

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Мікросервіс для перевірки та виконання
програмного коду в ізольованому середовищі»**

Виконав студент групи ВТ-22-2
ЗАКРЕВСЬКИЙ Сергій Олегович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ДУНЄВ Сергій Сергійович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ЗАКРЕВСЬКИЙ Сергій Олегович

Керівник роботи: МОРОЗОВ Андрій Васильович

Тема роботи: «Мікросервіс для перевірки та виконання програмного коду в ізольованому середовищі»,

затверджена Наказом закладу вищої освіти від « 27 » лютого 2026 р., № 92/с

Вихідні дані для роботи: розробити програмний комплекс для перевірки та виконання програмного коду в ізольованому середовищі.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	01.03.2026р.	01.03.2026р
2	Морозов А.В.	01.03.2026р	01.03.2026р
3	Морозов А.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра на тему:
«Мікросервіс для перевірки та виконання програмного коду в ізолюваному середовищі».

Робота містить 76 сторінок, 38 рисунків, 20 таблиць, 2 додатки, 40 використаних джерел.

Метою роботи є розробка мікросервісу для перевірки та виконання програмного коду користувачів в ізолюваному середовищі з використанням сучасних технологій контейнеризації та мікросервісної архітектури.

У процесі виконання роботи було проведено аналіз існуючих систем перевірки програмного коду, досліджено методи ізоляції виконання програм, обґрунтовано вибір архітектури програмного комплексу та використаних програмних засобів. Для реалізації проєкту використано технології Next.js, NestJS, PostgreSQL, Redis, Docker та систему черг Bull.

Розроблений програмний комплекс забезпечує приймання програмного коду від користувача, постановку задач на виконання до черги обробки, запуск коду в ізолюваних Docker-контейнерах, контроль часу виконання та отримання результатів роботи програм.

У результаті виконання кваліфікаційної роботи створено працездатний прототип мікросервісу для безпечного виконання програмного коду, який може бути використаний як основа для онлайн-компіляторів, навчальних платформ програмування та систем автоматизованої перевірки програмних рішень.

Ключові слова: МІКРОСЕРВІС, ВИКОНАННЯ КОДУ, ІЗОЛЮВАНЕ СЕРЕДОВИЩЕ, DOCKER, NESTJS.

					ІПЗ.КР.Б-121.26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Закревський С			Мікросервіс для перевірки та виконання програмного коду в ізолюваному	Літ.	Арк.	Аркушів
Перевір.							3	76
Керівник		Морозов А.В.				ФІКТ, Гр. ВТ-22-2		
Н. контр.								
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

Explanatory note to the bachelor's qualification work on the topic: “Microservice for Verification and Execution of Program Code in an Isolated Environment”.

The thesis consists of 76 pages, 38 figures, 20 tables, 2 appendices, and 40 references.

The purpose of this work is to develop a microservice for verification and execution of user program code in an isolated environment using modern containerization technologies and a microservice architecture.

During the research, existing code execution and verification systems were analyzed, methods of program isolation were studied, and the architecture of the software system along with the selected development technologies was justified. The implementation is based on Next.js, NestJS, PostgreSQL, Redis, Docker, and the Bull queue management system.

The developed software system provides acceptance of source code from users, task submission to a processing queue, execution of code inside isolated Docker containers, execution time control, and retrieval of program results. The architecture follows microservice principles, ensuring scalability, security, and extensibility.

As a result of the qualification work, a functional prototype of a secure code execution microservice was developed. The system can serve as a foundation for online compilers, programming education platforms, and automated code evaluation systems.

The practical value of the work lies in the development of a software solution that enables secure execution of user-provided code in a controlled environment and can be integrated into educational and commercial information systems.

Keywords: MICROSERVICE, CODE EXECUTION, ISOLATED ENVIRONMENT, DOCKER, NESTJS.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ СИСТЕМ ПЕРЕВІРКИ ТА ВИКОНАННЯ	
ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	13
1.3 Вибір архітектури мікросервісу для перевірки та виконання програмного коду	
.....	17
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного	
забезпечення	20
Висновки до першого розділу	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ МІКРОСЕРВІСУ ДЛЯ ПЕРЕВІРКИ ТА	
ВИКОНАННЯ ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ	
.....	24
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	
системи.....	24
2.2 Розробка бази даних системи	31
2.3 Проєктування та реалізація алгоритмів роботи системи	35
2.4 Реалізація програмного комплексу	42
2.5 Проєктування компонентної структури програмного комплексу	48
Висновки до другого розділу.....	54
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МІКРОСЕРВІСОМ ДЛЯ	
ВИКОНАННЯ ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ	
.....	55
3.1 Порядок встановлення та налаштування системи.....	55
3.2 Структура інтерфейсу вебзастосунку	59
3.3 Тестування роботи програмного комплексу	63
Висновки до третього розділу	68
ВИСНОВКИ.....	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	71
ДОДАТКИ	74

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – програмний інтерфейс взаємодії застосунків (Application Programming Interface)

CPU – центральний процесор (Central Processing Unit)

CRUD – базові операції роботи з даними (Create, Read, Update, Delete)

HTTP – протокол передачі гіпертекстових даних (HyperText Transfer Protocol)

JSON – текстовий формат обміну даними (JavaScript Object Notation)

REST – архітектурний стиль побудови вебсервісів (Representational State Transfer)

SaaS – модель надання програмного забезпечення як послуги (Software as a Service)

SQL – мова структурованих запитів до баз даних (Structured Query Language)

UI – інтерфейс користувача (User Interface)

UML – уніфікована мова моделювання програмних систем (Unified Modeling Language)

URL – уніфікований локатор ресурсу (Uniform Resource Locator)

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

У сучасному світі програмне забезпечення використовується практично в усіх сферах людської діяльності. Стрімкий розвиток інформаційних технологій сприяв появі великої кількості веборієнтованих платформ для навчання програмуванню, проведення технічних співбесід, автоматизованої перевірки програмних рішень та організації дистанційного навчання. Однією з ключових функцій таких систем є можливість безпечного виконання програмного коду, який надходить від користувачів через вебінтерфейс. [1; 2; 3]

Виконання стороннього програмного коду без належних механізмів ізоляції створює значні ризики для інформаційної безпеки та стабільності роботи інформаційних систем. Помилки в програмному коді, спроби несанкціонованого доступу до системних ресурсів, надмірне використання процесорного часу або оперативної пам'яті можуть призвести до порушення працездатності сервера та негативно вплинути на роботу інших користувачів. Саме тому особливої актуальності набуває використання сучасних засобів контейнеризації, які дозволяють виконувати користувацькі програми в ізольованому середовищі з контрольованим доступом до ресурсів операційної системи. [4; 5; 6]

Сучасні технології контейнеризації та мікросервісної архітектури дозволяють створювати програмні комплекси, які поєднують високий рівень безпеки, масштабованість та відмовостійкість. Використання таких підходів дає можливість розділити функціональність системи між незалежними компонентами, спростити супровід програмного забезпечення та забезпечити ефективну обробку великої кількості запитів. Обрані інструментальні засоби реалізації детально описані в офіційній документації відповідних технологій. [7; 8; 9; 10; 11; 12; 13; 14]

На сьогодні існує значна кількість програмних продуктів, які реалізують функції перевірки та виконання програмного коду, серед яких Judge0,

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

HackerRank, LeetCode, Sphere Engine та інші рішення. Вони забезпечують широкий спектр можливостей для роботи з програмним кодом, однак більшість таких систем мають певні обмеження. Частина з них є комерційними продуктами, частина не надає можливості повноцінного самостійного розгортання або інтеграції у сторонні інформаційні системи, а окремі рішення орієнтовані лише на навчальні чи тестові сценарії використання. У зв'язку з цим актуальним завданням є розробка власного програмного комплексу, який поєднуватиме безпечне виконання програмного коду, можливість подальшого розширення функціоналу та використання сучасних архітектурних підходів. [15]

Метою випускної кваліфікаційної роботи є розробка мікросервісу для перевірки та виконання програмного коду в ізолюваному середовищі із застосуванням сучасних засобів контейнеризації та мікросервісної архітектури.

Для досягнення поставленої мети в роботі передбачено виконання аналізу існуючих систем перевірки та виконання програмного коду, дослідження сучасних методів ізоляції програмних середовищ та забезпечення безпеки виконання користувацького коду, обґрунтування вибору архітектури програмного комплексу й технологій реалізації, проєктування структури мікросервісу та механізмів взаємодії його компонентів, розробку структури бази даних для збереження інформації про результати виконання програм, реалізацію серверної частини системи та сервісу обробки задач, створення механізму запуску програмного коду в ізолюваних Docker-контейнерах, а також проведення тестування розробленого програмного комплексу та оцінювання результатів його роботи. [16; 17; 18; 19; 20]

Об'єктом дослідження є процес автоматизованого виконання програмного коду в інформаційних системах.

Предметом дослідження є методи, алгоритми та програмні засоби реалізації мікросервісу для безпечного виконання програмного коду в ізолюваному середовищі.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Під час виконання роботи використано методи системного аналізу, об'єктно-орієнтованого проєктування, моделювання програмних систем із використанням UML-діаграм, сучасні технології веброзробки, контейнеризації та підходи до побудови мікросервісної архітектури. [21; 22; 23; 24; 25; 26]

Практичне значення одержаних результатів полягає у створенні працездатного програмного комплексу, який забезпечує безпечне виконання користувацького програмного коду в ізольованому середовищі. Розроблений мікросервіс може бути використаний як складова онлайн-компіляторів, освітніх платформ з програмування, систем автоматизованої перевірки завдань, сервісів технічного тестування та інших інформаційних систем, що потребують функціональності безпечного виконання програмного коду.

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі виконано аналіз предметної області, існуючих аналогів та технологій реалізації. У другому розділі наведено проєктування системи, опис архітектури, структури бази даних та алгоритмів роботи програмного комплексу. У третьому розділі розглянуто реалізацію мікросервісу, інтерфейс користувача, порядок встановлення, налаштування та результати тестування розробленої системи.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ СИСТЕМ ПЕРЕВІРКИ ТА ВИКОНАННЯ ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ

1.1 Постановка задачі

У даному підрозділі визначено постановку задачі для розробки мікросервісу перевірки та виконання програмного коду в ізольованому середовищі. У сучасних інформаційних системах все більшої популярності набувають сервіси автоматизованої перевірки програмного коду. Такі системи використовуються в онлайн-компіляторах, освітніх платформах, системах дистанційного навчання, сервісах технічного тестування кандидатів та змаганнях з програмування. Основною особливістю таких платформ є необхідність виконання програмного коду, який створюється безпосередньо користувачем.

