робочих днів у місяці складає 22, кількість робочих днів кожного програміста складає 33, кількість робочих днів QA-інженера складає 10, кількість робочих днів системного інженера складає 5.

Зведемо сумарні розрахунки до таблиці 5.6.

Таблиця 5.6 – Заробітна плата спеціалістів для розробки програмних засобів

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	12000	545,5	5	2727
Розробник С++	120000	5454,6	33	180000
Розробник С#	125000	5681,8	33	187500
QA-інженер	115000	5227,3	10	52273
Системний інженер	135000	6136,4	5	30682
Всього				453182

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10-12% від основної заробітної плати робітників за формулою 5.7.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = \frac{(З_о + З_р) * Н_{доп}}{100\%} \quad (5.7)$$

$$З_д = 0{,}10 \cdot 453182 = 45318{,}2(грн)$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.8):

$$Н_{ЗП} = \frac{(З_о + З_д) * \beta}{100} \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = \frac{(453182 + 45318{,}2) \cdot 22}{100} = 109670(грн)$$

4. Витрати на матеріали М та комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою 5.9:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \qquad (5.9)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.7 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	140	1	140
Ручка	20	1	20
CD-диск	12	1	12
Флешка	155	1	155
Всього			327
З врахуванням коефіцієнта транспортування			359,7

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. Для нової розробки використовувались безкоштовні програмні засоби.

6. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо за формулою 5.10.

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.10}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувалися чотири персональних комп'ютери кожен вартістю 45000 грн.

$$\text{А} = \frac{45000 \cdot 4}{2 \cdot 12} = 7500$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що використовуються з технологічною метою на проведення досліджень, та розраховуються за формулою 5.11.

$$\text{В}_\text{е} = \sum_{\text{i}=1}^{n} \frac{W_{yt} \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_{\text{впі}}}{\eta_\text{i}} \tag{5.11}$$

де W_{yt} – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Для написання магістерської роботи використовувалися чотири персональних комп'ютери для яких розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}5 \cdot 250 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} + \frac{0{,}5 \cdot 250 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} + \frac{0{,}5 \cdot 250 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} + \frac{0{,}5 \cdot 250 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} = 1281{,}25$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати $В_{нзв}$ охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати $В_{нзв}$ можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, що розраховуються за формулою 5.12:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 453182 \cdot \frac{100}{100\%} = 453182(грн)$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$\text{В} = 453182 + 45318{,}2 + 109670 + 359{,}7 + 7500 + 1281{,}25 + 453182 = 1070493{,}15(\text{грн})$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою 5.13:

$$\text{ЗВ} = \frac{\text{В}}{\eta}, \tag{5.13}$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії впровадження, то коефіцієнт $\eta = 0{,}9$. Звідси:

$$\text{ЗВ} = \frac{1070493{,}15}{0{,}9} = 1189436{,}83(\text{грн.})$$

5.3 Оцінювання важливості та наукової значимості науково-дослідної роботи

Застосовані методи і розроблені програмні засоби системи тренування і оцінювання робіт зі спортивного програмування можуть бути цікаві профільним школам та закладам вищої освіти, що приймають участь у міжнародних олімпіадах зі спортивного програмування, а також зацікавленим студентам. Враховуючи особливості цільової аудиторії та освітню спрямованість проекту, магістерська кваліфікаційна робота є науково-дослідного й некомерційного характеру. Тому, розрахуємо комплексний показник $К_р$ рівня важливості та наукової значимості науково-дослідної роботи за формулою 5.14:

$$К_р = \frac{I^n \cdot T_c \cdot R}{B \cdot t} \tag{5.14}$$

де I – коефіцієнт важливості роботи, I = 2...5;

n – коефіцієнт використання результатів роботи; n = 0, коли результати роботи не будуть використовуватись; n = 1, коли результати роботи будуть використовуватись частково; n = 2, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках; n = 3 , коли результати роботи можуть використовуватись навіть без проведення дослідно-конструкторських розробок;

T_c – коефіцієнт складності роботи, T_c = 1...3;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то R = 4 ; якщо результати роботи відповідають відомому рівню, то R = 3 ; якщо нижче відомих результатів, то R = 1;

B – вартість науково-дослідної роботи, тис. грн;

t – час проведення дослідження, років.

Визначимо показники I, n, T_c, R, B, t на основі нормативів. У результаті підрахунків отримаємо наступне значення показника рівня науково-дослідної роботи (вираз 5.15):

$$К_р = \frac{4^2 \cdot 3 \cdot 4}{1070{,}49315 \cdot \frac{2}{12}} = 1{,}08 \qquad (5.15)$$

Оскільки розрахований показник рівня науково-дослідної роботи $К_р > 1$, то науково-дослідну роботу можна вважати ефективною з високим науковим, технічним і економічним рівнями.

5.4 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу методів та програмних засобів системи тренування і оцінювання робіт зі спортивного програмування, який є на рівні вище середнього. При порівнянні нової розробки з аналогом виявлено, що вона є якіснішою і більш конкурентоспроможною в порівнянні з аналогом, а також є кращою по технічним і економічним показникам.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 1070493,15 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 1189436,83 грн.

Було розраховано показник рівня важливості та наукової значимості науково-дослідної роботи $К_р$. За його значенням науково-дослідну роботу можна вважати ефективною з високим науковим, технічним і економічним рівнями.

Мова про те, що вкладені у проект інвестиції окупляться, вестися не може, тому що розробка (проект) є освітньою, а результати її впровадження не передбачають отримання прибутку. Розробка має на меті надання вільного доступу до знань та інструментів для навчання, тому її фінансування може бути цікавим для профільних закладів вищої освіти та шкіл.

ВИСНОВКИ

У магістерський кваліфікаційній роботі було розроблено серверну частину веб-ресурсу для розміщення і автоматичної перевірки олімпіадних задач зі спортивного програмування. Такий веб-ресурс може забезпечити як автоматичне проведення олімпіади через мережу Інтернет, так і дозволяє тренуватись до майбутніх олімпіад, відправляючи на перевірку рішення до доступних задач.

Було проаналізовано стан питання оцінювання робіт на сучасних олімпіадах. Розглянуто основні аналоги пропонованої системи, визначено їх особливості та недоліки, порівняно з власним програмним продуктом, за рахунок чого визначено актуальність розробки. Спроектовано структуру роботи програми, розділено її на модулі, а також розроблено блок-схеми необхідних алгоритмів. Для вибору засобів реалізації системи проведено варіантний аналіз, за результатом якого обрано операційну систему Windows, мову програмування C# для створення Windows сервісу та мову C++ для виконання native-викликів з Windows API через бібліотеку C++/CLI.

У результаті роботи було виконано такі задачі: розроблено загальну модель роботи серверної частини; розроблено алгоритм роботи Windows сервісу; розроблено алгоритм роботи модуля безпечного тестування «Tester»; розроблено метод використання процесорних тактів для оцінки часу тестування роботи програми; розроблено інтерфейс конфігурації сервісу; проведено тестування програмного продукту. Розроблено комбінований метод компіляції для використання бібліотеки libkrun, для отримання програми, що автоматично запускається у легковісній віртуальній машині, яка має більший рівень ізоляції, ніж контейнери, проте не потребує таких ресурсів, як повноцінна віртуальна машина

Для тестування було проаналізовано існуючі методи і підходи, виконано перевірку роботи веб-ресурсу за їх допомогою, що довело повну працездатність розробленої системи та її відповідність поставленому технічному завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Нікітіна Н. С. Олімпіада як науковий захід для студентів. – К.: ЦУЛ, 2015. 175 с.

2 Войтко В.В., Бевз С. В., Бурбело С. М., Кузнєцов Л. Г., Костюк К. А. Розробка веб-ресурсу для розміщення та автоматизованої перевірки олімпіадних. – XLIX Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, 2020. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9620/8008 (дата звернення 30.11.2021)

3 Бурбело С. М., Костюк К. А., Кузнєцов Л. Г. Особливості використання процесорних тактів при оцінюванні часу роботи програм. – Конференція «Молодь у світі сучасних технологій», 2020. URL: http://conference.ho.ua/index.php (дата звернення 30.11.2021)

4 Войтко В. В., Коваленко О. О., Бевз С. В., Бурбело С. М., Кузнєцов Л. Г., Костюк К. А. Використання технологій віртуалізації для запуска недовіреного коду на сервері. – всеукраїнська науково-практична інтернет-конференція «електронні інформаційні ресурси: створення, використання, доступ», 2021. URL: http://konferencia.voipopp.vn.ua/ (дата звернення 30.11.2021)

5 Самойлов В. Way of problemsetter. 2014. URL: https://codeforces.com/blog/entry/12642?locale=en (дата звернення 30.11.2021)

6 Система ejudge. URL: https://ejudge.ru/wiki/index.php/Система_ejudge (дата звернення 30.11.2021)

7 E-Olymp. URL: https://www.e-olymp.com/uk/pages/about (дата звернення 30.11.2021)

8 Перша в Україні школа мистецтва розв'язування алгоритмічно складних задач. URL: http://algotester.com (дата звернення 30.11.2021)

9 Інформаційне забезпечення. – 2019.. URL: https://uk.wikipedia.org/wiki Інформаційне_забезпечення (дата звернення 30.11.2021)

10 Protocol Buffers. URL: https://developers.google.com/protocol-buffers/docs/reference/overview (дата звернення 30.11.2021)

11 Майданюк В. П. Кодування та захист інформації. – Вінниця, 2009. 164 с.

12 Let's Encrypt - Free SSL/TLS Certificates. URL: https://letsencrypt.org/ (дата звернення 30.11.2021)

13 Хошаба О. М. Операційні системи комп’ютерних мереж. Навчальній посібник. – Вінниця: ВНТУ, 2004. 115 с.

14 Chirammal H., Mukhedkar P., Dakic V. Mastering KVM Virtualization/ – Packt, 2020. 686 с.

15 Спортивне програмування. Державний університет телекомунікацій. 2017. URL: http://www.dut.edu.ua/ua/news-1-1009-4178-sportivne-programuvannya_kafedra-inzhenerii-programnogo-zabezpechennya (дата звернення 30.11.2021)

16 A dynamic library providing Virtualization-based process isolation capabilities. URL: https://github.com/containers/libkrun (дата звернення 30.11.2021)

17 Kata Containers - Open Source Container Runtime Software. URL https://katacontainers.io/ (дата звернення 30.11.2021)

18 Comparison of platform virtualization software. URL: https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software (дата звернення 30.11.2021)

19 KVM Forum 2021: libkrun: More than a VMM. URL: https://kvmforum2021.sched.com/event/ke36/libkrun-more-than-a-vmm-in-dynamic-library-form-sergio-lopez-pascual-red-hat (дата звернення 30.11.2021)

20 Хогенсон Г. C++/CLI: язык Visual C++ для среды .NET. 2007. 464 с.

21 Troelsen A., Japikse P. Pro C# 7 With .NET and .NET Core. – 2017. 1372 с.

22 Плагин XML Tools для Notepad++. URL: https://habr.com/en/sandbox/37354/ (дата звернення 30.11.2021)

23 Шилдт Г. Самоучитель C++. – СПб: БХВ-Петербург, 2003. 687 с.

24 Шилдт Г. Полный справочник по C#. – М.: Издательский дом "Вильяме", 2004. 752 с.

25 Fain Y. Java Programming: 24-Hour Trainer. – 2015. 624 с.

26 Chandrasekara C. Visual Studio 2019 Cookbook. – 2020. 710 с.

27 Russell J., Cohn R. Monodevelop. – Book on Demand, 2012. 166 с.

28 Tutorials RAD Studio. URL: http://docwiki.embarcadero.com/RADStudio/Rio/en/Tutorials (дата звернення 30.11.2021)

29 Gregoire M. Professional C++. – 2018. 1184 с.

30 Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ. – San Francisco: Wiley, 2019. 272 с.

31 Криспин Л., Грегори Д. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. М.: Вильямс, 2018. 464 с.

32 Morin R. Programming Windows(tm) Services: Implementing Application Servers / Randy Morin. – San Francisco: Wiley, 2000. 416 с.

33 Статистика зарплат програмістів, тестувальників і PM в Україні | DOU. URL: https://jobs.dou.ua/salaries/ (дата звернення 30.11.2021)

ДОДАТКИ

Додаток А – Технічне завдання

ВНТУ

УЗГОДЖУЮ
Директор КЗ «ВТЛ»
_____ О. М. Козяр
"____" _____ 2021 р.

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
_____ О. Н. Романюк
"____" _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу
зі спеціальності 121 «Інженерія програмного забезпечення»
студенту групи 1ПІ-20м Костюку Костянтину Андрійовичу

1.1 Найменування та галузь застосування

Розробка серверної частини веб-ресурсу для розміщення і автоматизованої перевірки олімпіадних задач.

1.2 Підстава для проведення робіт

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 277 від « 24 » вересня 2021 р.

1.3 Мета та призначення роботи

Метою роботи є підвищення якості систем автоматичної перевірки програм розв’язків задач зі спортивного програмування та оцінювання параметрів їх роботи за рахунок розробки та програмної реалізації спеціалізованого програмного забезпечення для веб-ресурсу, зокрема серверної частини, що дозволить підвищити надійність перевірки та оцінювання робіт.

Об’єктом дослідження є процес автоматичної перевірки розв’язків задач зі спортивного програмування, безпечної компіляції та запуску програм і точного оцінювання параметрів їх роботи.

Предметом дослідження є методи та засоби розробки серверної частини веб-ресурсу для автоматичної перевірки програм та оцінювання параметрів їх роботи.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

– розробка загальної моделі роботи серверної частини;

– розробка алгоритму роботи Windows сервісу;

– розробка алгоритму роботи модуля безпечного тестування «Tester»;

– розробка методу покращення точності оцінки часу тестування роботи програми;

– удосконалення методу перевірки завдань на боці серверу, який використовує легковісні віртуальні машини для підвищення безпеки;

– розробка інтерфейсу конфігурації сервісу;

– тестування програмного продукту.

1.4 Технічні вимоги

– Операційна система Microsoft Windows Server 2019 build 18363 64-біт і вище.

– Оперативна пам’ять системи не менше 4 ГБ.

– Пам’ять на жорсткому диску не менше 16 ГБ.

1.5 Перелік технічної документації, що пред’являється по закінченню робіт:

– технічне завдання;

– технічне обґрунтування;

– лістинги модулів програмного додатку та бібліотеки.