Безпосередній запуск стороннього коду на сервері є потенційно небезпечним. Користувач може випадково або навмисно створити програму, яка споживає надмірну кількість процесорного часу, оперативної пам'яті або намагається отримати доступ до системних ресурсів. Внаслідок цього можуть виникати загрози інформаційній безпеці, зниження продуктивності сервера або порушення роботи інших сервісів. Для усунення зазначених проблем необхідно використовувати механізми ізоляції середовища виконання.

Одним із найбільш поширених способів вирішення даної задачі є використання контейнеризації. Контейнери дозволяють запускати програми в ізольованому середовищі з обмеженим доступом до файлової системи, мережевих ресурсів та апаратних компонентів сервера. Такий підхід забезпечує підвищення рівня безпеки та дозволяє контролювати процес виконання програмного коду.

Розроблюваний програмний комплекс повинен реалізовувати архітектуру, яка дозволить розділити процес прийому запитів, обробки задач та безпосереднього виконання коду між окремими компонентами системи. Для цього передбачається використання мікросервісного підходу, при якому

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

кожен компонент виконує власний набір функцій та взаємодіє з іншими компонентами через стандартизовані інтерфейси.

Мікросервіс перевірки та виконання програмного коду повинен складатися з клієнтської частини, яка забезпечує взаємодію користувача із системою, серверної частини, що приймає запити користувачів та керує процесом виконання програм, системи черг повідомлень для передачі задач між компонентами, окремого сервісу обробки задач, який виконує запуск контейнерів, ізолюваного середовища виконання програмного коду на базі Docker та бази даних для збереження інформації про користувачів і результати виконання програм. У процесі роботи системи користувач повинен мати можливість ввести програмний код через вебінтерфейс та відправити його на виконання. Після отримання запиту серверна частина повинна створити задачу та помістити її до черги обробки. Сервіс виконання задач отримує завдання з черги та запускає відповідний Docker-контейнер, у якому здійснюється виконання програми. Після завершення роботи результат повертається до серверної частини та стає доступним користувачу.

Основними функціональними можливостями системи є введення та передача програмного коду на сервер, запуск програм у контейнеризованому середовищі, обмеження часу виконання програм, отримання стандартного виводу програми, обробка синтаксичних та виконавчих помилок, збереження історії запусків програм і перегляд результатів виконання.

Крім функціональних вимог система повинна відповідати низці нефункціональних вимог. Зокрема, забезпечувати безпечне виконання програмного коду, підтримувати одночасну роботу декількох користувачів, забезпечувати масштабованість архітектури та можливість подальшого розширення функціоналу.

Важливою особливістю розроблюваного програмного комплексу є необхідність забезпечення стабільної роботи при одночасному виконанні декількох користувацьких програм. У випадку збільшення кількості запитів система повинна зберігати працездатність та забезпечувати коректну обробку

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

задач без суттєвого зниження продуктивності. Для досягнення цього доцільно використовувати механізми асинхронної обробки задач і розподілу навантаження між окремими компонентами мікросервісної архітектури.

Додатковою вимогою до програмного комплексу є забезпечення можливості подальшого розвитку та модернізації системи. Архітектура повинна дозволяти додавання нових мов програмування, зміну механізмів виконання коду та інтеграцію з іншими інформаційними системами без необхідності суттєвої зміни вже реалізованих компонентів. Такий підхід спрощує супровід програмного забезпечення та забезпечує його адаптацію до нових вимог користувачів.

Не менш важливим аспектом є контроль використання системних ресурсів під час виконання програмного коду. Система повинна мати можливість обмежувати час виконання програм, обсяг доступної оперативної пам'яті та інші ресурси середовища виконання. Це дозволяє запобігати виникненню ситуацій, коли окремі користувацькі програми негативно впливають на роботу всього програмного комплексу або створюють загрозу для стабільності сервера.

Крім того, програмний комплекс повинен забезпечувати зручний механізм взаємодії користувача із системою. Інтерфейс користувача має надавати можливість введення програмного коду, вибору мови програмування, запуску задач на виконання та перегляду результатів роботи програм. Отримані результати повинні містити інформацію про статус виконання, стандартний вивід програми, повідомлення про помилки та код завершення процесу.

Основними етапами розробки програмного комплексу є аналіз предметної області та існуючих аналогів, вибір архітектури програмного комплексу і технологій реалізації, проєктування структури бази даних, розробка серверної частини системи та сервісу виконання задач, реалізація контейнеризованого середовища виконання програм, створення клієнтської частини, а також проведення тестування та оцінка результатів роботи системи.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Результатом виконання кваліфікаційної роботи повинен стати програмний комплекс, який забезпечує безпечне виконання користувацького програмного коду в ізольованому середовищі, контроль використання системних ресурсів та надання результатів виконання через вебінтерфейс.

1.2 Аналіз аналогів програмного продукту

Перед розробкою програмного забезпечення доцільно виконати аналіз існуючих рішень, що реалізують функціональність перевірки та виконання програмного коду. Аналіз аналогів дозволяє визначити типові можливості систем даного класу, виявити їх переваги та недоліки, а також сформулювати вимоги до власного програмного продукту.

Одним із найбільш відомих відкритих рішень є система Judge0. Judge0 являє собою веборієнтовану платформу для виконання програмного коду в ізольованому середовищі. Система підтримує велику кількість мов програмування та надає REST API для інтеграції із зовнішніми сервісами. [15]

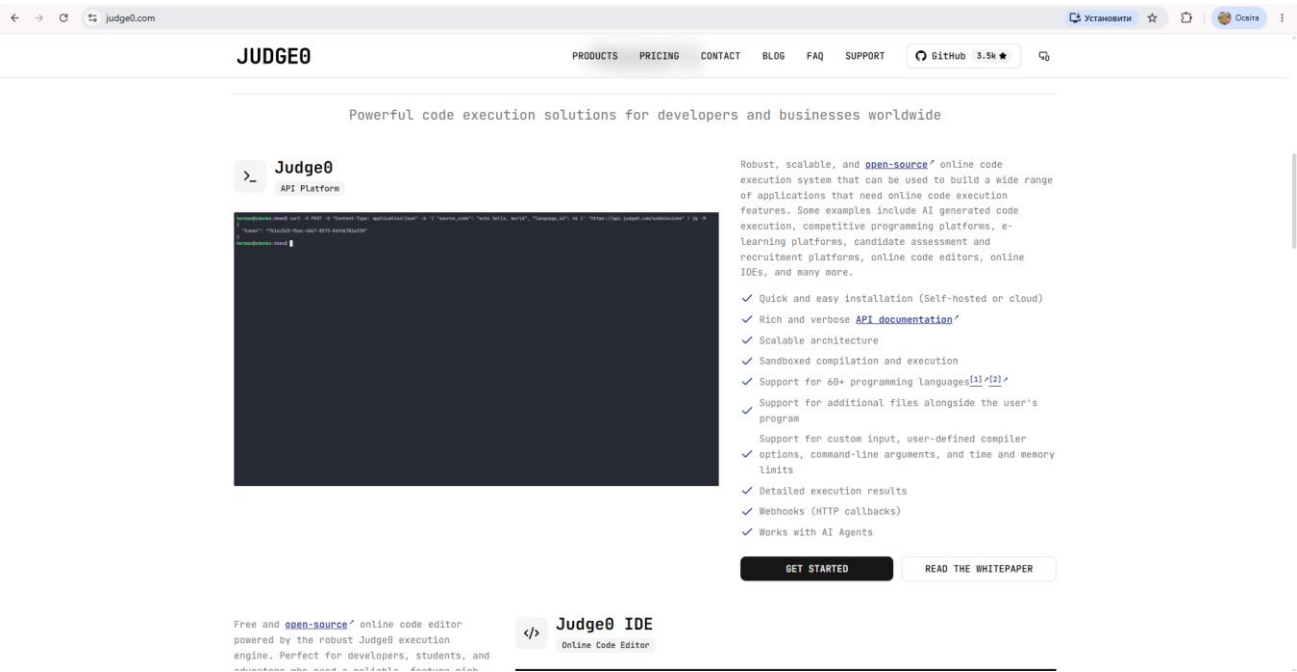


Рис. 1.1. Інтерфейс системи Judge0

Judge0 дозволяє виконувати програмний код у контейнеризованому середовищі, забезпечує контроль часу виконання та використання ресурсів. Важливою перевагою системи є відкритий вихідний код та можливість самостійного розгортання на власній інфраструктурі. Разом з тим базова версія

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

орієнтована переважно на API-взаємодію та не містить повноцінного користувацького середовища для управління запуском програм. [15]

Ще одним популярним рішенням є HackerRank. Дана платформа широко використовується для проведення технічних співбесід, навчання програмуванню та оцінювання професійних навичок розробників.

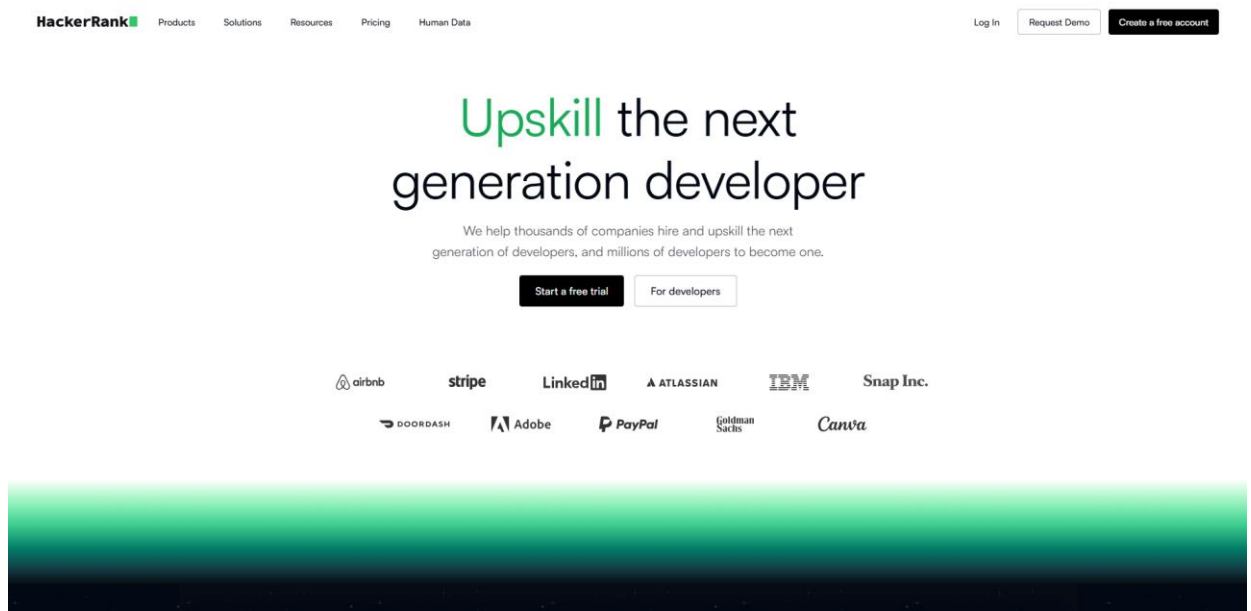


Рис. 1.2. Інтерфейс платформи HackerRank

Платформа надає користувачам великий набір задач різного рівня складності та підтримує автоматизовану перевірку рішень. Виконання програм здійснюється на серверній інфраструктурі платформи із використанням механізмів ізоляції. До недоліків можна віднести закритість внутрішньої архітектури та відсутність можливості використання платформи як окремого компонента власної інформаційної системи.