1.6 Стадії і етапи розробки

Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	15.09 – 20.09.2021
Розробка архітектури та алгоритмів роботи системи	20.09 – 02.10.2021
Вибір середовища та мови розробки	02.10 – 10.10.2021
Розробка програмного продукту	10.10 – 25.10.2021
Тестування роботи системи	25.10 – 05.11.2021
Економічна частина	05.11 – 20.11.2021
Оформлення матеріалів до захисту МКР	20.11 – 30.11.2021

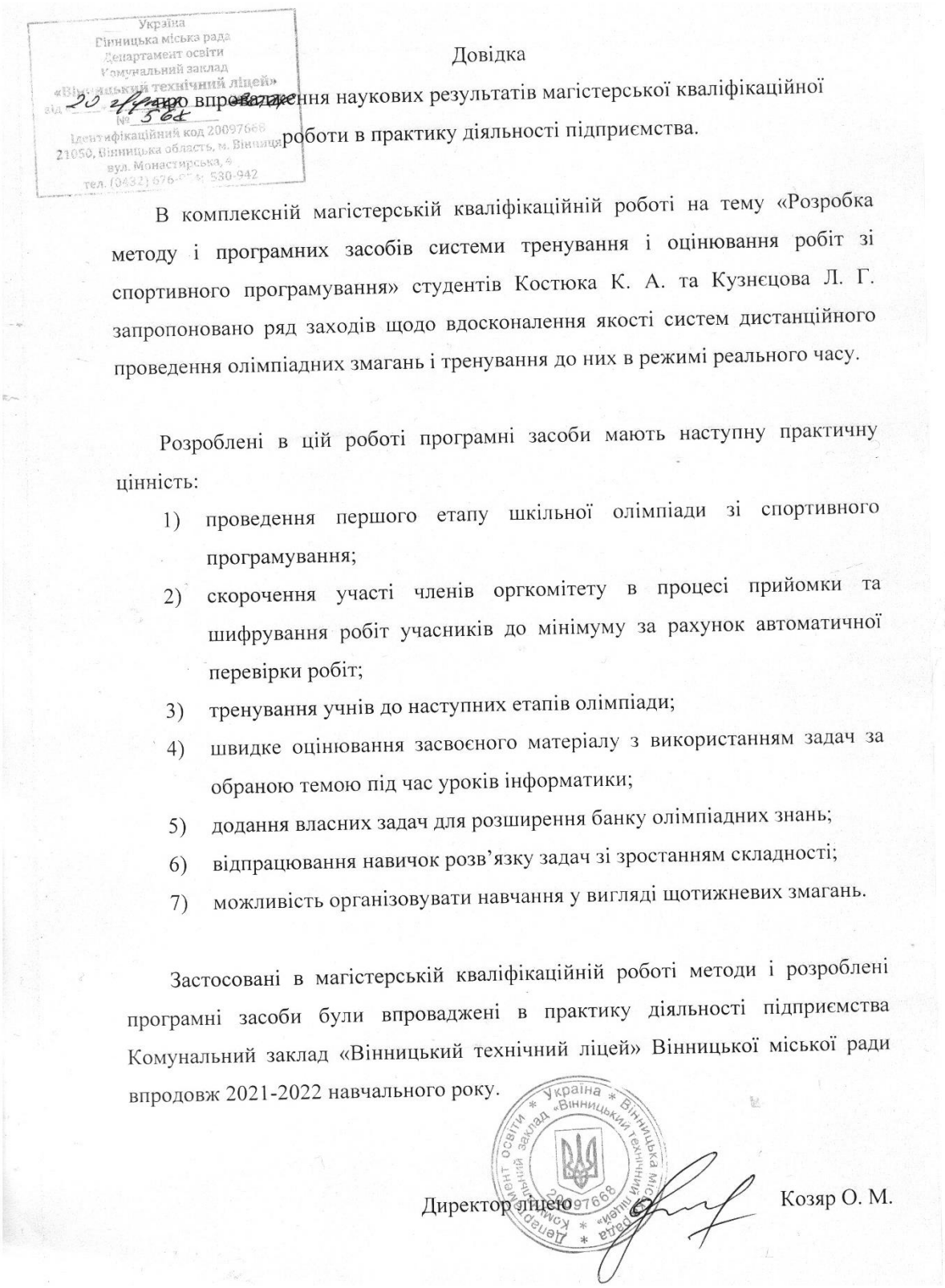
1.7 Порядок контролю і приймання

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

Завдання отримав _____ (підпис) Костюк К. А. (прізвище та ініціали)

Науковий керівник _____ (підпис) Войтко В. В. (прізвище та ініціали)

Додаток Б – Акт впровадження



Україна
Вінницька міська рада
Департамент освіти
Комунальний заклад
«Вінницький технічний ліцей»
від 20.12.2021 №564
Ідентифікаційний код 20097668
21050, Вінницька область, м. Вінниця
вул. Монастирська, 9
тел. (0432) 676-67-74, 530-942

Довідка

про впровадження наукових результатів магістерської кваліфікаційної роботи в практику діяльності підприємства.

В комплексній магістерській кваліфікаційній роботі на тему «Розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування» студентів Костюка К. А. та Кузнєцова Л. Г. запропоновано ряд заходів щодо вдосконалення якості систем дистанційного проведення олімпіадних змагань і тренування до них в режимі реального часу.

Розроблені в цій роботі програмні засоби мають наступну практичну цінність:

1) проведення першого етапу шкільної олімпіади зі спортивного програмування;
2) скорочення участі членів оргкомітету в процесі прийомки та шифрування робіт учасників до мінімуму за рахунок автоматичної перевірки робіт;
3) тренування учнів до наступних етапів олімпіади;
4) швидке оцінювання засвоєного матеріалу з використанням задач за обраною темою під час уроків інформатики;
5) додання власних задач для розширення банку олімпіадних знань;
6) відпрацювання навичок розв'язку задач зі зростанням складності;
7) можливість організовувати навчання у вигляді щотижневих змагань.

Застосовані в магістерській кваліфікаційній роботі методи і розроблені програмні засоби були впроваджені в практику діяльності підприємства Комунальний заклад «Вінницький технічний ліцей» Вінницької міської ради впродовж 2021-2022 навчального року.

Директор ліцею Козяр О. М.

Рисунок Б.1 – Акт впровадження результатів магістерської кваліфікаційної роботи

Додаток В – Протокол перевірки роботи на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування. Частина 1. Сервер оцінювання робіт**.

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 20м

Науковий керівник: к.т.н. доц. Войтко В. В..

Unicheck	
Оригінальність	**91,1 %**
Схожість	8,9 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлена з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методу і програмних засобів системи тренування і оцінювання робіт зі спортивного програмування. Частина 1. Сервер оцінювання робіт».

Автор _____ Костюк Костянтин Андрійович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток Г – Лістинг коду

Файл Application.cs

```
using NLog;
using System;
using System.Collections.Specialized;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net;
using System.Reflection;
using TestLib.Worker.ClientApi;
using TestLib.Worker.ClientApi.Models;

namespace TestLib.Worker
{
  public static class StringConvertionExtension
  {
    public static int ToInt32OrDefault(this string value, int defaultValue = 0)
=>
       int.TryParse(value, out int result) ? result : defaultValue;

    public static uint ToUInt32OrDefault(this string value, uint defaultValue =
0) =>
       uint.TryParse(value, out uint result) ? result : defaultValue;
  }

  internal enum CheckUpdateStatus
  {
    None,
    Ok,
    Restart,
    Failed,
  }

  internal sealed class Application
  {
    private static Application application;
    public static Application Get() => application ?? (application = new
Application());

    private Application()
    {
      logger = LogManager.GetCurrentClassLogger();
      logger.Info("Initiated the creation of application. Version: {0}",
GetVersion());

      LoggerManaged = new LoggerManaged();
      apiClient = new HttpCodelabsApiClient();

      NameValueCollection config = ConfigurationManager.AppSettings;
      Configuration = new Configuration(config);
```

```
            FileProvider = new FileProvider(Configuration.FileCacheFolder);
            Compilers = new CompilerManager(Configuration.CompilersConfigFolder);

            Problems = new ProblemCache(Configuration.ProblemsCacheSize);
            RequestMessages                              =                              new
BlockingQueue<RequestMessage>(Configuration.ResultSendingCacheSize);

            workerTasks = new WorkerTaskManager();
          }

          public bool Init()
          {
            LoggerManaged.InitNativeLogger(new
LoggerManaged.LogEventHandler(LogManager.GetLogger("Native").Log));

            logger.Debug("Current directory is {0}", Directory.GetCurrentDirectory());
            logger.Debug("AppData              folder              is              {0}",
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
            logger.Debug("Current              user              is              {0}",
System.Security.Principal.WindowsIdentity.GetCurrent().Name);


            FileProvider.Init();

            if (!Compilers.Init())
            {
              logger.Error("Compilers initialization failed");

              return false;
            }

            if (Configuration.WorkerId == Guid.Empty)
            {
              if (!signUp())
              {
                logger.Error("Application sign up failed");
                return false;
              }
            }

            var status = apiClient.SignIn(Configuration.WorkerId);
            if (status == UpdateWorkerStatus.LoginIncorrect)
            {
              logger.Error("Application sign in failed. Worker id is incorrect. Trying
re-sign up");

              if (!signUp())
              {
                logger.Error("Application sign up failed");
                return false;
              }
            }
            else if (status != UpdateWorkerStatus.Ok)
            {
              logger.Error("Application sign in failed");
              return false;
```

```
            }

            if (!workerTasks.Init(Configuration))
            {
              apiClient.SignOut(Configuration.WorkerId);
              return false;
            }

            return true;
          }

          public Version GetVersion()
          {
            var assembly = Assembly.GetExecutingAssembly();
            var fvi = FileVersionInfo.GetVersionInfo(assembly.Location);

            return new Version(fvi.ProductMajorPart, fvi.ProductMinorPart,
              fvi.ProductBuildPart, fvi.ProductPrivatePart);
          }

          public void Start()
          {
            workerTasks.Start();
          }
          public void Stop()
          {
            workerTasks.Stop();
          }
          public void Restart()
          {
            Stop();
            Start();
          }
          public void Status() => workerTasks.PrintStatus();
          public void End()
          {
            Stop();
            workerTasks.End();
            LoggerManaged.Destroy();
            apiClient.SignOut(Configuration.WorkerId);
          }

          public CheckUpdateStatus Update()
          {
            using (WebClient client = new WebClient())
            {
              try
              {
                string                          stringVersion                          =
client.DownloadString(Configuration.Update.LatestVersionUrl).Replace("\"", "");
                if (!Version.TryParse(stringVersion, out var version))
                {
                  logger.Warn("Can't parse latest version {0} from update server {1}.
Update not available.",
                    stringVersion, Configuration.Update.LatestVersionUrl);
```

```
                return CheckUpdateStatus.Failed;
              }

              Version current = GetVersion();
              if (version > current)
              {
                var msg = string.Format("Latest version {0}, current version {1}:
update is necessary.", version, current);

                logger.Info(msg);
                Console.WriteLine(msg);

                var args = new string[]
                {
                  "TestLib.Worker.Updater.exe",
                  Process.GetCurrentProcess().Id.ToString(),
                  Directory.GetCurrentDirectory(),
                  AppDomain.CurrentDomain.FriendlyName,
                };
                Process.Start("TestLib.Worker.Updater.exe", string.Join(" ", args));

                return CheckUpdateStatus.Restart;
              }
              else
              {
                var msg = string.Format("Latest version {0}, current version {1}:
update is not necessary.", version, current);

                logger.Info(msg);
                Console.WriteLine(msg);

                return CheckUpdateStatus.Ok;
              }
            }
            catch (Exception ex)
            {
              logger.Error(ex, "Error while update.");

              return CheckUpdateStatus.Failed;
            }
          }
        }
        private bool signUp()
        {
          WorkerInformation wi = new WorkerInformation(Configuration.WorkerName,
            Dns.GetHostAddresses(Dns.GetHostName()).Select(ip                    =>
ip.ToString()).ToArray());

          return (Configuration.WorkerId = apiClient.SignUp(wi)) != Guid.Empty;
        }
        #region Variables
        public FileProvider FileProvider { get; private set; }
        public ProblemCache Problems { get; private set; }
        public BlockingQueue<RequestMessage> RequestMessages { get; private set; }
        public CompilerManager Compilers { get; private set; }
        public Configuration Configuration { get; private set; }
```

```
        public LoggerManaged LoggerManaged { get; private set; }
        private WorkerTaskManager workerTasks;
        private IApiClient apiClient;
        private Logger logger;
        #endregion
      }
    }
```

Файл Configuration.cs

```
    using System;
    using System.Reflection;
    using System.Collections.Concurrent;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Collections.Specialized;
    using System.Diagnostics;

    namespace TestLib.Worker
    {
      internal class Configuration
      {
        public class UpdateConfiguration
        {
          public string LatestVersionUrl;
          public string LatestProgramUrl;
        }


        private Guid _workerId;

        public Configuration(NameValueCollection config)
        {
          Update = new UpdateConfiguration();

          WorkerName = config.Get("worker_name") ?? "TestLib.Worker";

          TestingWorkDirectory = config.Get("workarea") ?? ".\\workarea\\";

          FileCacheFolder = config.Get("cache_folder") ?? ".\\cache\\";
          CompilersConfigFolder     =     config.Get("compilers_config_folder")     ??
".\\compilers\\";
          ProblemsCacheSize = config.Get("problems_cache_size").ToUInt32OrDefault(1);
          ResultSendingCacheSize                                                    =
config.Get("result_sending_cache_size").ToUInt32OrDefault(2048);

          BaseAddress       =       new       Uri(config.Get("base_address")       ??
"http://localhost:8080/");
          BaseApiAddress = new Uri(BaseAddress, config.Get("api_path") ?? "/api");
          ApiAuthToken = config.Get("api_token");

          InputFileName = config.Get("input_file_name") ?? "input.txt";
          OutputFileName = config.Get("output_file_name") ?? "output.txt";
```

```
            AnswerFileName = config.Get("answer_file_name") ?? "answer.txt";
            ReportFileName = config.Get("report_file_name") ?? "report.txt";
            CompilerLogFileName      =      config.Get("compiler_log_file_name")      ??
"compiler_log.txt";

            GetSubmissionDelayMs                                                    =
config.Get("get_submission_delay").ToInt32OrDefault(500);
            WorkerSlotCount = config.Get("worker_slot_count").ToUInt32OrDefault(1);

            Update.LatestVersionUrl    =    config.Get("update_latest_version_url")    ??
"http://localhost:8081/api/version";
            Update.LatestProgramUrl    =    config.Get("update_latest_program_url")    ??
"http://localhost:8081/api/update?version=latest";

            fid = Path.Combine(
              Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
              Process.GetCurrentProcess().ProcessName);

            if (!Directory.Exists(fid))
              Directory.CreateDirectory(fid);

            fid = Path.Combine(fid, "fid");

            if (File.Exists(fid))
              if (!Guid.TryParse(File.ReadAllText(fid), out _workerId))
              {
                File.Delete(fid);
                _workerId = Guid.Empty;
              }
          }

          public string WorkerName { get; }
          public string TestingWorkDirectory { get; }
          public string FileCacheFolder { get; }
          public string CompilersConfigFolder { get; }
          public uint ProblemsCacheSize { get; }
          public uint ResultSendingCacheSize { get; }
          public Uri BaseAddress { get; }
          public Uri BaseApiAddress { get; }
          public string ApiAuthToken { get; }
          public string InputFileName { get; }
          public string OutputFileName { get; }
          public string AnswerFileName { get; }
          public string ReportFileName { get; }
          public string CompilerLogFileName { get; }
          public int GetSubmissionDelayMs { get; }
          public uint WorkerSlotCount { get; }
          public UpdateConfiguration Update { get; }
          public Guid WorkerId
          {
            get => _workerId;
            set => File.WriteAllText(fid, (_workerId = value).ToString());
          }

          private string fid;
        }
```

```
}
```

Файл WorkerTaskManager.cs

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using NLog;
using TestLib.Worker.ClientApi;
using TestLib.Worker.ClientApi.Models;

namespace TestLib.Worker
{
  internal class WorkerTaskManager
  {
    private Logger logger;
    private Slot[] slots;
    private Task[] workerTasks;
    private Task aliveStatusSenderTask;
    private CancellationTokenSource sendingCancellationTokenSource;
    private CancellationTokenSource slotsCancellationTokenSource;
    private CancellationTokenSource aliveStatusSenderCancellationTokenSource;
    private bool started;

    public WorkerTaskManager()
    {
      logger = LogManager.GetCurrentClassLogger();

      started = false;
    }

    public bool Init(Configuration configuration)
    {
      slots = new Slot[configuration.WorkerSlotCount];
      workerTasks = new Task[configuration.WorkerSlotCount + 1];

      aliveStatusSenderCancellationTokenSource = new CancellationTokenSource();

      try
      {
        aliveStatusSenderTask = Task.Run(
          () => sendAliveStatus(),
aliveStatusSenderCancellationTokenSource.Token);

        return true;
      }
      catch (Exception ex)
      {
        logger.Error(ex, "Can't initalize WorkerTaskManager. Start is not
avaliable");

        return false;
      }
    }
```

```
        public void PrintStatus()
        {
          Console.WriteLine("==========STATUS==========");

          foreach (var item in GetStatus())
            Console.WriteLine(item);
        }

        private string[] GetStatus()
        {
          string[] res = new string[Application.Get().Configuration.WorkerSlotCount +
1];

          res[0] = $"SendingTestingResult: {workerTasks[0]?.Status.ToString()}";
          for (uint i = 1; i <= Application.Get().Configuration.WorkerSlotCount; i++)
            res[i] = $"Slot {i}: {workerTasks[i]?.Status.ToString()}";

          return res;
        }

        public void Start()
        {
          if (started)
            return;