Серед найпопулярніших навчальних платформ також слід виділити LeetCode. Основним призначенням даного сервісу є підготовка розробників до технічних співбесід та вдосконалення навичок програмування. Платформа надає велику кількість практичних завдань різного рівня складності, що охоплюють алгоритми, структури даних, бази даних та інші напрямки папвпавпавпрограмноїінженерії.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

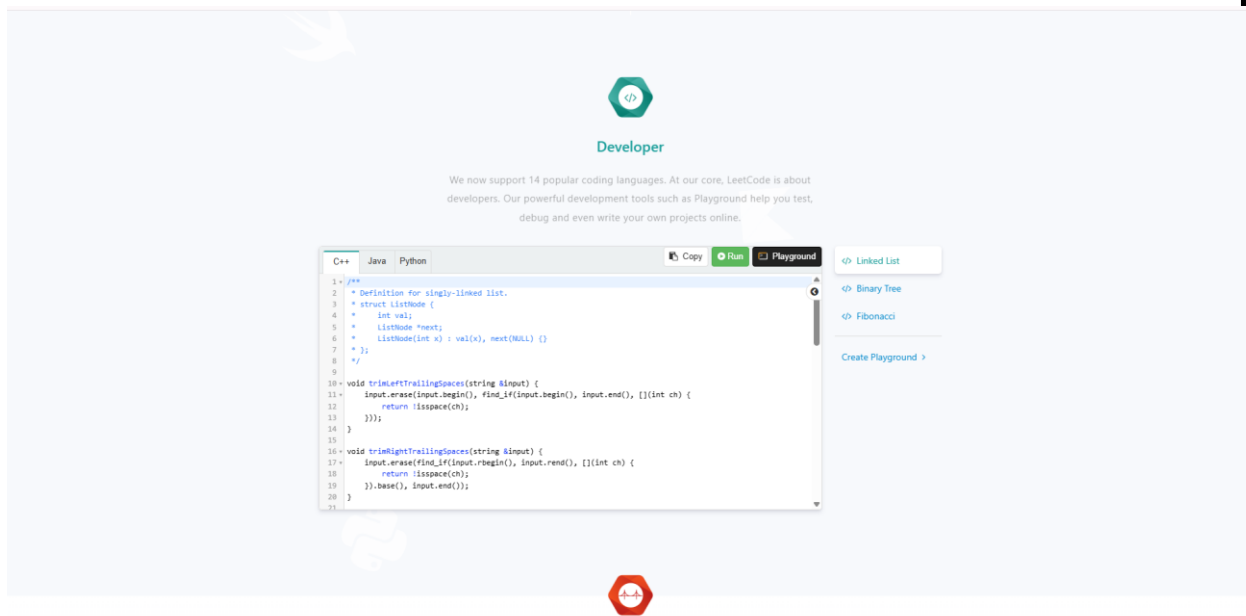


Рис. 1.3. Інтерфейс сервісу LeetCode

LeetCode забезпечує можливість написання та запуску програм безпосередньо у браузері. Платформа підтримує автоматичну перевірку результатів виконання та аналіз ефективності алгоритмів. Водночас система орієнтована насамперед на навчання та не надає відкритого API для повноцінного використання як окремого сервісу виконання програмного коду.

Ще одним прикладом спеціалізованої системи є Sphere Engine. Це комерційна платформа, яка надає API для інтеграції функціональності виконання програмного коду у сторонні програмні продукти.

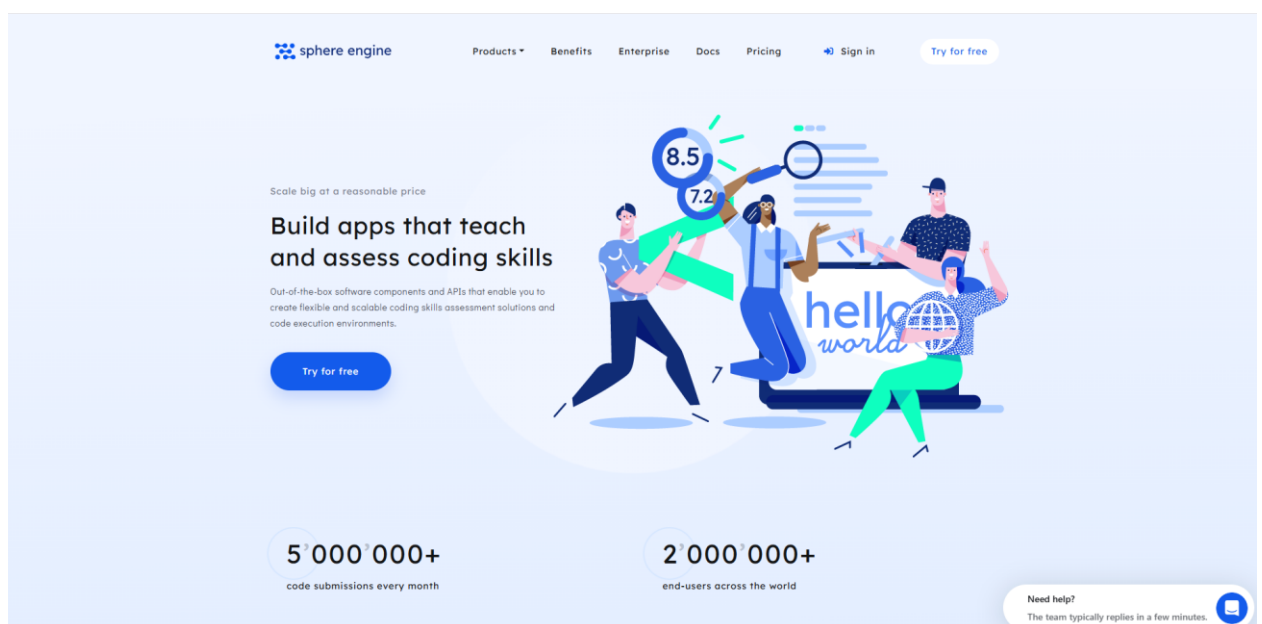


Рис. 1.4. Інтерфейс сервісу Sphere Engine

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Sphere Engine підтримує значну кількість мов програмування та забезпечує масштабовану хмарну інфраструктуру для виконання програм. Основною перевагою є готовність до інтеграції з іншими інформаційними системами. Недоліком є комерційна модель використання та залежність від стороннього постачальника послуг.

Для порівняння розглянутих аналогів складено таблицю 1.1.

Таблиця 1.1

Порівняння існуючих систем виконання програмного коду

Характеристика	Judge0	HackerRank	LeetCode	Sphere Engine
Виконання коду	+	+	+	+
REST API	+	-	-	+
Історія запусків	-	+	+	-
Власне розгортання	+	-	-	-
Реєстрація та авторизація	-	+	+	+
Хмарна інфраструктура	-	+	+	+

Як видно з таблиці 1.1, найбільш близьким за концепцією до розроблюваного програмного комплексу є система Judge0. Проте навіть вона не повністю відповідає поставленим вимогам, оскільки в рамках кваліфікаційної роботи необхідно створити власний мікросервіс з використанням сучасної мікросервісної архітектури, окремого сервісу обробки задач, системи черг повідомлень та інтегрованого вебінтерфейсу.

Проведений аналіз показав, що існуючі рішення успішно реалізують функції виконання програмного коду, однак більшість із них є або закритими комерційними продуктами, або орієнтовані на вузькоспеціалізовані задачі. Тому розробка власного мікросервісу для перевірки та виконання програмного коду є актуальним завданням та дозволяє реалізувати архітектуру, яка повністю відповідає поставленим вимогам.

1.3 Вибір архітектури мікросервісу для перевірки та виконання програмного коду

Одним із найважливіших етапів розробки програмного забезпечення є вибір архітектури системи. Від архітектурних рішень залежать масштабованість, продуктивність, надійність та можливість подальшого розвитку програмного комплексу. Для систем виконання програмного коду особливого значення набувають питання безпеки, ізоляції середовища виконання та обробки великої кількості незалежних задач. [21; 22; 23; 24]

Існує декілька підходів до побудови подібних систем. Найпростішим рішенням є монолітна архітектура, при якій усі компоненти системи реалізуються в межах одного застосунку. У такому випадку вебінтерфейс, бізнес-логіка, виконання програмного коду та робота з базою даних знаходяться в одному програмному комплексі.

Таблиця 1.2

Порівняння монолітної та мікросервісної архітектур

Характеристика	Монолітна архітектура	Мікросервісна архітектура
Простота розробки	Висока	Середня
Масштабованість	Обмежена	Висока
Відмовостійкість	Низька	Висока
Незалежність компонентів	Низька	Висока
Гнучкість розвитку	Середня	Висока
Безпечна ізоляція сервісів	Обмежена	Висока

Як видно з таблиці 1.2, монолітна архітектура є простішою на початкових етапах розробки, проте має суттєві обмеження щодо

масштабування та розширення функціональності. Для системи виконання програмного коду така архітектура є менш придатною, оскільки процес запуску користувацьких програм може створювати додаткове навантаження на основний серверний застосунок.

З огляду на особливості предметної області для реалізації програмного комплексу було обрано мікросервісну архітектуру. Даний підхід передбачає поділ системи на окремі незалежні компоненти, кожен з яких виконує визначений набір функцій та може розгортатися окремо від інших компонентів.