          StartSendingTestingResult();
          StartSlots();

          started = true;
        }
        public void Stop()
        {
          if (!started)
            return;

          slotsCancellationTokenSource.Cancel();
          sendingCancellationTokenSource.Cancel();

          StopSlots();
          StopSendingTestingResult(true);

          started = false;
        }
        public void End()
        {
          Stop();
          aliveStatusSenderCancellationTokenSource.Cancel();

          try
          { Task.WaitAll(aliveStatusSenderTask); }
          catch { }
        }

        #region Privates
        private void StartSendingTestingResult()
```

```
        {
          sendingCancellationTokenSource = new CancellationTokenSource();

          workerTasks[0] =
              Task.Run(()                    =>                    SendTestingResult(),
sendingCancellationTokenSource.Token);
        }

        private void StartSlots()
        {
          slotsCancellationTokenSource = new CancellationTokenSource();

          for (uint i = 1; i <= Application.Get().Configuration.WorkerSlotCount; i++)
            StartSlot(i);
        }

        private void StopSendingTestingResult(bool flushQuery)
        {
          sendingCancellationTokenSource.Cancel();

          try
          { Task.WaitAll(workerTasks[0]); }
          catch { }
        }

        private void StopSlots()
        {
          slotsCancellationTokenSource.Cancel();

          try
          { Task.WaitAll(new ArraySegment<Task>(workerTasks, 1, workerTasks.Length -
1).Array); }
          catch { }
        }

        private void SendTestingResult()
        {
          Application app = Application.Get();
          IApiClient client = new HttpCodelabsApiClient();
          var logger = LogManager.GetCurrentClassLogger();

          sendingCancellationTokenSource.Token.Register(()                          =>
app.RequestMessages.Enqueue(null));

          while (true)
          {
            var request = app.RequestMessages.Dequeue();
            if (request is null)
              break;

            try
            {
              client.SendRequest(request);
            }
            catch (Exception ex)
            {
```

```
                logger.Error("Error sending request {0} to server: {1}. Some data will
be lose", request.RequestUri, ex);
              }
            }

            sendingCancellationTokenSource.Token.ThrowIfCancellationRequested();
          }

          private void StartSlot(uint id)
          {
            slots[id - 1] = slots[id - 1] ?? new Slot(id);
            workerTasks[id] =
              Task.Run(()  =>  slots[id  -  1].Do(slotsCancellationTokenSource.Token),
slotsCancellationTokenSource.Token);
          }

          private void sendAliveStatus()
          {
            Application app = Application.Get();
            IApiClient client = new HttpCodelabsApiClient();

            while (!aliveStatusSenderCancellationTokenSource.IsCancellationRequested)
            {
              try
              {
                int stopped = 0;
                int failed = 0;

                foreach (var item in workerTasks)
                {
                  switch (item?.Status)
                  {
                    case TaskStatus.Running:
                      break;
                    case TaskStatus.WaitingToRun:
                    case TaskStatus.Canceled:
                    case TaskStatus.Created:
                      stopped++;
                      break;
                    case TaskStatus.Faulted:
                    default:
                      failed++;
                      break;
                  }
                }

                WorkerStatus status;
                if (failed > 0)
                  status = WorkerStatus.Failed;
                else if (stopped > 0)
                  status = WorkerStatus.Stopped;
                else
                  status = WorkerStatus.Ok;

                client.UpdateWorker(app.Configuration.WorkerId,                    new
WorkerInformation(status, GetStatus()));
```

```
              }
              catch (Exception ex)
              {
                logger.Error(ex, "Updating worker information failed with error {0}",
ex.GetType().Name);
              }
              finally
              {
                Thread.Sleep(60 * 1000);
              }
            }


aliveStatusSenderCancellationTokenSource.Token.ThrowIfCancellationRequested();
          }
          #endregion
        }
      }
```

Файл Tester.h

```
#pragma once
#include <Windows.h>
#include <cstdio>

#include "Utils.h"
#include "Logger.h"

#pragma comment(lib, "kernel32.lib")
#pragma comment(lib, "user32.lib")
#pragma comment(lib, "advapi32.lib")

using uint8 = unsigned char;
using uint16 = unsigned short int;
using uint32 = unsigned long int;

#define nameof(name) #name

namespace TestLib
{
  public enum class IOHandleType
  {
    Input,
    Output,
    Error,
  };

  public enum class TestingResult
  {
    Ok = 0,
    WrongAnswer = 1,
    PresentationError = 2,
    Dirt = 4,
    Points = 5,
    BadTest = 6,
    UnexpectedEof = 8,
    CompilerError = 9,
```

```
    RuntimeError = 10,
    TestingError = 13,
    MemoryLimitExceeded = 14,
    TimeLimitExceeded = 15,
    PartiallyCorrect = 16
  };

  public enum class WaitingResult
  {
    Ok,
    Timeout,
    Fail,
  };

  public value class UsedResources
  {
  public:
    double RealTimeUsageMS;
    double CPUWorkTimeMS;
    double PeakMemoryUsageKB;
    uint32 ProcessExitCode;
  };
}

namespace Internal
{
  HANDLE DuplicateCurrentProcessToken();
  bool SetTokenIntegrityLevel(HANDLE hToken, DWORD mandatoryLabelAuthority);
  bool SetProcessIntegrityLevel(HANDLE hProcess, DWORD mandatoryLabelAuthority);

  class Tester
  {
  public:
    static ULONG current_processor;
    static ULONG processor_count;

    static void Init()
    {
      current_processor = 0;

      const int buffer_size = 8;
      wchar_t env_var_num_of_proc[buffer_size];

      if (GetEnvironmentVariableW(L"NUMBER_OF_PROCESSORS",
        env_var_num_of_proc, sizeof(wchar_t) * buffer_size) == 0
        || GetLastError() == ERROR_ENVVAR_NOT_FOUND)
        processor_count = 1;
      else
        processor_count = _wtoi(env_var_num_of_proc);
    }

    Tester()
    {
      programSet = false;
      realTimeLimitSet = false;
      memoryLimitSet = false;
```

```
  workDirSet = false;

  userInfo.useLogon = false;
  userInfo.userName = nullptr;
  userInfo.domain = nullptr;
  userInfo.password = nullptr;

  IoHandles.input = INVALID_HANDLE_VALUE;
  IoHandles.output = INVALID_HANDLE_VALUE;
  IoHandles.error = INVALID_HANDLE_VALUE;

  startupHandles.process = INVALID_HANDLE_VALUE;
  startupHandles.thread = INVALID_HANDLE_VALUE;
  startupHandles.job = INVALID_HANDLE_VALUE;

  limits.realTimeLimitMs = 0;
  limits.memoryLimitKb = 0;

  usedResources.CPUWorkTimeMS = 0;
  usedResources.PeakMemoryUsageKB = 0;
  usedResources.ProcessExitCode = 0;
  usedResources.RealTimeUsageMS = 0;

  program = nullptr;
  args = nullptr;
  workDirectory = nullptr;
}

~Tester()
{
  programSet = false;
  realTimeLimitSet = false;
  memoryLimitSet = false;
  workDirSet = false;

  SafeCloseHandle(&IoHandles.input);
  SafeCloseHandle(&IoHandles.output);
  //SafeCloseHandle(&IoHandles.error);

  if (startupHandles.process != INVALID_HANDLE_VALUE)
  {
    TerminateProcessS(startupHandles.process, 0);
    //SafeCloseHandle(&startupHandles.process);
    //SafeCloseHandle(&startupHandles.thread);
  }

  TerminateJobObject(startupHandles.job, 0);
  //SafeCloseHandle(&startupHandles.job);

  limits.realTimeLimitMs = 0;
  limits.memoryLimitKb = 0;

  usedResources.CPUWorkTimeMS = 0;
  usedResources.PeakMemoryUsageKB = 0;
  usedResources.ProcessExitCode = 0;
  usedResources.RealTimeUsageMS = 0;
```

```
            //free(program);
            //free(args);
            //free(workDirectory);

            if (userInfo.password != nullptr)
              SecureZeroMemory(userInfo.password,           sizeof(wchar_t)           *
wcslen(userInfo.password));
          }

          void SetProgram(const wchar_t * _program, const wchar_t * _args)
          {
            programSet = true;

            program = _wcsdup(_program);
            args = _wcsdup(_args);
          }

          void  SetUser(const  wchar_t  *  _userName,  const  wchar_t  *  _domain,  const
wchar_t * _password)
          {
            userInfo.useLogon = true;

            userInfo.userName = _wcsdup(_userName);
            userInfo.domain = _wcsdup(_domain);
            userInfo.password = _wcsdup(_password);
          }

          void SetWorkDirectory(const wchar_t * _dir)
          {
            workDirSet = true;

            workDirectory = _wcsdup(_dir);
          }

          void SetRealTimeLimit(uint32 _timeMs)
          {
            realTimeLimitSet = true;

            limits.realTimeLimitMs = _timeMs;
          }

          void SetMemoryLimit(uint32 _memoryKb)
          {
            memoryLimitSet = true;

            limits.memoryLimitKb = _memoryKb;
          }

          bool  RedirectIOHandleToFile(TestLib::IOHandleType _handleType, const wchar_t
* _fileName)
          {
            if (_fileName == nullptr)
            {
              Internal::logger->Error(__FUNCTION__       L"ArgumentNullException       "
nameof(_fileName) "\n");
```

```
            throw "ArgumentNullException " nameof(_fileName);
          }

          SECURITY_ATTRIBUTES attr;
          attr.nLength = sizeof(SECURITY_ATTRIBUTES);
          attr.lpSecurityDescriptor = nullptr;
          attr.bInheritHandle = TRUE;

          DWORD rwMode;
          DWORD openMode;

          switch (_handleType)
          {
          case TestLib::IOHandleType::Input:
            rwMode = GENERIC_READ;
            openMode = OPEN_EXISTING;
            break;
          case TestLib::IOHandleType::Output:
          case TestLib::IOHandleType::Error:
            rwMode = GENERIC_WRITE;
            openMode = CREATE_ALWAYS;
            break;
            /*case RedirectFileHandleMode::Rewrite:
            rwMode = GENERIC_WRITE;
            openMode = CREATE_ALWAYS;
            break;
            */
          default:
            Internal::logger->Error(__FUNCTION__ L"IOHandleType incorrect _handleType
= %hhu\n",
              (uint8)_handleType);

            return false;
          }

          HANDLE   h   =   CreateFileW(_fileName,   rwMode,   0,   &attr,   openMode,
FILE_ATTRIBUTE_NORMAL, nullptr);
          if (h == INVALID_HANDLE_VALUE)
          {
            Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
CreateFileW failed error %s. File name = %s\n",
              __LINE__, GetErrorMessage().get(), _fileName);

            return false;
          }

          switch (_handleType)
          {
          case TestLib::IOHandleType::Input:
            SafeCloseHandle(&IoHandles.input);
            IoHandles.input = h;
            break;
          case TestLib::IOHandleType::Output:
            SafeCloseHandle(&IoHandles.output);
            IoHandles.output = h;
```

```
            break;
          case TestLib::IOHandleType::Error:
            SafeCloseHandle(&IoHandles.error);
            IoHandles.error = h;
            break;
          default:
            Internal::logger->Error(__FUNCTION__ L"IOHandleType incorrect _handleType
= %hhu\n",
              (uint8)_handleType);
            return false;
          }

          return true;
        }

        bool  RedirectIOHandleToHandle(TestLib::IOHandleType  _handleType,  void  *
_handle, bool _duplicate)
        {
          if (_handle == nullptr)
          {
            Internal::logger->Error(__FUNCTION__      L"ArgumentNullException      "
nameof(_handle) "\n");

            throw "ArgumentNullException " nameof(_handle);
          }

          PHANDLE h;

          switch (_handleType)
          {
          case TestLib::IOHandleType::Input:
            SafeCloseHandle(&IoHandles.input);
            h = &IoHandles.input;
            break;
          case TestLib::IOHandleType::Output:
            SafeCloseHandle(&IoHandles.output);
            h = &IoHandles.output;
            break;
          case TestLib::IOHandleType::Error:
            SafeCloseHandle(&IoHandles.error);
            h = &IoHandles.error;
            break;
          default:
            Internal::logger->Error(__FUNCTION__ L"IOHandleType incorrect _handleType
= %hhu",
              (uint8)_handleType);
            return false;

          }

          if (_duplicate)
          {
            if  (!DuplicateHandle(GetCurrentProcess(),  _handle,  GetCurrentProcess(),
h,
              0, TRUE, DUPLICATE_SAME_ACCESS))
            {
```

```
                Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
DuplicateHandle failed error %s",
                  __LINE__, GetErrorMessage().get());

                return false;
              }
            }
            else
            {
              (*h) = _handle;
            }

            return true;
          }

          void * GetIORedirectedHandle(TestLib::IOHandleType _handleType)
          {
            switch (_handleType)
            {
            case TestLib::IOHandleType::Input:
              return IoHandles.input;
            case TestLib::IOHandleType::Output:
              return IoHandles.output;
            case TestLib::IOHandleType::Error:
              return IoHandles.error;
            default:
              Internal::logger->Error(L"%hhd      is      incorrect      _handleType",
(uint8)_handleType);
              return nullptr;
            }
          }

          bool Run(bool useRestrictions = false);
          TestLib::WaitingResult Wait();
          void CloseIORedirectionHandles();

          uint32 GetRunResult()
          {
            return 0;
          }
          uint32 GetExitCode()
          {
            return usedResources.ProcessExitCode;
          }

          TestLib::UsedResources GetUsedResources()
          {
            return usedResources;
          }

        private:
          bool applyMemoryLimit()
          {
            if (limits.memoryLimitKb <= 0)
            {
```

```
            Internal::logger->Warning(L"Can't set not positive memory limit.
MemoryLimitKb = %lu\n",
              limits.memoryLimitKb);

            return false;
          }

          JOBOBJECT_EXTENDED_LIMIT_INFORMATION jobExtendedLimits = { 0 };

          if                          (!QueryInformationJobObject(startupHandles.job,
JobObjectExtendedLimitInformation,
            &jobExtendedLimits, sizeof(jobExtendedLimits), nullptr))
          {
            Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
QueryInformationJobObject failed error %s",
              __LINE__, GetErrorMessage().get());

            return false;
          }

          jobExtendedLimits.BasicLimitInformation.LimitFlags                      |=
JOB_OBJECT_LIMIT_PROCESS_MEMORY;
          jobExtendedLimits.ProcessMemoryLimit    =    static_cast<SIZE_T>(1.5    *
limits.memoryLimitKb * 1024); // 1.5X reserve;

          if                            (!SetInformationJobObject(startupHandles.job,
JobObjectExtendedLimitInformation,
            &jobExtendedLimits, sizeof(JOBOBJECT_EXTENDED_LIMIT_INFORMATION)))
          {
            Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
SetInformationJobObject failed error %s. Memory limit = %lu\n",
              __LINE__, GetErrorMessage().get(), limits.memoryLimitKb);

            return false;
          }

          return true;
        }
        bool applyMandatoryLevel(HANDLE hProcessCreationToken)
        {
          if                          (!SetTokenIntegrityLevel(hProcessCreationToken,
SECURITY_MANDATORY_LOW_RID))// SECURITY_MANDATORY_UNTRUSTED_RID))
          {
            Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
Can't set container process creation token security info. SetTokenIntegrityLevel failed
error %s",
              __LINE__, GetErrorMessage().get());

            SafeCloseHandle(&hProcessCreationToken);

            return false;
          }

          return true;
        }
        bool applyStartupAttribute(LPSTARTUPINFOEXW _startupInfoEx)
```

```
          {
            SIZE_T attributeListSize = 0;
            InitializeProcThreadAttributeList(nullptr, 2, 0, &attributeListSize);

            _startupInfoEx->lpAttributeList                                    =
(LPPROC_THREAD_ATTRIBUTE_LIST)malloc(attributeListSize);

            if (!InitializeProcThreadAttributeList(_startupInfoEx->lpAttributeList, 2,
0, &attributeListSize))
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
InitializeProcThreadAttributeList failed error %s",
                __LINE__, GetErrorMessage().get());

              return false;
            }

            DWORD64 mitigationPolicy =
              PROCESS_CREATION_MITIGATION_POLICY_DEP_ENABLE |
              PROCESS_CREATION_MITIGATION_POLICY_DEP_ATL_THUNK_ENABLE |
              PROCESS_CREATION_MITIGATION_POLICY_SEHOP_ENABLE |
              PROCESS_CREATION_MITIGATION_POLICY_HEAP_TERMINATE_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_BOTTOM_UP_ASLR_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_HIGH_ENTROPY_ASLR_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_STRICT_HANDLE_CHECKS_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_WIN32K_SYSTEM_CALL_DISABLE_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_EXTENSION_POINT_DISABLE_ALWAYS_ON;
      #ifdef _WIN32_WINNT_WIN10
            mitigationPolicy |=
              PROCESS_CREATION_MITIGATION_POLICY_FONT_DISABLE_ALWAYS_ON |
              PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON;
      #endif

            if     (!UpdateProcThreadAttribute(_startupInfoEx->lpAttributeList,     0,
PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY,
              &mitigationPolicy, sizeof(mitigationPolicy), nullptr, nullptr))
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
UpdateProcThreadAttribute for PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY failed error %s",
                __LINE__, GetErrorMessage().get());

              DeleteProcThreadAttributeList(_startupInfoEx->lpAttributeList);
              free(_startupInfoEx->lpAttributeList);

              return false;
            }

      #ifdef _WIN32_WINNT_WIN10
            DWORD childProcessPolicy = PROCESS_CREATION_CHILD_PROCESS_RESTRICTED;
            if     (!UpdateProcThreadAttribute(_startupInfoEx->lpAttributeList,     0,
PROC_THREAD_ATTRIBUTE_CHILD_PROCESS_POLICY,
              &childProcessPolicy, sizeof(childProcessPolicy), nullptr, nullptr))
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
UpdateProcThreadAttribute for PROC_THREAD_ATTRIBUTE_CHILD_PROCESS_POLICY failed error
%s",
```

```
                __LINE__, GetErrorMessage().get());

              DeleteProcThreadAttributeList(_startupInfoEx->lpAttributeList);
              free(_startupInfoEx->lpAttributeList);

              return false;
            }
      #endif

            return true;
          }
          bool applyUIRestrictions()
          {
            JOBOBJECT_BASIC_UI_RESTRICTIONS jobUILimits = { 0 };

            DWORD restrictionsClass =
              JOB_OBJECT_UILIMIT_EXITWINDOWS |
              JOB_OBJECT_UILIMIT_DESKTOP |
              JOB_OBJECT_UILIMIT_GLOBALATOMS |
              JOB_OBJECT_UILIMIT_DISPLAYSETTINGS |
              JOB_OBJECT_UILIMIT_SYSTEMPARAMETERS |
              JOB_OBJECT_UILIMIT_WRITECLIPBOARD |
              JOB_OBJECT_UILIMIT_READCLIPBOARD |
              JOB_OBJECT_UILIMIT_HANDLES |
              JOB_OBJECT_UILIMIT_ALL;

            jobUILimits.UIRestrictionsClass = restrictionsClass;
            if                              (!SetInformationJobObject(startupHandles.job,
JobObjectBasicUIRestrictions,
              &jobUILimits, sizeof(JOBOBJECT_BASIC_UI_RESTRICTIONS)))
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
Can't set UI restrinctions. SetInformationJobObject failed error %s",
                __LINE__, GetErrorMessage().get());

              return false;
            }

            return true;
          }
          bool applyAffinity()
          {
      #if _WIN32_WINNT > _WIN32_WINNT_VISTA
            if (!SetProcessAffinityUpdateMode(startupHandles.process, 0))
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
Can't disables dynamic update of the process affinity by the system.
SetProcessAffinityUpdateMode failed error %s",
                __LINE__, GetErrorMessage().get());

              return false;
            }
      #endif
            ULONG proc = InterlockedIncrement(&current_processor);
            if  (!SetProcessAffinityMask(startupHandles.process,  1ull  <<  (proc  %
processor_count)))
```

```
            {
              Internal::logger->Error(L"WinAPI error in " __FUNCTION__ " at line %d.
Can't update the process affinity. SetProcessAffinityMask failed error %s",
                __LINE__, GetErrorMessage().get());

              return false;
            }

            return true;
          }

          HANDLE LoginUser()
          {
            HANDLE hToken;

            BOOL res = LogonUserW(userInfo.userName, userInfo.domain, userInfo.domain,
              LOGON32_LOGON_INTERACTIVE, LOGON32_PROVIDER_DEFAULT, &hToken);

            if (res == 0)
            {
              Internal::logger->Error(__FUNCTIONW__ L" failed. Login with username %s
failed. Error: %s",
                userInfo.userName, GetErrorMessage().get());

              return INVALID_HANDLE_VALUE;
            }

            return hToken;
          }

          bool programSet;
          bool realTimeLimitSet;
          bool memoryLimitSet;
          bool workDirSet;
          wchar_t * program;
          wchar_t * args;
          wchar_t * workDirectory;

          DWORD startTime;
          TestLib::UsedResources usedResources;

          struct {
            wchar_t * userName;
            wchar_t * domain;
            wchar_t * password;

            bool useLogon;
          } userInfo;

          struct
          {
            HANDLE input;
            HANDLE output;
            HANDLE error;
          } IoHandles;
```

```
        struct
        {
          HANDLE process;
          HANDLE thread;
          HANDLE job;
        } startupHandles;

        struct
        {
          uint32 realTimeLimitMs;
          uint32 memoryLimitKb;
        } limits;
      };
    }
```

Файл Tester.cpp

```
    #include "Tester.h"
    #include <cmath>

    namespace Internal
    {
      ULONG Tester::current_processor = 0;
      ULONG Tester::processor_count = 1;

      bool Tester::Run(bool useRestrictions)
      {
        Internal::logger->Debug(L"Called " __FUNCTIONW__);

        if (!workDirSet || !programSet)
        {
          Internal::logger->Error(L"Can't start program w/o program name or
workdirectory. workDirectory = '%s', program = '%s'", workDirectory, program);

          return false;
        }
        if (!realTimeLimitSet)
        {
          Internal::logger->Warning(L"Real time limit was not set. workDirectory =
'%s', program = '%s'", workDirectory, program);
        }
        if (!memoryLimitSet)
        {
          Internal::logger->Warning(L"Memory limit was not set. workDirectory = '%s',
program = '%s'", workDirectory, program);
        }

        HANDLE hProcessCreationToken;

        if (userInfo.useLogon)
          hProcessCreationToken = LoginUser();
        else
          hProcessCreationToken = DuplicateCurrentProcessToken();

        if (hProcessCreationToken == INVALID_HANDLE_VALUE)
```

```
            {
              Internal::logger->Error(L"Can't start program while hProcessCreationToken
is invalid. workDirectory = '%s', program = '%s'", workDirectory, program);

              return false;
            }

            startupHandles.job = CreateJobObjectW(nullptr, nullptr);

            STARTUPINFOEXW startupInfoEx = { 0 };
            startupInfoEx.StartupInfo.cb = sizeof(startupInfoEx);

            if (IoHandles.input != INVALID_HANDLE_VALUE)
            {
              startupInfoEx.StartupInfo.dwFlags |= STARTF_USESTDHANDLES;
              startupInfoEx.StartupInfo.hStdInput = IoHandles.input;
            }
            if (IoHandles.output != INVALID_HANDLE_VALUE)
            {
              startupInfoEx.StartupInfo.dwFlags |= STARTF_USESTDHANDLES;
              startupInfoEx.StartupInfo.hStdOutput = IoHandles.output;
            }
            if (IoHandles.error != INVALID_HANDLE_VALUE)
            {
              startupInfoEx.StartupInfo.dwFlags |= STARTF_USESTDHANDLES;
              startupInfoEx.StartupInfo.hStdError = IoHandles.error;
            }

            if (useRestrictions)
            {
              applyMandatoryLevel(hProcessCreationToken);
              applyMemoryLimit();
              applyUIRestrictions();
              //applyStartupAttribute(&startupInfoEx);
            }

            PROCESS_INFORMATION processInfo = { 0 };
            BOOL result = CreateProcessAsUserW(hProcessCreationToken, program, args,
              nullptr, nullptr, TRUE, CREATE_SUSPENDED | EXTENDED_STARTUPINFO_PRESENT |
CREATE_NO_WINDOW | CREATE_BREAKAWAY_FROM_JOB,
              nullptr, workDirectory, (STARTUPINFOW*)&startupInfoEx, &processInfo);

            if (!result)
            {
              Internal::logger->Error(L"Can't create process in " __FUNCTIONW__ " at line
%d. CreateProcessAsUserW failed, error %s", __LINE__, GetErrorMessage().get());

              return false;
            }

            if (useRestrictions)
            {
              applyAffinity();
            }

            if (!AssignProcessToJobObject(startupHandles.job, processInfo.hProcess))
```

```
          {
            Internal::logger->Error(L"Can't assign process to job in " __FUNCTIONW__ "
at    line    %d.    AssignProcessToJobObject    failed,    error    %s",    __LINE__,
GetErrorMessage().get());

            if (!TerminateProcessS(startupHandles.process, -1))
              Internal::logger->Error(L"Failed  to  TerminateProcess.  workDirectory  =
'%s', program = '%s'", workDirectory, program);

            SafeCloseHandle(&processInfo.hThread);
            SafeCloseHandle(&processInfo.hProcess);

            DeleteProcThreadAttributeList(startupInfoEx.lpAttributeList);
            free(startupInfoEx.lpAttributeList);
            TerminateJobObject(startupHandles.job, -1);

            SafeCloseHandle(&startupHandles.job);
            SafeCloseHandle(&hProcessCreationToken);

            return false;
          }

          BOOL res = FALSE;
          if (!IsProcessInJob(processInfo.hProcess, startupHandles.job, &res) || !res)
          {
            Internal::logger->Error(L"Process didnot assign to job in " __FUNCTIONW__ "
at line %d. IsProcessInJob failed, error %s", __LINE__, GetErrorMessage().get());

            if (!TerminateProcessS(startupHandles.process, -1))
              Internal::logger->Error(L"Failed  to  TerminateProcess.  workDirectory  =
'%s', program = '%s'", workDirectory, program);

            SafeCloseHandle(&processInfo.hThread);
            SafeCloseHandle(&processInfo.hProcess);

            DeleteProcThreadAttributeList(startupInfoEx.lpAttributeList);
            free(startupInfoEx.lpAttributeList);
            TerminateJobObject(startupHandles.job, -1);

            SafeCloseHandle(&startupHandles.job);
            SafeCloseHandle(&hProcessCreationToken);
            return false;
          }

          startTime = GetTickCount();
          if (ResumeThread(processInfo.hThread) == (DWORD)-1)
          {
            Internal::logger->Error(L"Can't resume main thread in " __FUNCTIONW__ " at
line %d. ResumeThread failed, error %s", __LINE__, GetErrorMessage().get());

            if (!TerminateProcessS(startupHandles.process, -1))
              Internal::logger->Error(L"Failed  to  TerminateProcess.  workDirectory  =
'%s', program = '%s'", workDirectory, program);

            SafeCloseHandle(&processInfo.hThread);
            SafeCloseHandle(&processInfo.hProcess);
```

```
            DeleteProcThreadAttributeList(startupInfoEx.lpAttributeList);
            free(startupInfoEx.lpAttributeList);
            TerminateJobObject(startupHandles.job, -1);

            SafeCloseHandle(&startupHandles.job);
            SafeCloseHandle(&hProcessCreationToken);
            return false;
          }

          DeleteProcThreadAttributeList(startupInfoEx.lpAttributeList);
          free(startupInfoEx.lpAttributeList);
          SafeCloseHandle(&hProcessCreationToken);

          startupHandles.process = processInfo.hProcess;
          startupHandles.thread = processInfo.hThread;

          return true;
        }

        TestLib::WaitingResult Tester::Wait()
        {
          TestLib::WaitingResult result;

          DWORD timeOut = limits.realTimeLimitMs - (GetTickCount() - startTime);

          DWORD waitCode = WaitForSingleObject(startupHandles.process, timeOut);
          switch (waitCode)
          {
          case WAIT_TIMEOUT:
            result = TestLib::WaitingResult::Timeout;

            if (!TerminateProcessS(startupHandles.process, -1))
            {
              Internal::logger->Error(L"Failed  to  TerminateProcess.  workDirectory  =
'%s', program = '%s'", workDirectory, program);
              result = TestLib::WaitingResult::Fail;
            }

            SafeCloseHandle(&startupHandles.thread);
            SafeCloseHandle(&startupHandles.process);

            TerminateJobObject(startupHandles.job, -1);
            SafeCloseHandle(&startupHandles.job);

            usedResources.RealTimeUsageMS = limits.realTimeLimitMs + 1;
            usedResources.ProcessExitCode = WAIT_TIMEOUT;

            Internal::logger->Error(L"Waiting program timeout expired. workDirectory =
'%s', program = '%s'", workDirectory, program);
            break;
          case WAIT_FAILED:
            result = TestLib::WaitingResult::Fail;

            if (!TerminateProcessS(startupHandles.process, -1))
```

```
            Internal::logger->Error(L"Failed to TerminateProcess. workDirectory =
'%s', program = '%s'", workDirectory, program);

            SafeCloseHandle(&startupHandles.thread);
            SafeCloseHandle(&startupHandles.process);

            TerminateJobObject(startupHandles.job, -1);
            SafeCloseHandle(&startupHandles.job);

            usedResources.ProcessExitCode = -1;

            Internal::logger->Error(L"Waiting program failed. workDirectory = '%s',
program = '%s'", workDirectory, program);
            break;

          case WAIT_OBJECT_0:
            GetExitCodeProcess(startupHandles.process, &usedResources.ProcessExitCode);

            result = TestLib::WaitingResult::Ok;

            Internal::logger->Info(L"Program waited successfully. workDirectory = '%s',
program = '%s'", workDirectory, program);
            break;

          default:
            Internal::logger->Error(L"Error waiting process. Unknown status. status =
%u, workDirectory = '%s', program = '%s'", waitCode, workDirectory, program);
            break;
          }

          usedResources.RealTimeUsageMS    =    static_cast<uint32>(GetTickCount()    -
startTime);

          if (startupHandles.job != INVALID_HANDLE_VALUE)
          {
            /*JOBOBJECT_BASIC_ACCOUNTING_INFORMATION basicAccountingInfo;
            if                          (!QueryInformationJobObject(startupHandles.job,
JobObjectBasicAccountingInformation,
              &basicAccountingInfo,     sizeof(JOBOBJECT_BASIC_ACCOUNTING_INFORMATION),
nullptr))
            {
              //log->LogErrorLastSystemError(L"Error                              getting
BASIC_ACCOUNTING_INFORMATION");
            }
            else
            {
              usedResources.cpuWorkTimeMs = static_cast<uint32>
                ((basicAccountingInfo.TotalKernelTime.QuadPart                          +
basicAccountingInfo.TotalUserTime.QuadPart) / 10000);
            }*/