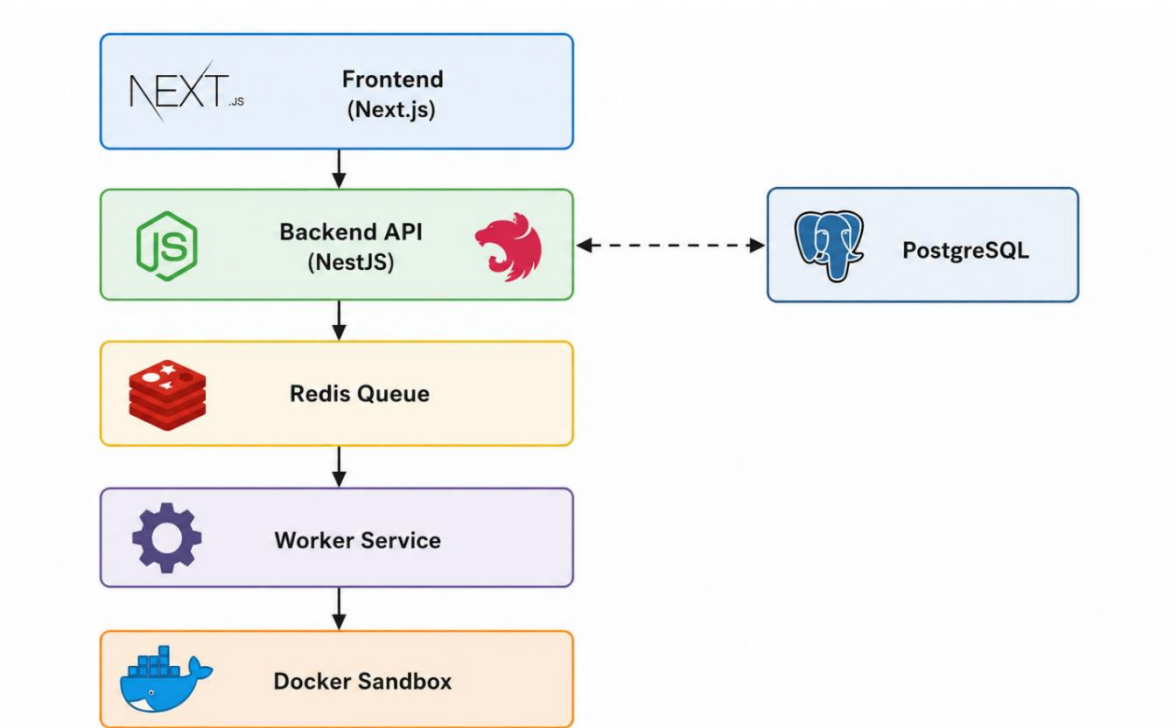


Рис. 1.5. Загальна архітектура мікросервісу перевірки та виконання програмного коду

На рис. 1.5 наведено загальну архітектуру розроблюваного програмного комплексу. Взаємодія користувача із системою здійснюється через клієнтську частину, реалізовану за допомогою фреймворку Next.js. Користувач вводить програмний код через вебінтерфейс та надсилає його на сервер для подальшої обробки.

Серверна частина системи реалізована за допомогою фреймворку NestJS. Основним завданням сервера є прийом запитів, перевірка коректності вхідних даних, збереження інформації до бази даних та постановка задач до черги обробки.

Для передачі задач між компонентами використовується Redis. Даний компонент виконує роль брокера повідомлень та забезпечує асинхронну взаємодію між серверною частиною та сервісом виконання задач. Такий підхід дозволяє уникнути блокування користувацьких запитів під час виконання програм.

Обробка задач виконується окремим сервісом Worker. Після отримання завдання з черги Worker створює контейнер Docker та передає до нього програмний код користувача. Використання окремого сервісу виконання дозволяє ізолювати ресурсоємні операції від основного API-сервера.

Для забезпечення безпеки виконання програм використовується Docker-контейнеризація. Кожна програма запускається в окремому контейнері з обмеженим доступом до ресурсів операційної системи. Після завершення роботи контейнер автоматично видаляється, що виключає можливість накопичення тимчасових даних та підвищує рівень безпеки системи.

Інформація про користувачів, задачі та результати виконання програм зберігається в реляційній базі даних PostgreSQL. Використання СКБД дозволяє забезпечити цілісність даних та ефективне виконання запитів.

Запропонована архітектура забезпечує масштабованість програмного комплексу. У разі збільшення навантаження можливо розгорнути декілька екземплярів Worker-сервісу без внесення змін до інших компонентів системи. Це дозволяє збільшувати продуктивність системи шляхом горизонтального масштабування.

Таким чином, використання мікросервісної архітектури дозволяє забезпечити безпечне виконання програмного коду, спростити супровід програмного комплексу та створити основу для подальшого розвитку функціональних можливостей системи.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Важливим етапом розробки програмного забезпечення є вибір технологій реалізації. Від правильності вибору програмних засобів залежить продуктивність системи, зручність супроводу, масштабованість та можливість подальшого розвитку програмного продукту. Для реалізації мікросервісу перевірки та виконання програмного коду було обрано сучасний стек технологій, який відповідає вимогам щодо безпеки, швидкодії та підтримки мікросервісної архітектури.

Використання NestJS дозволяє логічно розділяти функціональність системи між окремими модулями та забезпечує високу підтримуваність програмного коду.

Для збереження даних використовується PostgreSQL. Дана система керування базами даних є однією з найпоширеніших реляційних СКБД та широко застосовується при створенні сучасних вебсервісів.

Таблиця 1.3

Порівняння популярних систем керування базами даних

Характеристика	PostgreSQL	MySQL	SQLite
Відкритий код	Так	Так	Так
Підтримка транзакцій	Так	Так	Частково
Масштабованість	Висока	Висока	Низька
Робота з великими навантаженнями	Так	Так	Ні
Мережевий доступ	Так	Так	Ні

Як видно з таблиці 1.3, PostgreSQL забезпечує необхідний рівень функціональності для роботи багатокористувацького вебсервісу та дозволяє ефективно зберігати результати виконання програм.

Для організації взаємодії між компонентами системи використовується Redis. Даний програмний продукт є високопродуктивним сховищем типу «ключ-значення» та часто використовується як брокер повідомлень і система кешування.

У розробленому програмному комплексі Redis використовується для реалізації черги задач на виконання програмного коду. Після отримання запиту сервер додає задачу до черги, а сервіс Worker отримує її для подальшої обробки.

Таблиця 1.4

Обґрунтування вибору технологій реалізації

Технологія	Призначення
Next.js	Клієнтська частина
NestJS	Серверна частина
PostgreSQL	Зберігання даних
Redis	Черга повідомлень
Bull	Керування задачами
Docker	Ізольоване виконання коду
TypeScript	Основна мова розробки

Для реалізації програмного комплексу було обрано сучасний стек технологій, який забезпечує можливість побудови масштабованої, надійної та безпечної системи. Клієнтська частина реалізується за допомогою фреймворку Next.js, що дозволяє створювати сучасні вебзастосунки з високою продуктивністю та зручним інтерфейсом користувача. Для розробки серверної частини використовується NestJS, який підтримує модульну архітектуру та спрощує створення мікросервісних систем.

Таблиця 1.5

Мінімальні вимоги до серверного обладнання

Параметр	Значення
Процесор	4 ядра
Оперативна пам'ять	8 ГБ
Вільне місце на диску	20 ГБ
Операційна система	Linux Ubuntu Server
Docker Engine	Версія 24+

Зазначені вимоги є мінімально необхідними для коректного функціонування програмного комплексу. У разі збільшення кількості одночасних користувачів або задач, що виконуються, апаратні ресурси сервера можуть бути масштабовані відповідно до навантаження.

Таблиця 1.6

Мінімальні вимоги до клієнтського пристрою

Параметр	Значення
Процесор	Двоядерний
Оперативна пам'ять	4 ГБ
Браузер	Chrome, Firefox, Edge
Доступ до Інтернету	Так

Таким чином, проведений аналіз інструментальних засобів показав, що використання Next.js, NestJS, PostgreSQL, Redis та Docker дозволяє реалізувати сучасний мікросервіс для перевірки та виконання програмного коду, який відповідає вимогам безпеки, масштабованості та продуктивності. Обраний стек технологій забезпечує можливість подальшого розвитку програмного комплексу та розширення його функціональних можливостей.

Висновки до першого розділу

У першому розділі було проведено аналіз особливостей побудови систем для перевірки та виконання програмного коду в ізольованому середовищі. Визначено актуальність використання подібних програмних комплексів у сфері дистанційного навчання, автоматизованої перевірки програмних завдань, онлайн-компіляторів та платформ для оцінювання навичок програмування.

Проведено аналіз існуючих програмних продуктів, серед яких Judge0, HackerRank, LeetCode та Sphere Engine. Дослідження показало, що сучасні рішення забезпечують широкий набір функціональних можливостей для виконання та перевірки програмного коду, проте більшість із них мають обмеження щодо інтеграції, модифікації функціоналу або є комерційними продуктами.

В результаті аналізу було обґрунтовано доцільність використання мікросервісної архітектури для реалізації програмного комплексу. Такий підхід забезпечує незалежність окремих компонентів системи, спрощує масштабування та підвищує рівень відмовостійкості.

Також було виконано вибір інструментальних засобів реалізації. Для створення клієнтської частини обрано фреймворк Next.js, серверної частини — NestJS, для зберігання даних — PostgreSQL, для організації обробки задач — Redis та Bull, а для ізольованого виконання програмного коду — Docker. Обраний набір технологій відповідає сучасним вимогам до безпеки, продуктивності та масштабованості веборієнтованих систем.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ МІКРОСЕРВІСУ ДЛЯ ПЕРЕВІРКИ ТА ВИКОНАННЯ ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Після визначення вимог до програмного комплексу необхідно виконати проєктування основних сценаріїв взаємодії користувачів із системою та сформувати узагальнену структуру програмного комплексу. Даний етап дозволяє визначити функціональні можливості системи, взаємозв'язки між окремими компонентами та підготувати основу для подальшої реалізації програмного забезпечення.

Основним користувачем розроблюваного мікросервісу є зареєстрований користувач системи, який має можливість створювати задачі на виконання програмного коду, переглядати результати виконання та працювати з історією запусків. Взаємодія із системою здійснюється через вебінтерфейс, реалізований на базі технології Next.js.

До основних функціональних можливостей користувача належать:

- реєстрація та авторизація в системі;
- введення програмного коду через вебінтерфейс;
- запуск програмного коду на виконання;
- отримання результатів виконання;
- перегляд історії запусків;
- перегляд повідомлень про помилки та перевищення ліміту часу виконання.

Для відображення взаємодії користувача із системою використовується UML-діаграма варіантів використання. [25; 26]

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24



Рис. 2.1. UML-діаграма варіантів використання системи

На рис. 2.1 наведено основні сценарії взаємодії користувача з програмним комплексом. Діаграма демонструє функціональні можливості системи та дозволяє визначити перелік операцій, які повинні бути реалізовані в процесі розробки програмного забезпечення.

Для виконання програмного коду система використовує декілька взаємопов'язаних компонентів. Після надсилання коду на сервер створюється нова задача виконання, яка передається до черги обробки. Після цього Worker-сервіс отримує задачу та запускає контейнер Docker для виконання користувацької програми.

Послідовність взаємодії компонентів системи наведено на UML-діаграмі послідовності.