            //TODO: check need if
            LARGE_INTEGER frequency;
            QueryPerformanceFrequency(&frequency);

            ULONG64 processTime;
```

```
            if (!QueryProcessCycleTime(startupHandles.process, &processTime))
            {
              Internal::logger->Error(L"Error get process cycle time in " __FUNCTIONW__
" at line %d. QueryProcessCycleTime failed, error %s. workDirectory = '%s', program =
'%s'",
                __LINE__, GetErrorMessage().get(), workDirectory, program);
            }
            else
            {
              usedResources.CPUWorkTimeMS     =     trunc(10.     *     processTime     /
frequency.QuadPart) / 10.;
            }

            JOBOBJECT_EXTENDED_LIMIT_INFORMATION exLimitInfo;
            if                          (!QueryInformationJobObject(startupHandles.job,
JobObjectExtendedLimitInformation,
              &exLimitInfo, sizeof(JOBOBJECT_EXTENDED_LIMIT_INFORMATION), nullptr))
            {
              //log->LogErrorLastSystemError(L"Error                            getting
EXTENDED_LIMIT_INFORMATION");
            }
            else
            {
              usedResources.PeakMemoryUsageKB          =          trunc(10.          *
exLimitInfo.PeakJobMemoryUsed / 1024.) / 10.;
            }
          }

          return result;
        }
        void Tester::CloseIORedirectionHandles()
        {
          if (IoHandles.input != INVALID_HANDLE_VALUE)
            FlushFileBuffers(IoHandles.input);
          if (IoHandles.output != INVALID_HANDLE_VALUE)
            FlushFileBuffers(IoHandles.output);
          if (IoHandles.error != INVALID_HANDLE_VALUE)
            FlushFileBuffers(IoHandles.error);

          SafeCloseHandle(&IoHandles.input);
          SafeCloseHandle(&IoHandles.output);
          SafeCloseHandle(&IoHandles.error);
        }
      }
```

Файл WrapperTester.cpp

```
#include "Tester.h"
#include "Logger.h"
#include <vcclr.h>
#include <msclr/gcroot.h>
#include <assert.h>

using namespace System;
using namespace System::Runtime::InteropServices;
```

```
using namespace System::Runtime::CompilerServices;

namespace TestLib
{
  class NativeLogCallbackHandler
  {
  public:
    NativeLogCallbackHandler(ref class LoggerManaged^ owner);
  private:
    static void OnLog(Internal::Logger & log, const uint8 level, const wchar_t *
message, void * userData);
    msclr::gcroot<LoggerManaged^> owner;
  };

  public ref class LoggerManaged
  {
  public:
    enum class LogLevel : uint8
    {
      Trace,
      Debug,
      Info,
      Warn,
      Error,
      Fatal,
      Off,
    };

    delegate void LogEventHandler(ref class LoggerManaged^ log, LogLevel level,
String^ message);

    LoggerManaged() :
      nativeLogger(new Internal::Logger()),
      nativeHandler(new NativeLogCallbackHandler(this))
    { }

    ~LoggerManaged()
    {
      this->!LoggerManaged();
    }

    !LoggerManaged()
    {
      delete nativeLogger;
      nativeLogger = nullptr;

      delete nativeHandler;
      nativeHandler = nullptr;
    }

    void Destroy()
    {
      delete nativeLogger;
      nativeLogger = nullptr;

      delete nativeHandler;
```

```
            nativeHandler = nullptr;
          }

          void InitNativeLogger(LogEventHandler ^ logger)
          {
            this->Log += logger;

            Internal::logger = this->GetNative();
            Internal::logger->Debug(L"Check native log from " __FUNCTIONW__);
          }

          event LogEventHandler^ Log
          {
            [MethodImplAttribute(MethodImplOptions::Synchronized)]
            void add(LogEventHandler^ value) {
              managedHandler   =   safe_cast<LogEventHandler^>(Delegate::Combine(value,
managedHandler));
            }

            [MethodImplAttribute(MethodImplOptions::Synchronized)]
            void remove(LogEventHandler^ value) {
              managedHandler    =    safe_cast<LogEventHandler^>(Delegate::Remove(value,
managedHandler));
            }

          private:
            void raise(LoggerManaged^ log, LogLevel level, String^ message)
            {
              if (managedHandler != nullptr)
                managedHandler(this, level, message);
            }
          }

        internal:
          void RaiseLogEvent(Byte level, String^ message)
          {
            Log(this, (LogLevel)level, message);
          }
          void RaiseLogEvent(LogLevel level, String^ message)
          {
            Log(this, level, message);
          }

          Internal::Logger * GetNative() { return nativeLogger; }

        private:
          Internal::Logger * nativeLogger;
          NativeLogCallbackHandler * nativeHandler;
          LogEventHandler^ managedHandler;
        };

        public ref class Tester
        {
        public:
          static Tester()
          {
```

```
            Internal::Tester::Init();
          }

          Tester()
          {
            tester = new Internal::Tester();
          }

          void SetProgram(String ^ program, String ^ args)
          {
            using namespace Runtime::InteropServices;

            const          wchar_t*          program_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(program)).ToPointer();
            const           wchar_t*           args_native           =          (const
wchar_t*)(Marshal::StringToHGlobalUni(args)).ToPointer();

            tester->SetProgram(program_native, args_native);

            Marshal::FreeHGlobal(IntPtr((void*)program_native));
            Marshal::FreeHGlobal(IntPtr((void*)args_native));
          }
          void SetUser(String ^ userName, String ^ domain, String ^ password)
          {
            using namespace Runtime::InteropServices;

            const          wchar_t*          userName_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(userName)).ToPointer();
            const           wchar_t*           domain_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(domain)).ToPointer();
            const          wchar_t*          password_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(password)).ToPointer();

            tester->SetUser(userName_native, domain_native, password_native);

            Marshal::FreeHGlobal(IntPtr((void*)userName_native));
            Marshal::FreeHGlobal(IntPtr((void*)domain_native));
            Marshal::FreeHGlobal(IntPtr((void*)password_native));
          }

          void SetWorkDirectory(String ^ directory)
          {
            using namespace Runtime::InteropServices;

            const          wchar_t*          directory_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(directory)).ToPointer();
            tester->SetWorkDirectory(directory_native);

            Marshal::FreeHGlobal(IntPtr((void*)directory_native));
          }

          void SetRealTimeLimit(UInt32 timeMS)
          {
            tester->SetRealTimeLimit(timeMS);
          }
          void SetMemoryLimit(UInt32 memoryKB)
```

```
          {
            tester->SetMemoryLimit(memoryKB);
          }
          void RedirectIOHandleToFile(IOHandleType handleType, String ^ fileName)
          {
            if (fileName == nullptr)
              throw gcnew ArgumentNullException(nameof(fileName));

            using namespace Runtime::InteropServices;

            const          wchar_t*          fileName_native          =          (const
wchar_t*)(Marshal::StringToHGlobalUni(fileName)).ToPointer();

            tester->RedirectIOHandleToFile(handleType, fileName_native);

            Marshal::FreeHGlobal(IntPtr((void*)fileName_native));
          }
          Boolean RedirectIOHandleToHandle(IOHandleType handleType, void * handle, bool
duplicate)
          {
            if (handle == nullptr)
              throw gcnew ArgumentNullException(nameof(handle));
            return tester->RedirectIOHandleToHandle(handleType, handle, duplicate);
          }
          Boolean RedirectIOHandleToHandle(IOHandleType handleType, void * handle)
          {
            return RedirectIOHandleToHandle(handleType, handle, false);
          }
          Boolean  RedirectIOHandleToHandle(IOHandleType  handleType,  IntPtr  ^  handle,
bool duplicate)
          {
            return      RedirectIOHandleToHandle(handleType,      handle->ToPointer(),
duplicate);
          }
          Boolean RedirectIOHandleToHandle(IOHandleType handleType, IntPtr ^ handle)
          {
            return RedirectIOHandleToHandle(handleType, handle, false);
          }

          IntPtr ^ GetIORedirectedHandle(IOHandleType handleType)
          {
            return gcnew IntPtr(tester->GetIORedirectedHandle(handleType));
          }

          Boolean Run(Boolean useRestrictions) {  return tester->Run(useRestrictions);
}
          Boolean Run()   {   return tester->Run();   }
          WaitingResult Wait()
          {
            return tester->Wait();
          }
          void CloseIORedirectionHandles()  {    tester->CloseIORedirectionHandles(); }
          UInt32 GetRunResult()  {   return tester->GetRunResult();  }
          UInt32 GetExitCode()  {   return tester->GetExitCode();  }
          UsedResources GetUsedResources()   {   return tester->GetUsedResources();  }
        private:
```

```
          Internal::Tester * tester;