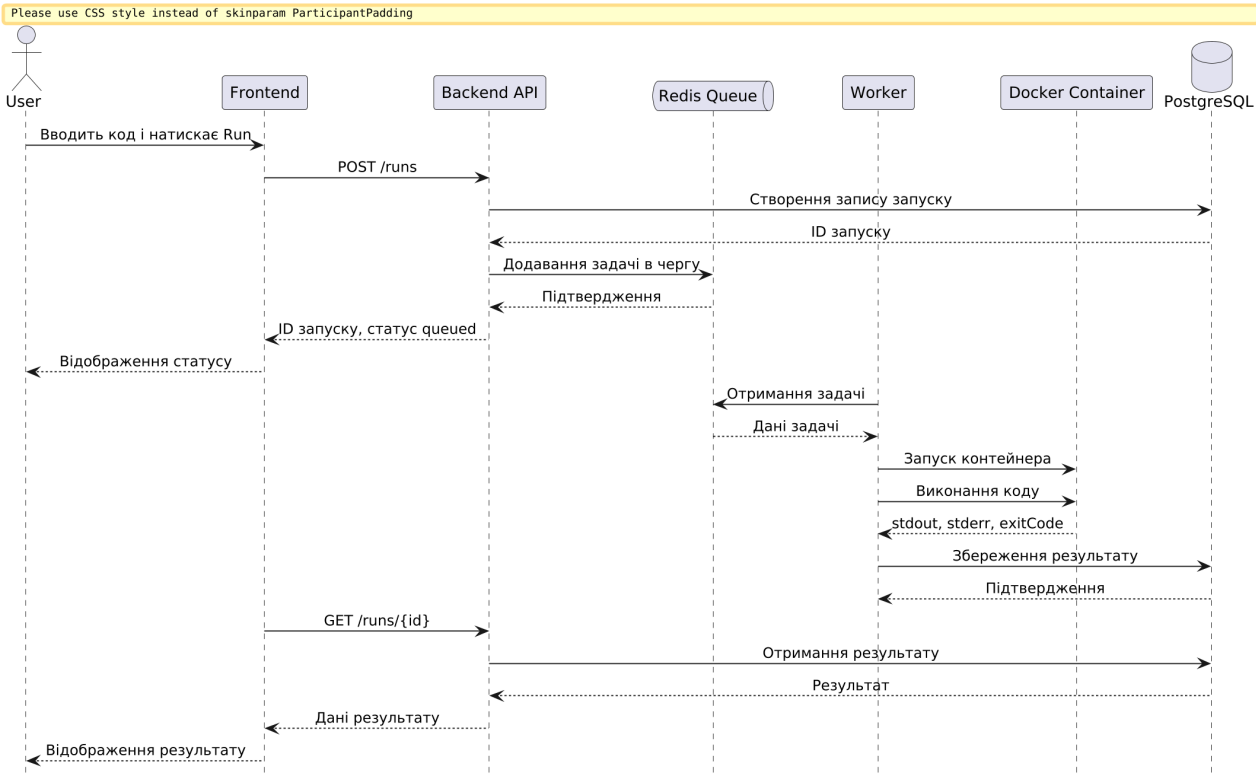


Рис. 2.2. UML-діаграма послідовності виконання програмного коду

На рис. 2.2 показано процес виконання програмного коду від моменту відправлення запиту користувачем до отримання результату виконання. Такий підхід дозволяє забезпечити асинхронну обробку задач та мінімізувати навантаження на серверну частину системи. [25; 26]

UML-діаграма також демонструє взаємодію між основними компонентами програмного комплексу під час виконання користувацького програмного коду. Клієнтська частина відповідає за приймання даних від користувача та відображення результатів виконання, тоді як серверна частина забезпечує обробку запитів, створення записів запуску та взаємодію з іншими сервісами системи. Такий підхід забезпечує розподіл навантаження між компонентами та підвищує надійність роботи програмного комплексу.

Для опису структури програмного комплексу доцільно використовувати UML-діаграму компонентів.

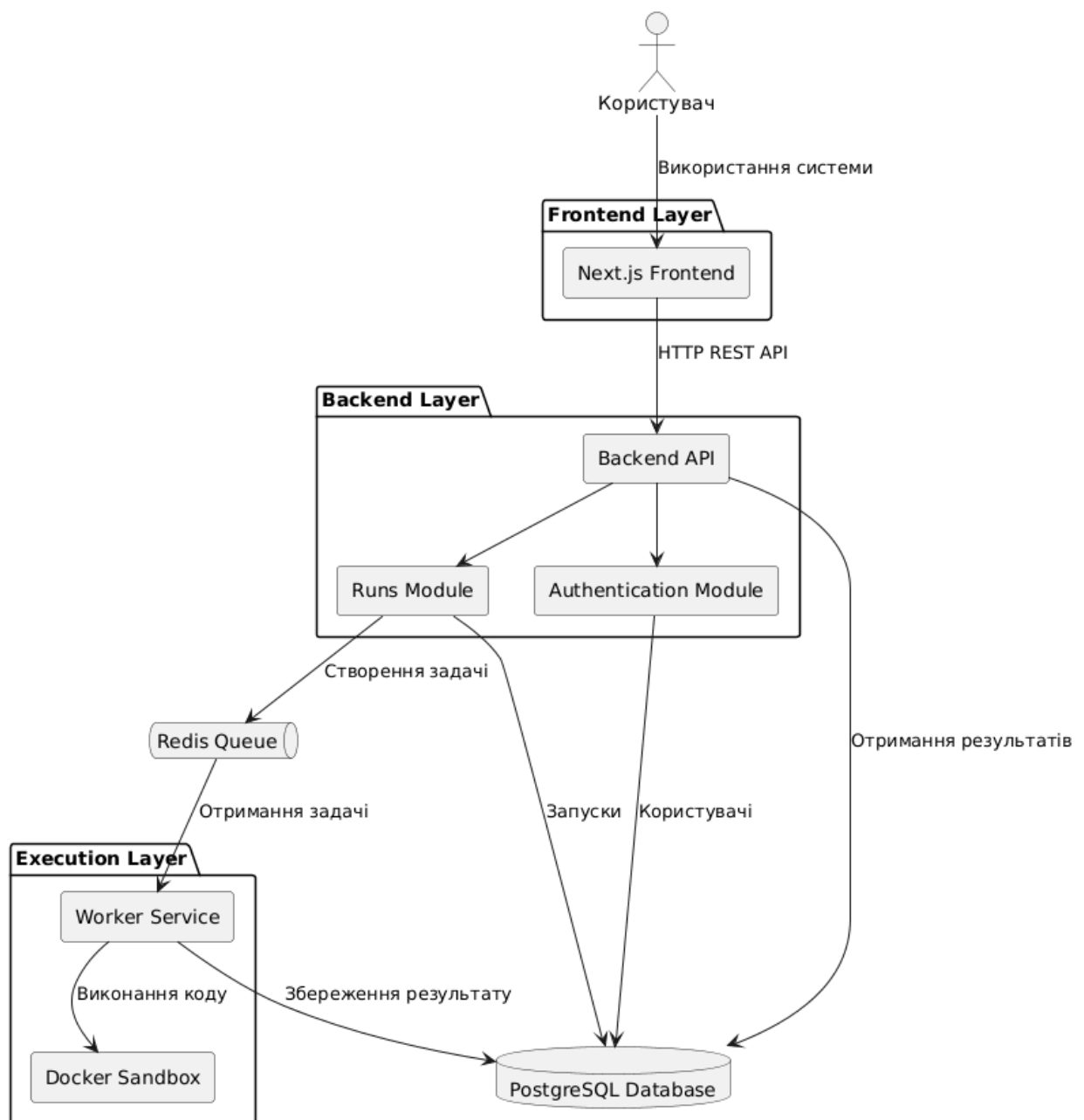


Рис. 2.3. UML-діаграма компонентів системи

На рис. 2.3 наведено загальну структуру компонентів програмного комплексу та зв'язки між ними. Представлена архітектура дозволяє незалежно розвивати окремі модулі системи та спрощує супровід програмного забезпечення.

Для подальшого проєктування необхідно визначити основні сутності предметної області та взаємозв'язки між ними. В рамках розроблюваної системи до основних сутностей належать користувач та запуск програми.

Інформація про результати виконання програм зберігається в межах сутності Run і не виділяється в окрему сутність.

Таблиця 2.1

Основні сутності системи

Сутність	Призначення
User	Дані користувача
Run	Інформація про запуск програми

Представлені сутності формують основу структури даних програмного комплексу та використовуються для подальшого проєктування бази даних і програмних модулів системи.

У межах проєктування програмного комплексу було визначено основні сценарії використання системи, побудовано логічну структуру взаємодії компонентів та сформовано перелік основних сутностей предметної області. На основі отриманих результатів доцільно сформувати функціональні та нефункціональні вимоги до системи, які визначають її можливості, обмеження та критерії якості програмного забезпечення.

Одним із перших етапів проєктування програмного комплексу є формування функціональних та нефункціональних вимог. Вони визначають можливості системи, очікувану поведінку та обмеження, яких необхідно дотримуватись під час розробки.

Функціональні вимоги визначають набір операцій, які система повинна виконувати для забезпечення роботи користувачів та внутрішніх компонентів.

Таблиця 2.2

Функціональні вимоги до системи

Код	Опис вимоги
FR-01	Реєстрація нового користувача
FR-02	Авторизація користувача
FR-03	Створення запуску програмного коду

FR-04	Передача задач до черги обробки
FR-05	Виконання програмного коду
FR-06	Отримання результатів виконання
FR-07	Збереження історії запусків
FR-08	Відображення повідомлень про помилки
FR-09	Контроль часу виконання
FR-10	Перегляд історії запусків

Функціональні вимоги охоплюють усі основні сценарії взаємодії користувача із системою та визначають перелік можливостей, які повинні бути реалізовані в програмному комплексі.

Окрім функціональних вимог, система повинна відповідати низці нефункціональних вимог, що характеризують якість роботи програмного забезпечення.

Таблиця 2.3

Нефункціональні вимоги до системи

Код	Опис вимоги
NFR-01	Забезпечення безпечного виконання коду
NFR-02	Масштабованість архітектури
NFR-03	Висока доступність сервісів
NFR-04	Підтримка контейнеризації
NFR-05	Підтримка багатокористувацької роботи
NFR-06	Простота розгортання
NFR-07	Можливість подальшого розширення
NFR-08	Стійкість до помилок користувача

Однією з найважливіших нефункціональних вимог є безпека виконання програмного коду. Оскільки система дозволяє запускати код, створений

користувачем, необхідно забезпечити його ізоляцію від основної операційної системи сервера.

Для виконання цієї вимоги використовується Docker-контейнеризація, що дозволяє обмежити доступ програм до файлової системи, мережевих ресурсів та інших процесів сервера.

Наступною важливою вимогою є масштабованість. Архітектура системи повинна дозволити збільшення кількості сервісів обробки задач без внесення суттєвих змін до програмного коду.

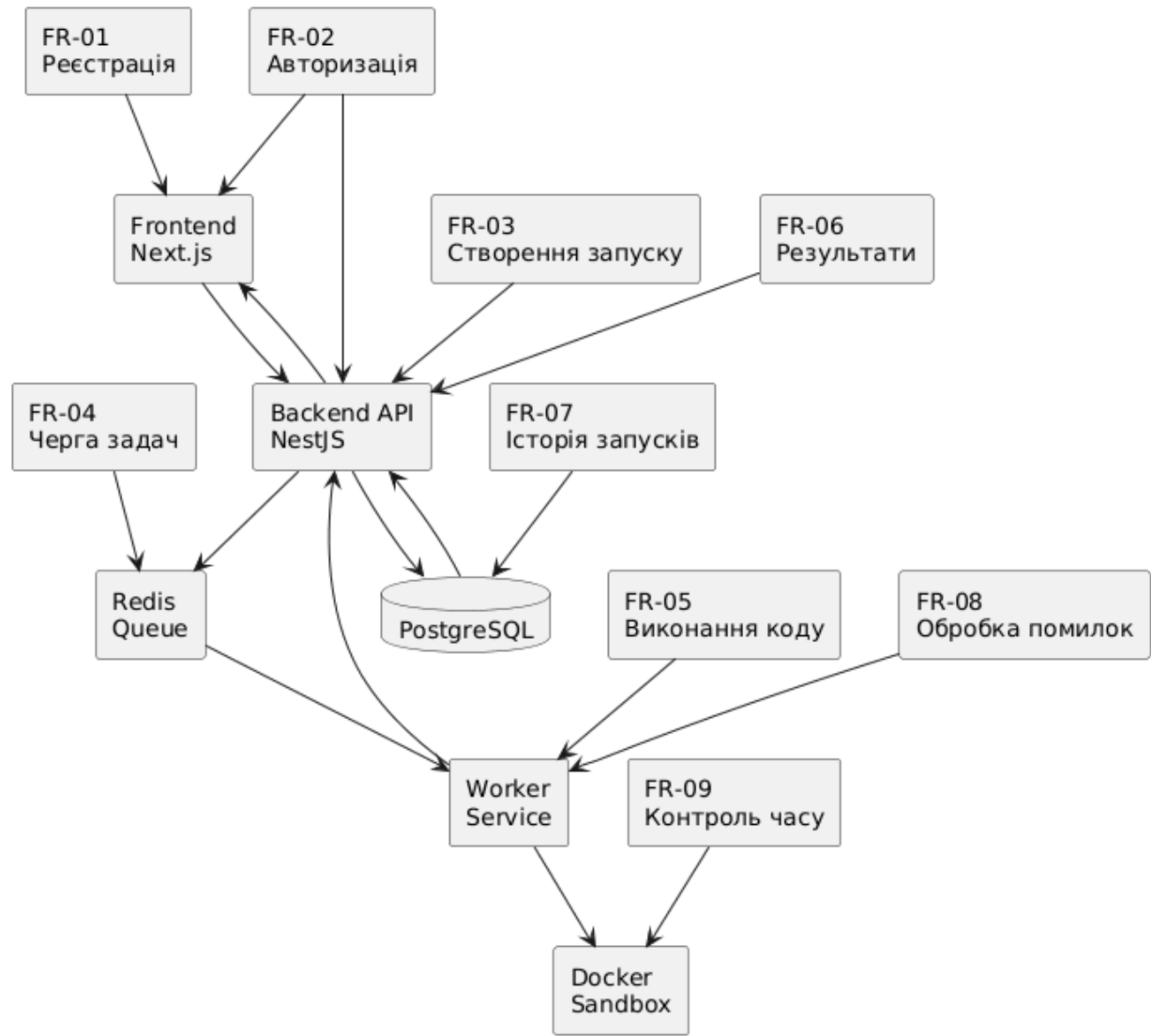


Рис. 2.4. Схема відповідності вимог компонентам системи

На рис. 2.4 наведено взаємозв'язок між основними вимогами та компонентами програмного комплексу. Представлена схема демонструє, яким чином окремі сервіси забезпечують виконання поставлених вимог.

Для оцінки відповідності системи поставленим вимогам було сформовано матрицю трасування.

Таблиця 2.4

Матриця відповідності вимог та компонентів

Вимога	Frontend	Backend	Redis	Worker	Docker
Авторизація	+	+	-	-	-
Створення запуску	+	+	+	-	-
Виконання коду	-	+	+	+	+
Історія запусків	+	+	-	-	-
Контроль часу	-	-	-	+	+
Безпека	-	+	-	+	+

Представлена матриця дозволяє простежити реалізацію кожної вимоги в межах програмного комплексу та підтверджує повноту проєктування системи.

Таким чином, визначені функціональні та нефункціональні вимоги стали основою для проєктування архітектури програмного комплексу та реалізації його програмних компонентів.

2.2 Розробка бази даних системи

Однією з ключових складових будь-якої інформаційної системи є база даних. Саме вона забезпечує зберігання інформації про користувачів, результати роботи програмного комплексу та історію виконання задач. Для розроблюваного мікросервісу було обрано реляційну модель зберігання даних, реалізовану за допомогою системи керування базами даних PostgreSQL.

Основним завданням бази даних є забезпечення надійного збереження інформації про користувачів системи, створені ними задачі та результати виконання програмного коду. При проєктуванні структури даних враховувались вимоги до масштабованості, підтримки цілісності даних та

можливості подальшого розширення функціональності програмного комплексу.

На початковому етапі було визначено перелік основних сутностей предметної області та взаємозв'язки між ними.

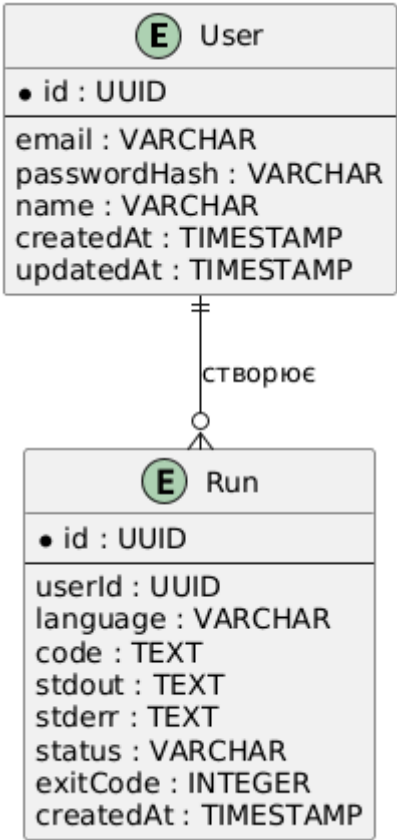


Рис. 2.5. ER-діаграма бази даних системи

На рис. 2.5 наведено логічну структуру бази даних програмного комплексу. Центральною сутністю є користувач системи, який може створювати необмежену кількість запусків програмного коду. Дані про кожен запуск зберігаються у сутності Run та включають мову програмування, вихідний код, статус виконання, а також результати виконання програми, зокрема повідомлення про помилки та службові параметри виконання. Для зберігання інформації про користувачів використовується таблиця User.

Структура таблиці User

Поле	Тип даних	Призначення
id	UUID	Унікальний ідентифікатор
email	VARCHAR	Адреса електронної пошти
passwordHash	VARCHAR	Хеш пароля
name	VARCHAR	Ім'я користувача
createdAt	TIMESTAMP	Дата створення запису
updatedAt	TIMESTAMP	Дата останнього оновлення

Таблиця User містить облікові дані користувачів системи та використовується для реалізації механізмів автентифікації та авторизації.

Інформація про запуск програмного коду зберігається в таблиці Run.

Структура таблиці Run

Поле	Тип даних	Призначення
id	UUID	Ідентифікатор запуску
userId	UUID	Власник запуску
language	VARCHAR	Мова програмування
code	TEXT	Програмний код
stdout	TEXT	Стандартний вивід програми
stderr	TEXT	Повідомлення про помилки
status	VARCHAR	Поточний стан виконання
exitCode	INTEGER	Код завершення програми
createdAt	TIMESTAMP	Час створення

Результати виконання програми зберігаються безпосередньо в таблиці Run у полях stdout, stderr та exitCode, тому окрема таблиця результатів виконання не використовується.

Основні індекси бази даних

Таблиця	Поле
User	email
Run	userId

Використання індексів дозволяє значно прискорити виконання запитів під час пошуку історії запусків користувача та отримання результатів виконання програм. [27]

Фізична структура взаємодії серверної частини із базою даних наведена на наступній схемі.

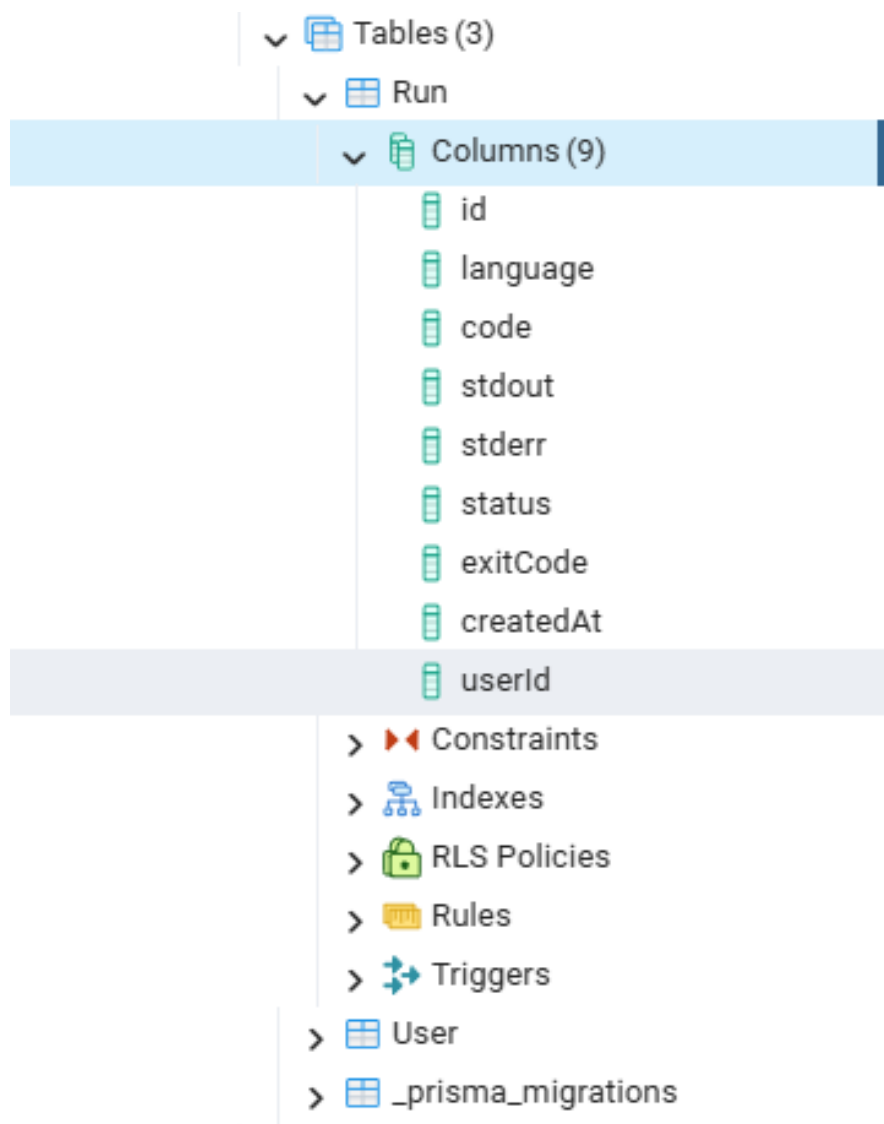


Рис. 2.6. Взаємодія Backend API з PostgreSQL

На рис. 2.6 показано взаємодію серверного застосунку з базою даних. Усі операції зі створення користувачів, запусків програм та збереження результатів виконуються через серверну частину системи, що дозволяє забезпечити контроль цілісності даних та дотримання правил безпеки.