          static LoggerManaged ^ loggerManaged;
        };
        inline    NativeLogCallbackHandler::NativeLogCallbackHandler(LoggerManaged    ^
owner) : owner(owner)
        {
          owner->GetNative()->SetCallback(&OnLog, this);
        }
        inline void NativeLogCallbackHandler::OnLog(Internal::Logger & log, const uint8
level, const wchar_t * message, void * userData)
        {
          static_cast<NativeLogCallbackHandler*>(userData)->owner->RaiseLogEvent(level,
gcnew String(message));
        }
      }
```

ДОДАТОК Д

Ілюстративна частина

РОЗРОБКА МЕТОДУ І ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ ТРЕНУВАННЯ І ОЦІНЮВАННЯ РОБІТ ЗІ СПОРТИВНОГО ПРОГРАМУВАННЯ. ЧАСТИНА 1 СЕРВЕР ОЦІНЮВАННЯ РОБІТ

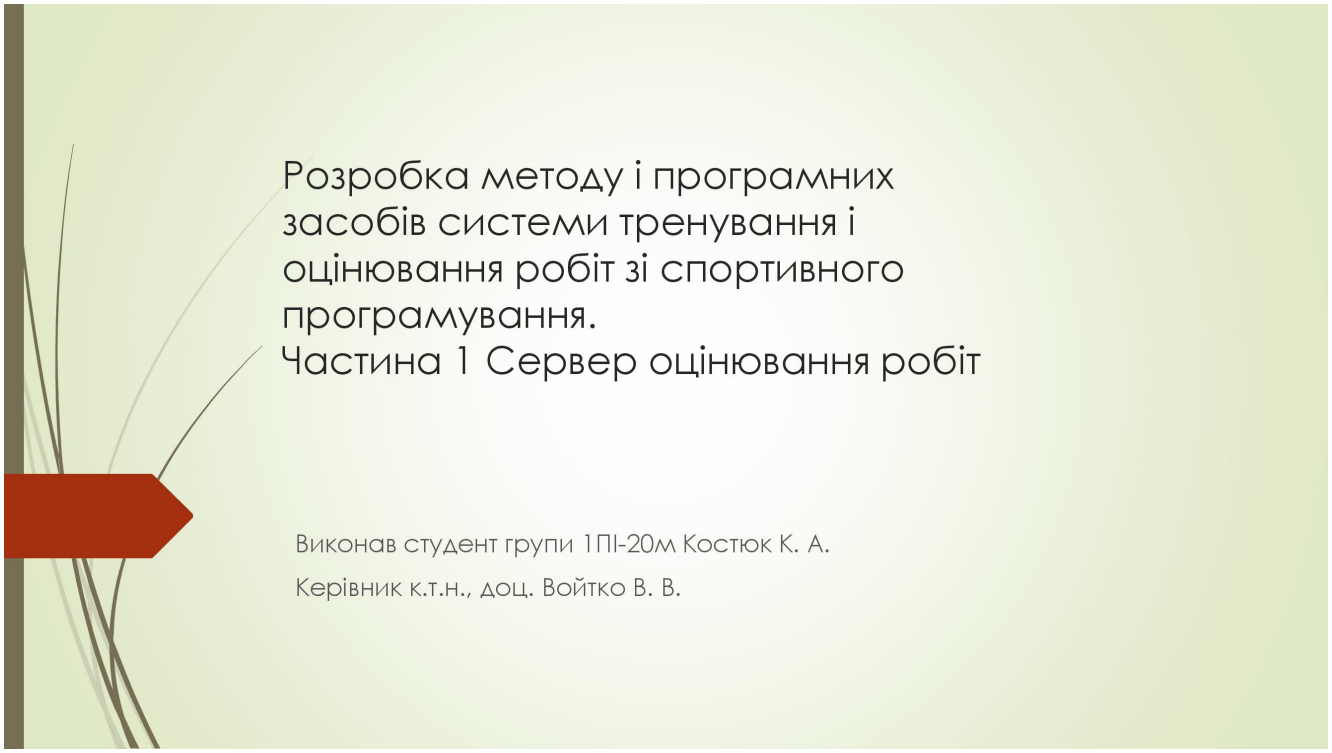


Рисунок Д.1 – Титульний слайд презентації

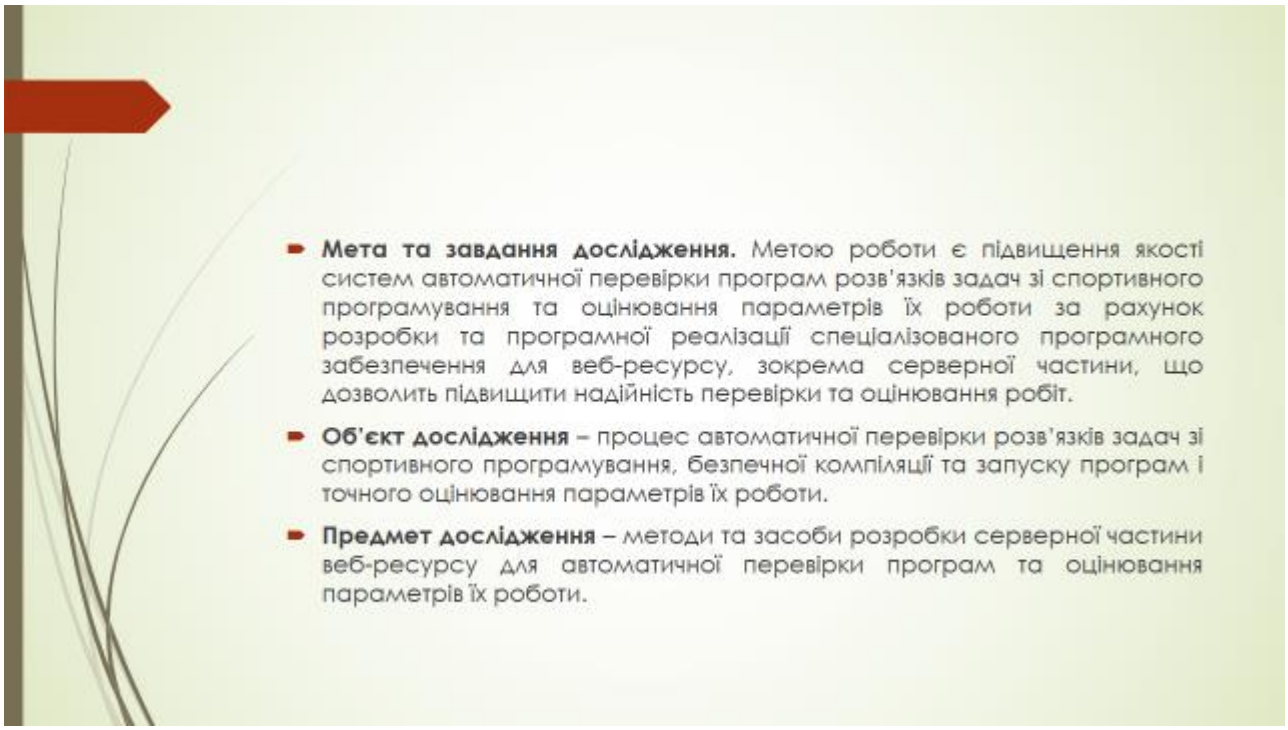


Рисунок Д.2 – Мета, об’єкт та предмет дослідження

Рисунок Д.3 – Задачі дослідження

Рисунок Д.4 – Наукова новизна отриманих результатів

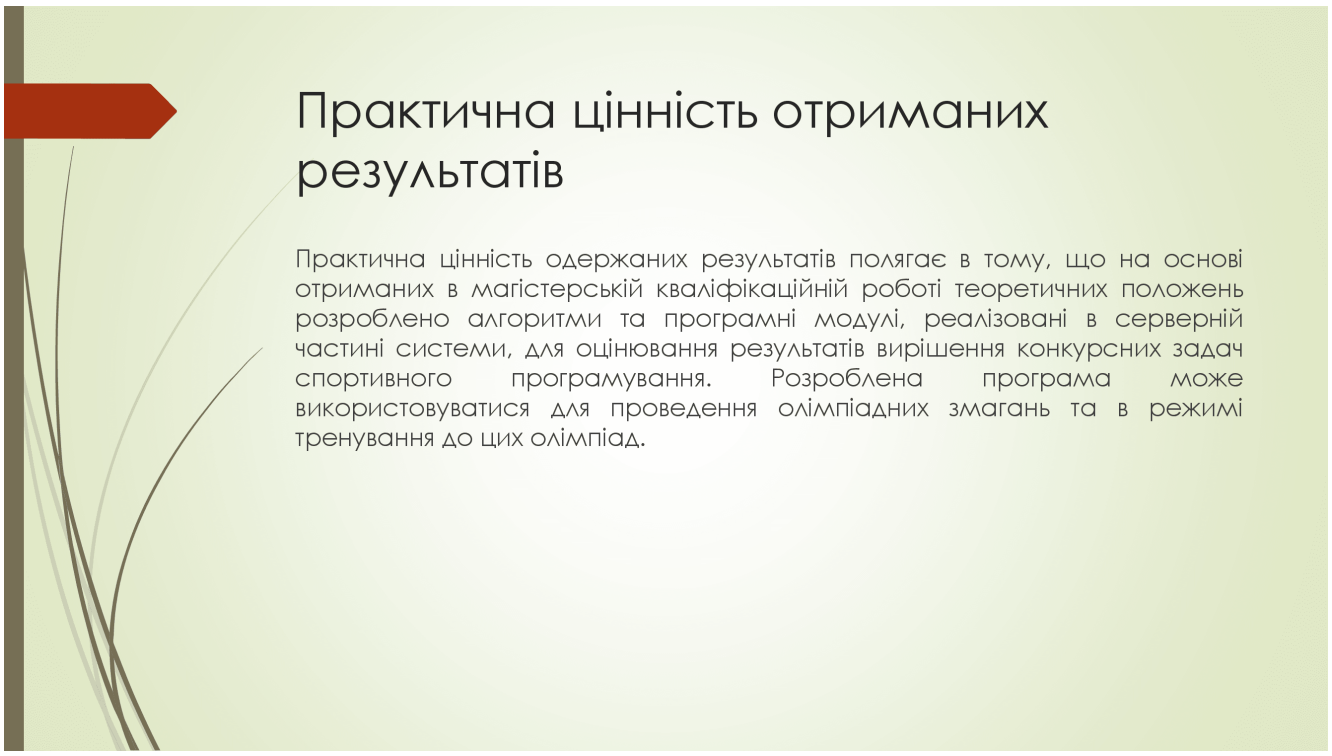