Для контролю структури бази даних та її версій використовується механізм міграцій. Це дозволяє автоматизувати процес оновлення структури таблиць під час розгортання нових версій програмного комплексу та спрощує супровід проєкту.

Таким чином, спроектована структура бази даних забезпечує зберігання всієї необхідної інформації для функціонування мікросервісу перевірки та виконання програмного коду. Запропонована модель даних є масштабованою, підтримує цілісність інформації та дозволяє реалізувати всі функціональні можливості програмного комплексу. Використання реляційної бази даних забезпечує надійне зберігання відомостей про користувачів і результати запусків програм, а також спрощує подальший супровід та розвиток системи. Розроблена структура може бути розширена додатковими сутностями та зв'язками без суттєвих змін існуючої архітектури програмного комплексу.

2.3 Проєктування та реалізація алгоритмів роботи системи

Для забезпечення коректного функціонування мікросервісу перевірки та виконання програмного коду необхідно реалізувати алгоритми взаємодії між окремими компонентами системи. Основними задачами програмного комплексу є приймання програмного коду від користувача, створення задачі на виконання, передача задачі до черги обробки, запуск коду в ізольованому середовищі та повернення результатів виконання.

Процес роботи системи складається з декількох послідовних етапів, кожен з яких виконується окремим компонентом архітектури. Використання асинхронної моделі обробки дозволяє мінімізувати навантаження на серверну частину та забезпечити стабільну роботу системи навіть при великій кількості одночасних користувачів.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Загальний алгоритм роботи програмного комплексу наведено на рис. 2.7.



Рис. 2.7. Загальний алгоритм роботи системи

На рис. 2.7 наведено узагальнену схему роботи програмного комплексу. Після надсилання програмного коду користувачем система створює нову задачу виконання та передає її до черги обробки. Подальше виконання здійснюється незалежно від користувацького інтерфейсу, що дозволяє реалізувати асинхронну обробку запитів.

Першим етапом роботи системи є створення задачі на виконання програмного коду.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				36
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис. 2.8. Алгоритм створення задачі на виконання

На рис. 2.8 представлено алгоритм формування нової задачі. Після отримання програмного коду сервер виконує первинну перевірку вхідних даних, створює запис у базі даних та додає задачу до черги обробки. Користувачу повертається унікальний ідентифікатор запуску, за допомогою якого можна отримати результат виконання.

Наступним етапом є обробка задачі сервісом Worker.



Рис. 2.9. Алгоритм роботи Worker-сервісу

На рис. 2.9 наведено алгоритм роботи Worker-сервісу. Даний компонент виконує основну частину обчислень та відповідає за запуск програмного коду в ізольованому середовищі.

Однією з ключових вимог до системи є забезпечення безпечного виконання користувацьких програм. Для цього використовується Docker-контейнеризація.



Рис. 2.10. Алгоритм виконання програмного коду в Docker-контейнері

На рис. 2.10 показано процес виконання програмного коду в ізолюваному середовищі. Кожен запуск здійснюється в окремому контейнері, що забезпечує високий рівень безпеки та ізоляції від основної операційної системи.

Окремої уваги потребує механізм контролю часу виконання програм. Користувач може випадково створити нескінченний цикл або навмисно намагатися зайняти обчислювальні ресурси сервера.

Для запобігання подібним ситуаціям використовується алгоритм обмеження часу виконання.



Рис. 2.11. Алгоритм контролю часу виконання

На рис. 2.11 наведено алгоритм контролю часу виконання. Якщо встановлений ліміт перевищено, контейнер примусово завершується, а користувачу повертається повідомлення про перевищення допустимого часу виконання.

Для забезпечення коректної роботи системи необхідно також реалізувати механізм обробки помилок.

Таблиця 2.8

Типи помилок системи

Тип помилки	Причина
Validation Error	Некоректні вхідні дані
Syntax Error	Помилка синтаксису програми

Runtime Error	Помилка під час виконання
Timeout Error	Перевищення часу виконання
Queue Error	Помилка роботи черги
Container Error	Помилка Docker-контейнера

Кожен тип помилки обробляється окремим механізмом та супроводжується відповідним повідомленням для користувача. Це дозволяє спростити пошук проблем та підвищує зручність використання програмного комплексу.

Для демонстрації взаємодії компонентів системи доцільно використовувати UML-діаграму послідовності.

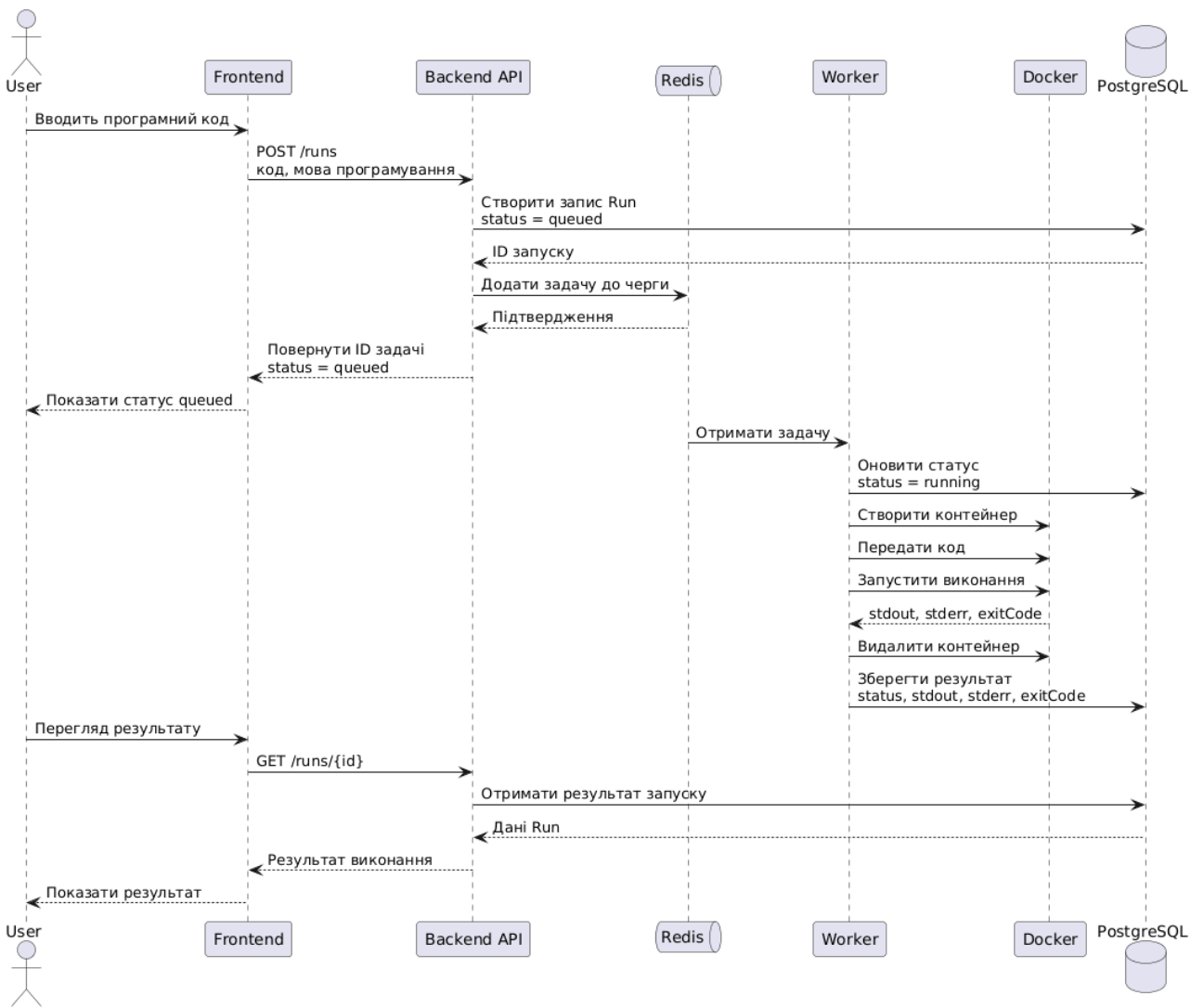


Рис. 2.12. UML-діаграма послідовності виконання задачі

На рис. 2.12 наведено послідовність обміну повідомленнями між компонентами системи під час виконання програмного коду. Представлена схема демонструє асинхронний характер роботи програмного комплексу та підтверджує відповідність обраної архітектури вимогам масштабованості.

Таким чином, були спроектовані основні алгоритми роботи програмного комплексу. Розроблені алгоритми забезпечують приймання програмного коду, асинхронну обробку задач, виконання програм в ізолюваному середовищі та повернення результатів користувачу. Запропоновані рішення створюють основу для реалізації мікросервісу перевірки та виконання програмного коду та забезпечують його безпечну та стабільну роботу.

2.4 Реалізація програмного комплексу

Після завершення етапів аналізу та проектування було виконано реалізацію програмного комплексу у вигляді мікросервісу для перевірки та виконання програмного коду в ізолюваному середовищі. Програмний продукт побудовано відповідно до розробленої архітектури та складається з клієнтської частини, серверної частини, сервісу обробки задач, бази даних та середовища виконання програмного коду.

Для реалізації клієнтської частини було використано фреймворк Next.js та мову програмування TypeScript. Основним призначенням клієнтської частини є забезпечення взаємодії користувача із системою, введення програмного коду, запуск задач на виконання та перегляд отриманих результатів. Клієнтська частина містить сторінки авторизації, реєстрації, профілю користувача, головну сторінку та сторінку перегляду результату запуску програмного коду. Повторно використовувані елементи інтерфейсу винесено в окремі компоненти, що спрощує супровід і подальше розширення вебзастосунку.