Рисунок Д.5 – Практична цінність отриманих результатів

Рисунок Д.6 – Порівняльний аналіз аналогів

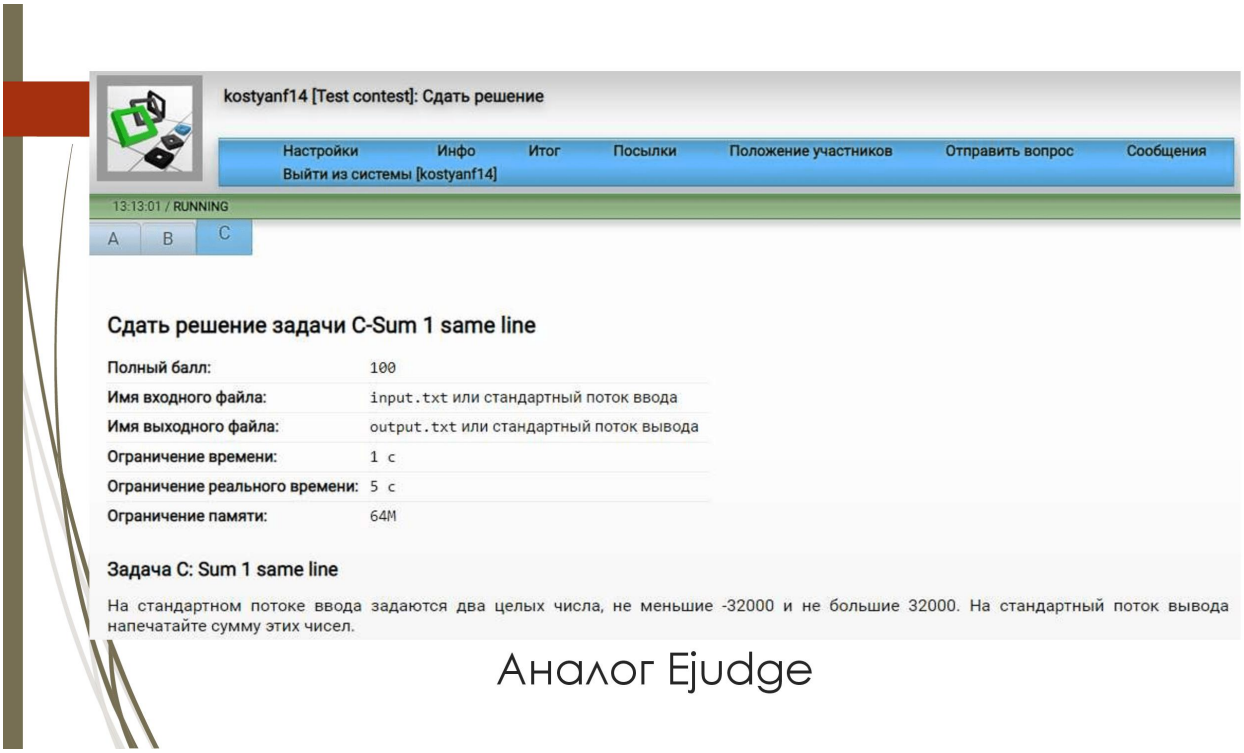


Рисунок Д.7 – Аналог Ejudge

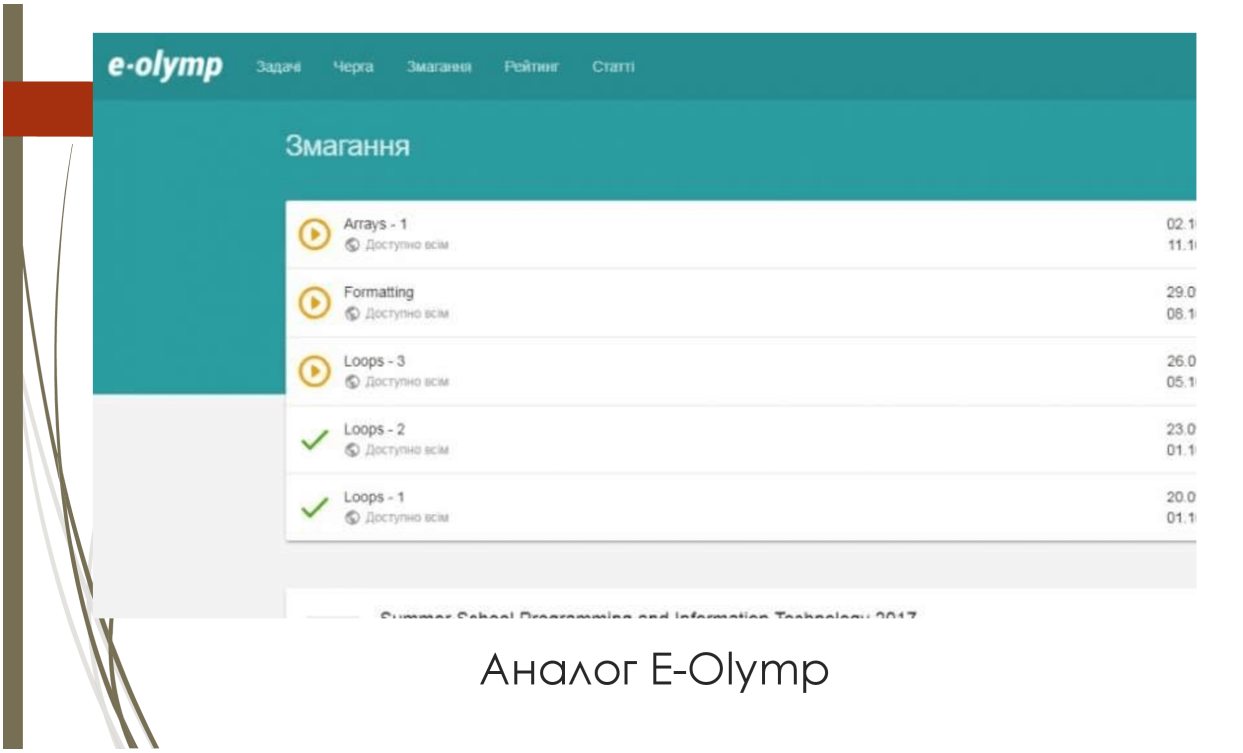


Рисунок Д.8 – Аналог E-Olymp

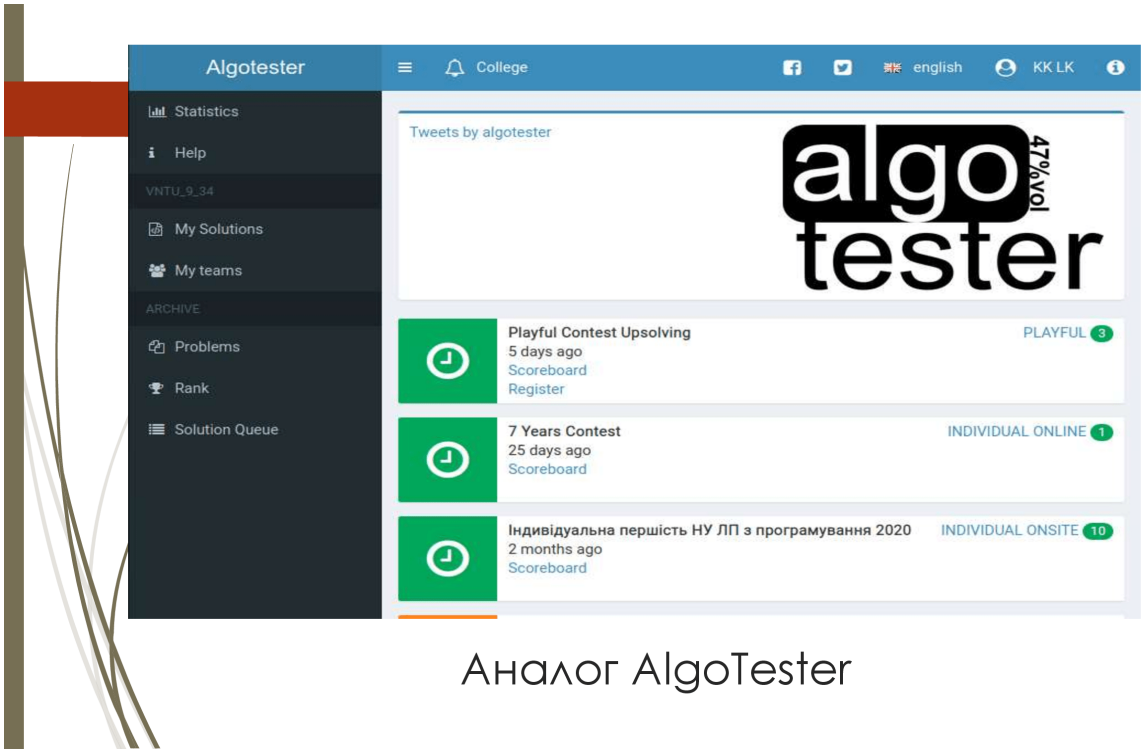


Рисунок Д.9 – Аналог AlgoTester

Порівняльний аналіз аналогів

Критерій	Ejudge	E-Olymp	Algotester	Codelabs
Можливість задання різних обмежень по часу та пам'яті для різних мов програмування	-	-	-	+
Детальний результат по кожному тесту	+	-	-	+
Точний розрахунок часу роботи програми	+/-	-	-	+
Багатопоточність компіляції/запуску рішень на перевірку	+	+	+	+
Можливість по-тестової компіляції	+	+	+	-
Можливість запуску на власних серверах	+	-	-	+
Безпечний запуск на Windows	-	-	-	+

Рисунок Д.10 – Порівняльний аналіз аналогів

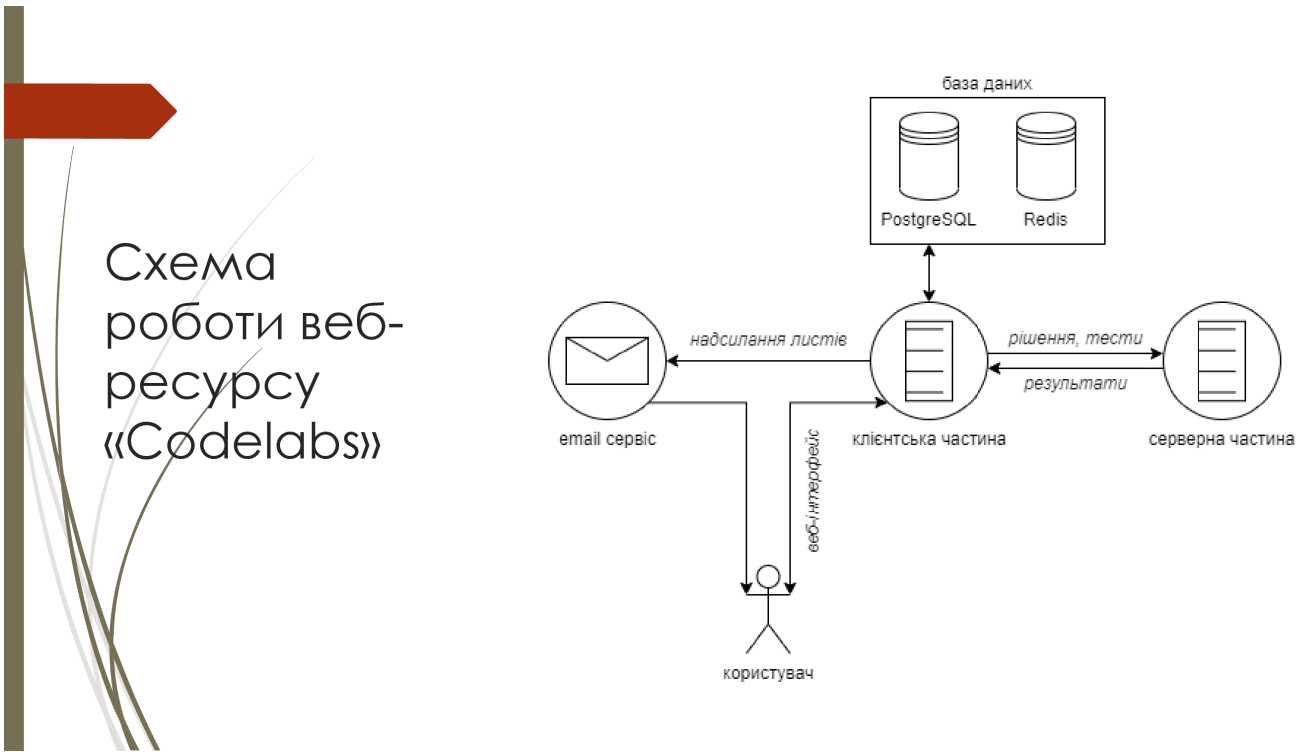


Рисунок Д.11 – Схема роботи веб-ресурсу «Codelabs»

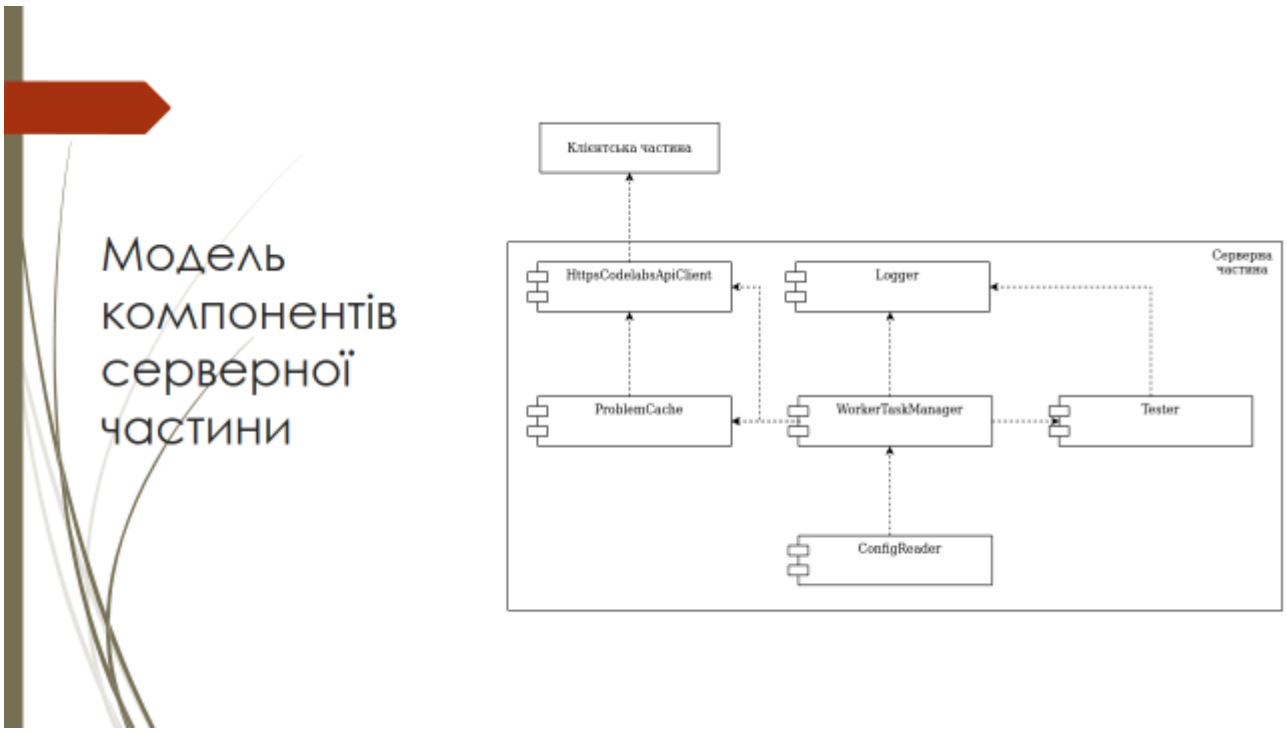


Рисунок Д.12 – Модель компонентів серверної частини

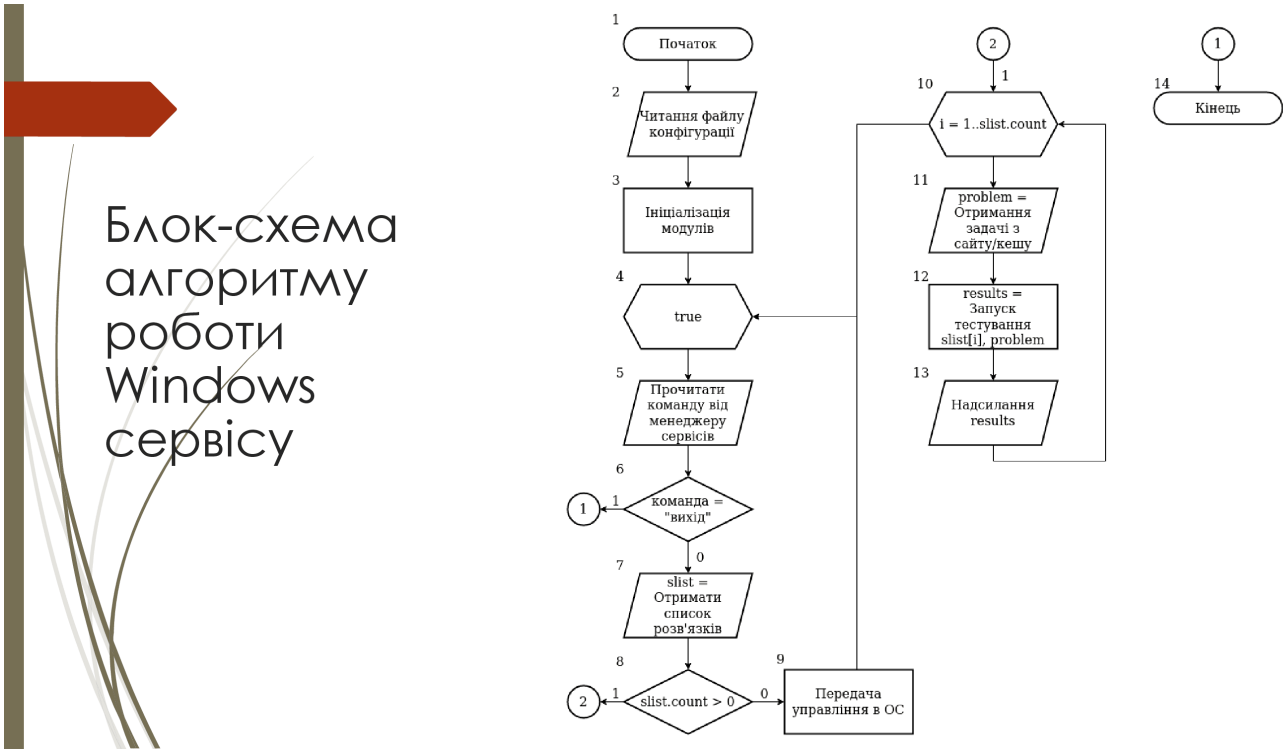
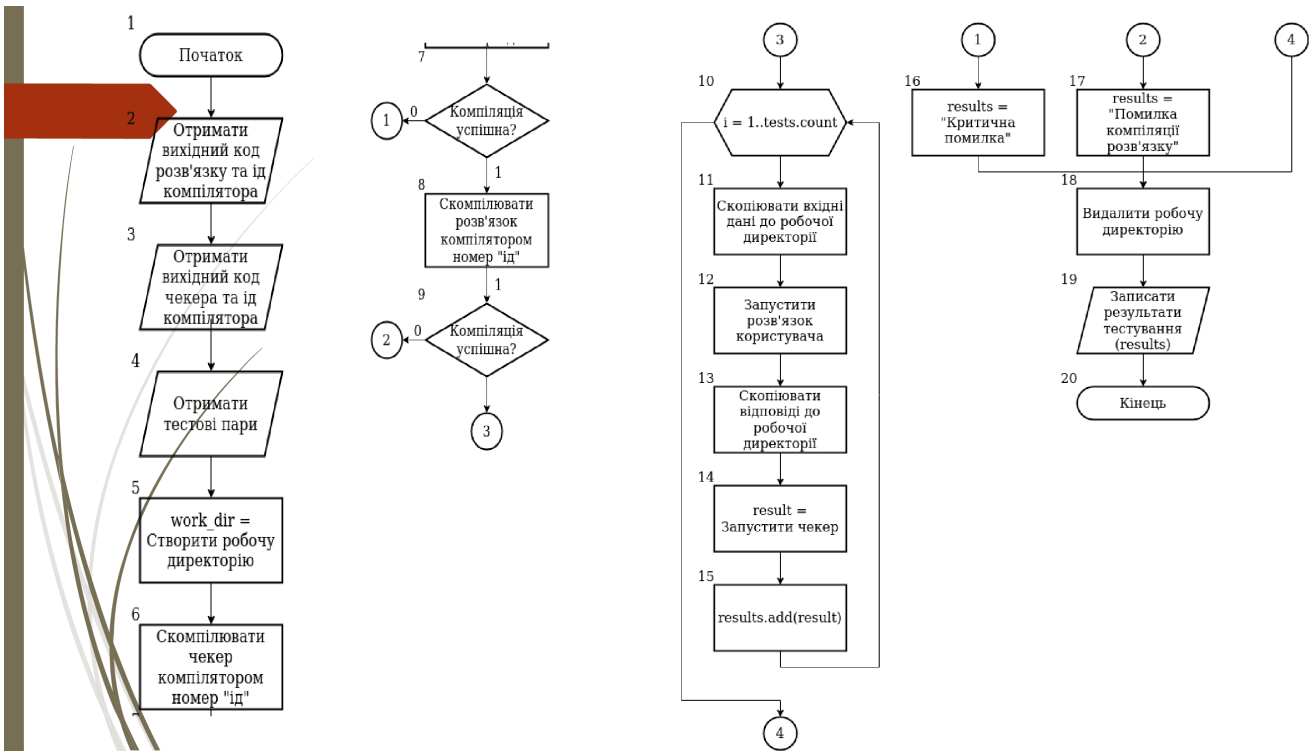


Рисунок Д.13 – Блок-схема алгоритму роботи Windows сервісу

Рисунок Д.14 – Блок-схема алгоритму роботи модуля «Tester»



РисунокД.15 – Блок-схема алгоритму роботи модуля «Tester»

Рисунок Д.16 – Розробка методу використання процесорних тактів для оцінки часу тестування програми

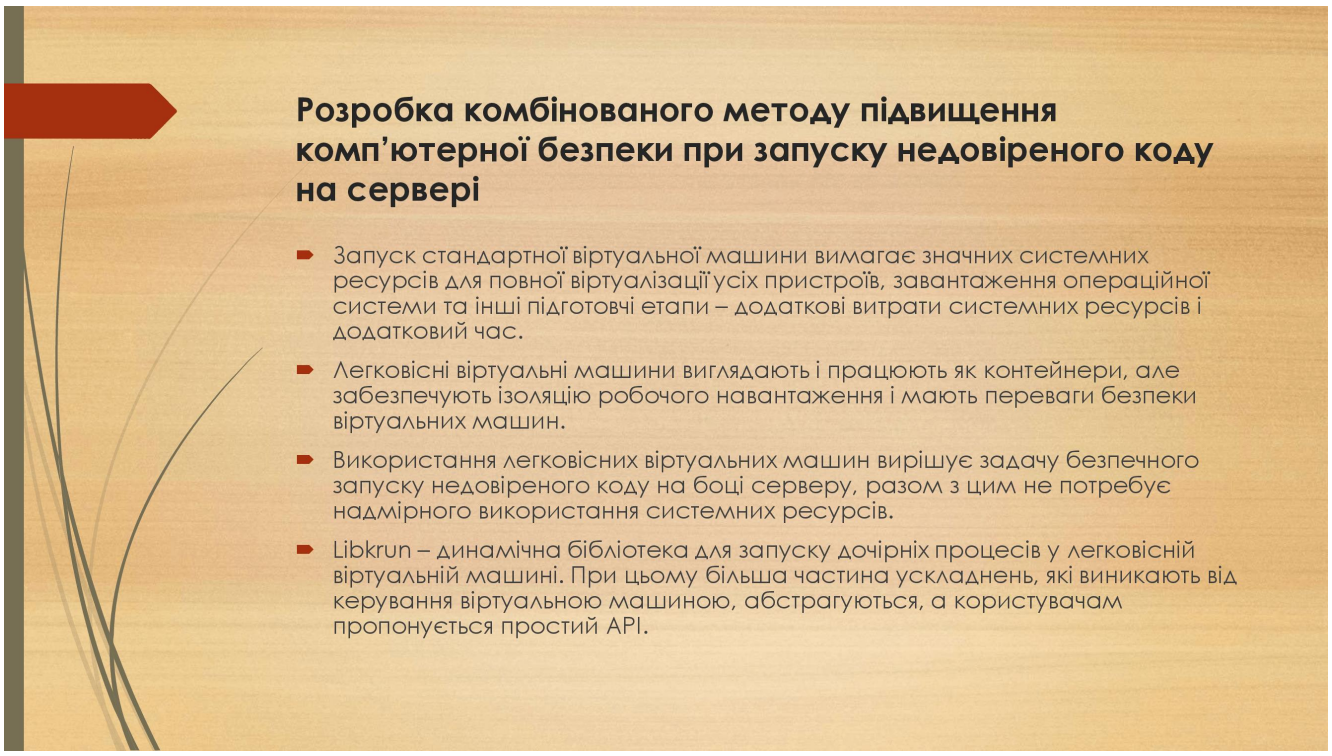


Рисунок Д.17 – Розробка комбінованого методу підвищення комп'ютерної безпеки при запуску недовіреного коду на сервері

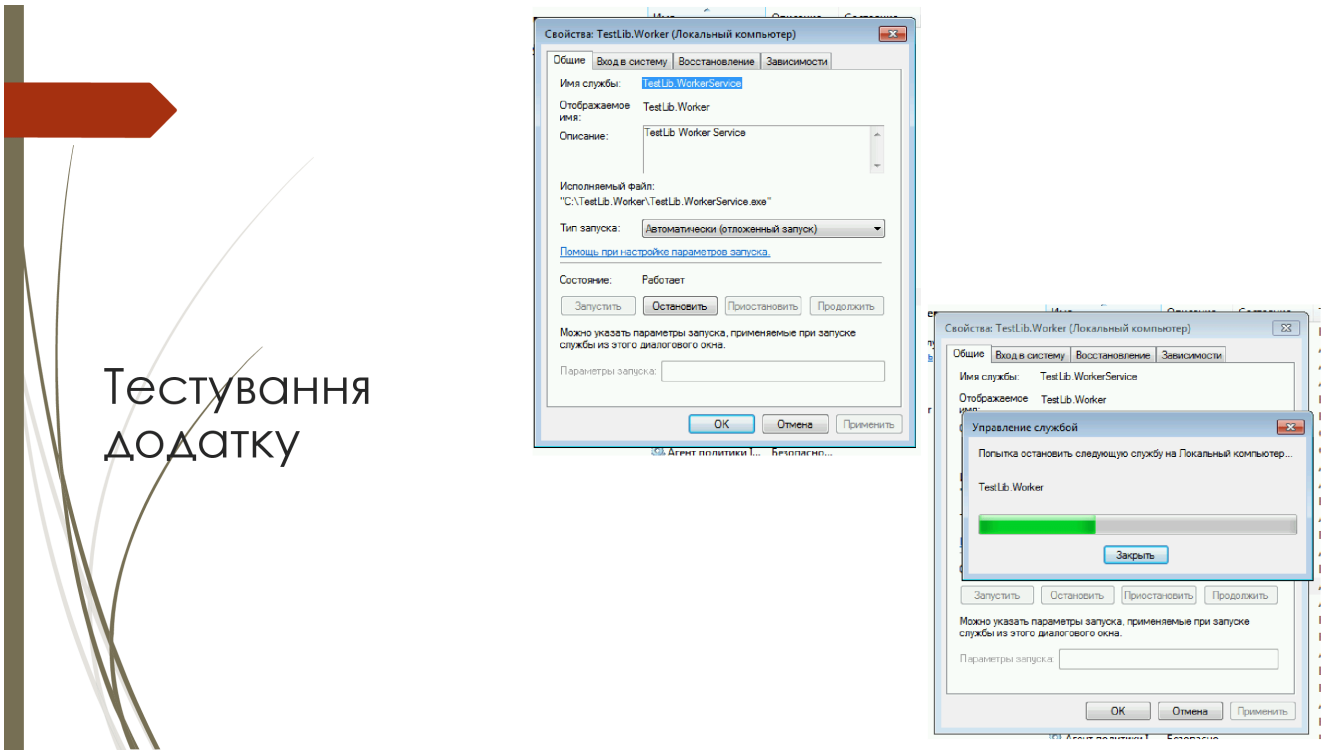


Рисунок Д.18 – Тестування додатку

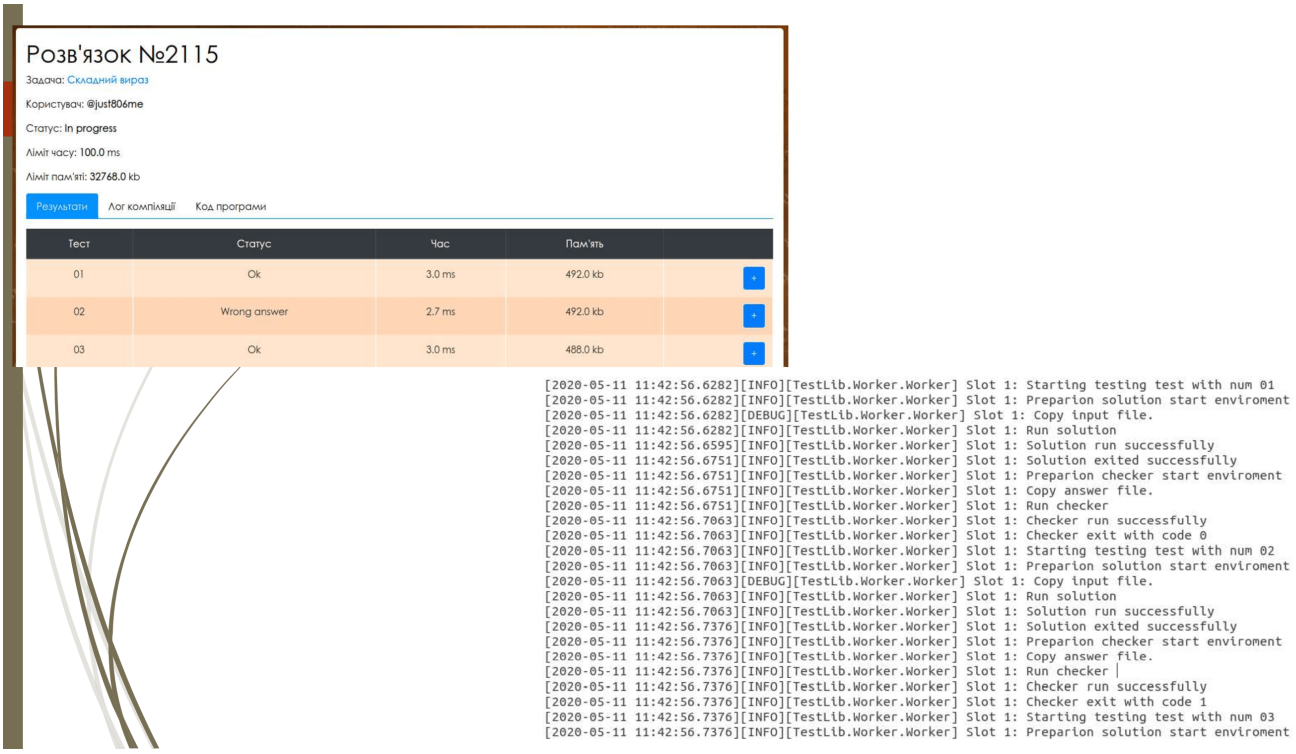


Рисунок Д.19 – Тестування додатку перевірки розв’язку задачі

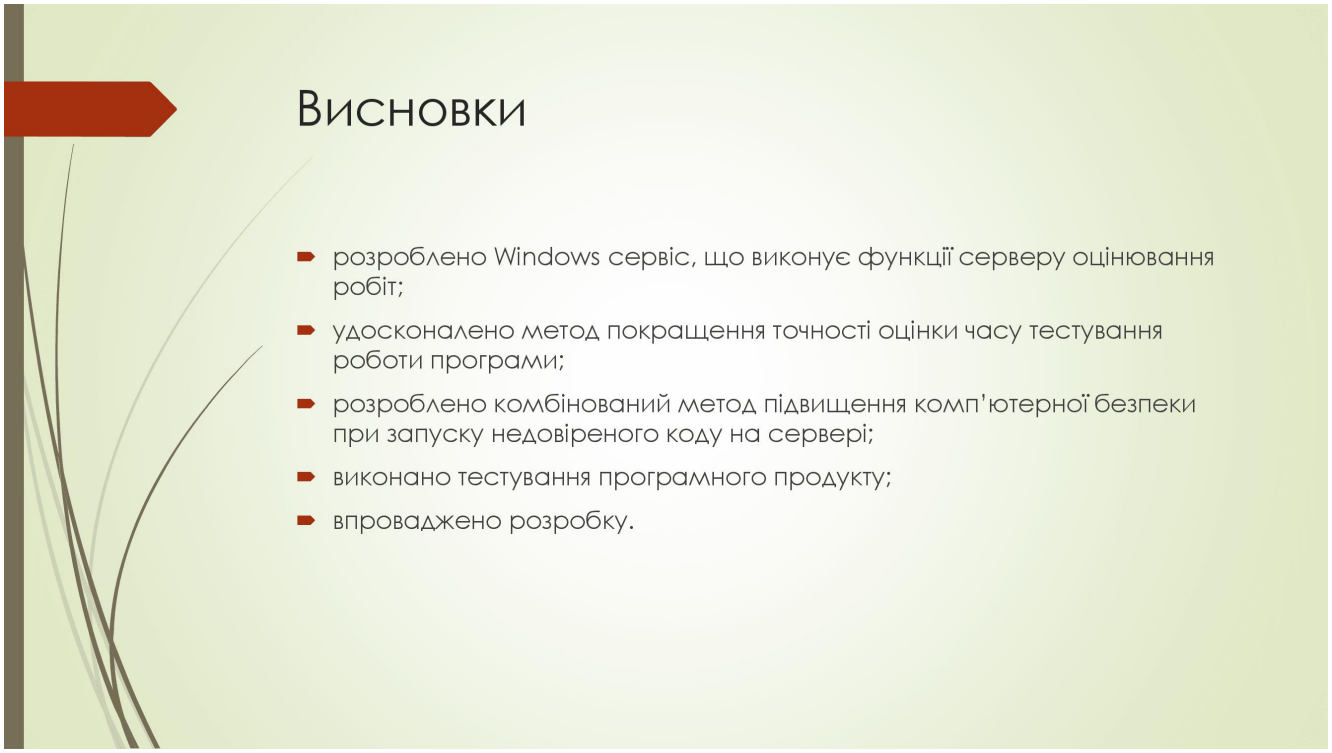


Рисунок Д.20 – Висновки

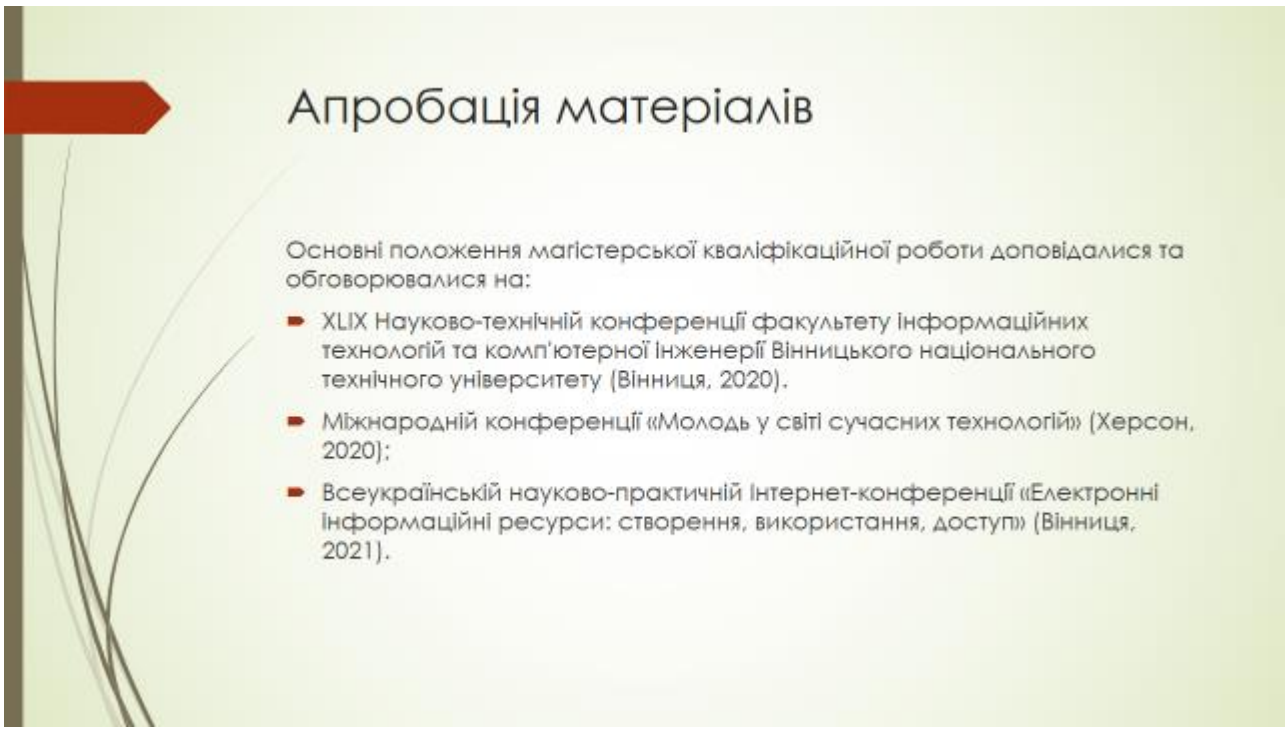


Рисунок Д.21 – Апробація матеріалів

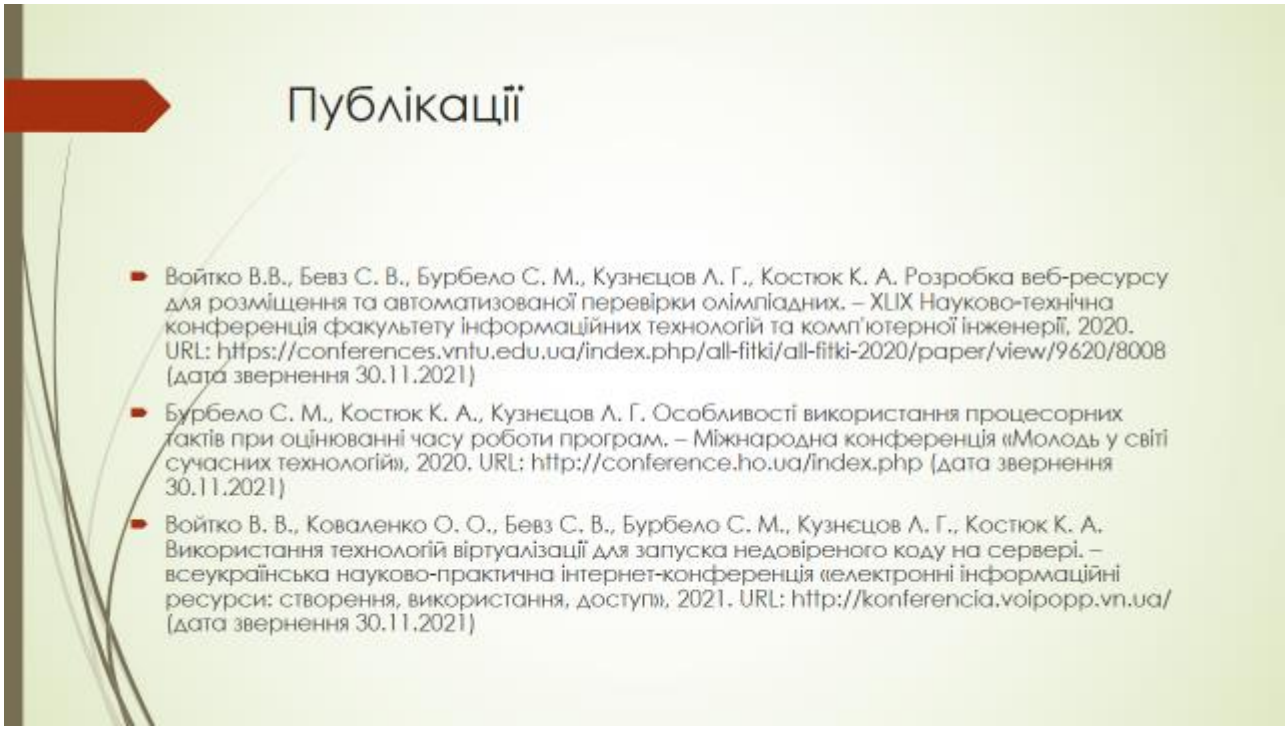


Рисунок Д.22 – Публікації

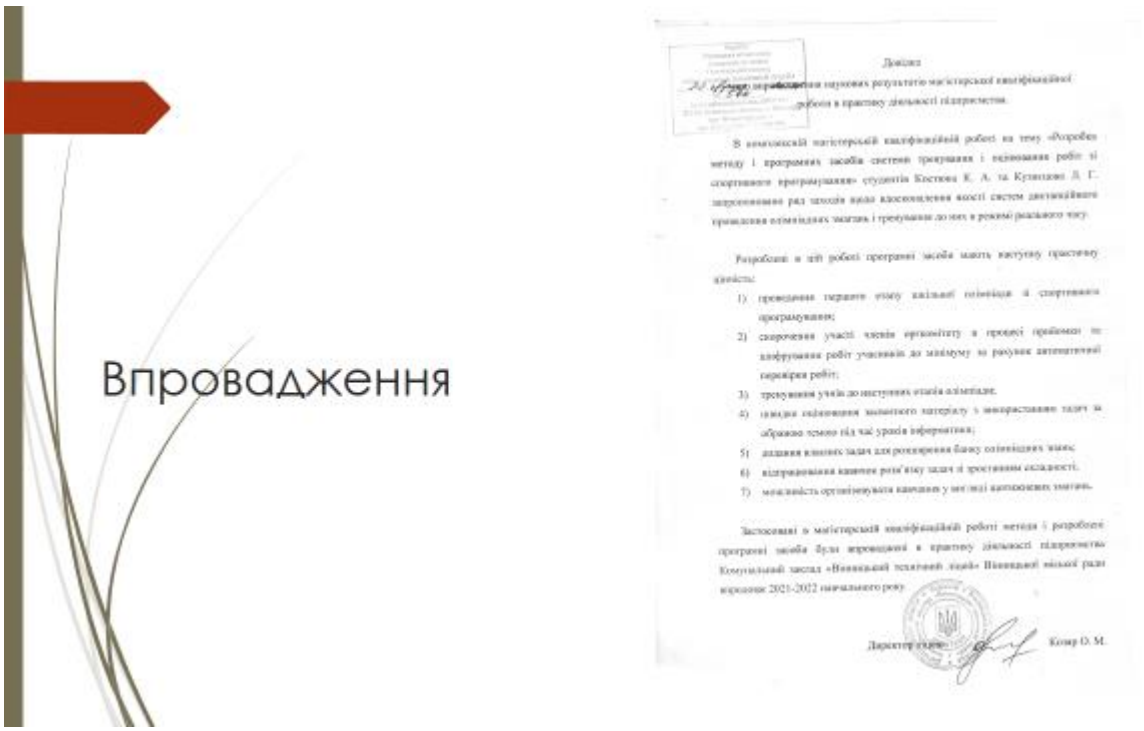


Рисунок Д.23 – Впровадження результатів магістерської кваліфікаційної роботи