Для реалізації серверної логіки використано фреймворк NestJS. Серверна частина відповідає за обробку HTTP-запитів, взаємодію з базою даних, авторизацію користувачів та формування задач на виконання

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

програмного коду. Вона побудована за модульним принципом і містить окремі модулі для роботи з авторизацією, запуском програм, доступом до бази даних та перевіркою працездатності сервісу. Такий підхід дозволяє розділити відповідальність між окремими частинами системи та спростити подальший супровід програмного комплексу. [28; 29; 30; 31]

Одним із центральних компонентів системи є модуль виконання програмного коду.

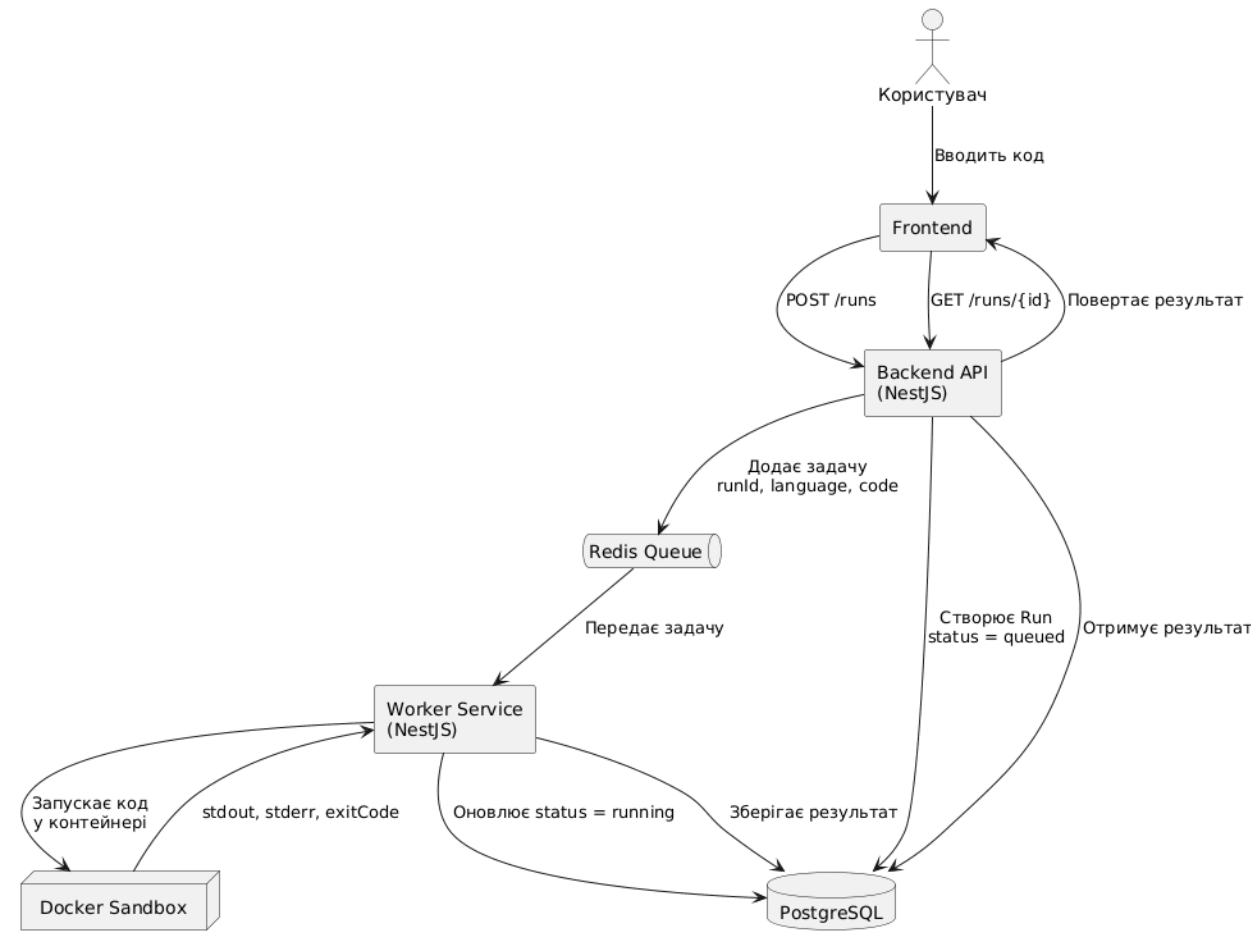


Рис. 2.13. Схема взаємодії Backend API та Worker

На рис. 2.13 показано процес передачі задачі від серверної частини до сервісу виконання. Після створення нового запуску задача потрапляє до Redis Queue, звідки її отримує Worker-сервіс для подальшої обробки.

Для збереження інформації про виконання програмного коду використовується база даних PostgreSQL.

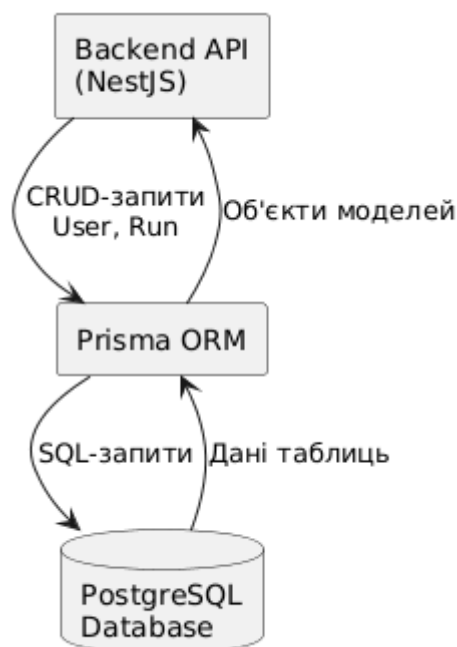


Рис. 2.14. База даних проєкту в середовищі pgAdmin або DBeaver

На рис. 2.14 наведено структуру бази даних розробленого програмного комплексу. Вона забезпечує зберігання інформації про користувачів, задачі та результати виконання програм.

Для забезпечення безпечного виконання користувацького коду використовується контейнеризація Docker.

Container CPU usage ⓘ

0.62% / 1600% (16 CPUs available)

Container memory usage ⓘ

25.09MB / 14.86GB

☰

☐ Only show running containers

☐		Name	Container ID	Image
☐	▼ ⓘ	coderrunner	-	-
☐	●	postgres	8154a6b9d8f1	postgres:16-alpine
☐	●	redis	f803d1e40e19	redis:7-alpine
☐	○	worker	42ce08a109bd	coderrunner-worker
☐	○	frontend	357faeef7cea	coderrunner-frontend
☐	○	backend	c32f1258415a	coderrunner-backend

Рис. 2.15. Робота Docker-контейнера під час виконання програми

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

На рис. 2.15 показано приклад роботи контейнеризованого середовища виконання. Для кожного запуску створюється окремий контейнер, який після завершення роботи автоматично видаляється.

Важливою частиною реалізації є механізм створення задач на виконання програмного коду.

```
async run(body: CreateRunDto, userId: string | null) {
  const run = await this.prisma.run.create({
    data: {
      language: body.language,
      code: body.code,
      status: 'queued',
      userId,
    },
  });

  await this.codeExecutionQueue.add('execute', {
    runId: run.id,
    language: body.language,
    code: body.code,
  });

  return {
    status: 'queued',
    runId: run.id,
  };
}
```

Лістинг 2.1. Метод створення нового запуску програми

Представлений фрагмент програмного коду відповідає за приймання програмного коду від користувача, створення нового запису в базі даних та додавання задачі до черги обробки.

Наступним важливим елементом є Worker-сервіс.

```
async process(job: Job) {
  const { runId, language, code } = job.data;

  if (!runId) {
    return;
  }

  await this.updateRun(runId, {
```

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

        status: 'running',
        stdout: '',
        stderr: '',
        exitCode: null,
    });

    try {
        const result =
            language === 'cpp'
                ? await this.executeCpp(code)
                : await this.executeDirect(language, code);

        await this.updateRun(runId, result);
    } catch (error: any) {
        await this.updateRun(runId, {
            status: error.timedOut ? 'timeout' : 'failed',
            stdout: error.stdout ?? '',
            stderr: error.timedOut
                ? 'Execution timeout exceeded'
                : (error.stderr ?? error.message),
            exitCode: error.exitCode ?? 1,
        });
    }
}

```

Лістинг 2.2. Обробник задач Worker-сервісу

У наведеному фрагменті реалізовано логіку отримання задач із Redis Queue та запуску процесу виконання програмного коду.

Для запуску програм використовується Docker-контейнер.

```

const result = await execa(
    'docker',
    [
        'run',
        '--rm',
        '--memory=128m',
        '--cpus=0.5',
        '--network=none',
        '--pids-limit=64',
        '--read-only',
        '--tmpfs',
        '/tmp:rw,size=64m',
        config.image,
        config.command,
    ],
);

```

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        ...config.args,
        code,
    ],
    {
        timeout: 5000,
    },
);

```

Лістинг 2.3. Формування та запуск Docker-контейнера

Даний алгоритм створює ізольоване середовище виконання, записує програмний код користувача до тимчасового файлу та запускає відповідний інтерпретатор.

Важливим елементом системи є механізм обмеження часу виконання.

```

const result = await execa(
    'docker',
    [
        'run',
        '--rm',
        '--memory=256m',
        '--cpus=0.5',
        '--network=none',
        '--pids-limit=64',
        '--read-only',
        '--tmpfs',
        '/tmp:rw,size=64m,exec',
        '-v',
        `${dockerPath}:/sandbox:ro`,
        'coderunner-cpp-sandbox',
        'sh',
        '-c',
        compileAndRun,
    ],
    {
        timeout: 8000,
    },
);

return {
    status: 'completed',
    stdout: result.stdout,
    stderr: result.stderr,
    exitCode: result.exitCode ?? 0,
};

```

```

}
} finally {
await rm(tempDir, {
recursive: true,
force: true,
});
}
}

```

Лістинг 2.4. Реалізація контролю часу виконання програми

Реалізований механізм дозволяє автоматично завершувати виконання програм, які перевищують встановлений часовий ліміт.

Для взаємодії користувача із системою використовується вебінтерфейс із редактором програмного коду.

2.5 Проектування компонентної структури програмного комплексу

Для забезпечення гнучкості, масштабованості та спрощення супроводу програмного забезпечення під час розробки мікросервісу було використано компонентний підхід. Кожен компонент системи відповідає за виконання окремого набору функцій та взаємодіє з іншими компонентами через стандартизовані інтерфейси.

Використання компонентної архітектури дозволяє зменшити залежність між окремими частинами програмного комплексу, спростити тестування та забезпечити можливість незалежного оновлення окремих сервісів.

Основними компонентами розробленої системи є:

- клієнтський застосунок;
- серверний API;
- сервіс виконання задач;
- система черг повідомлень;
- база даних;
- середовище виконання програмного коду.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

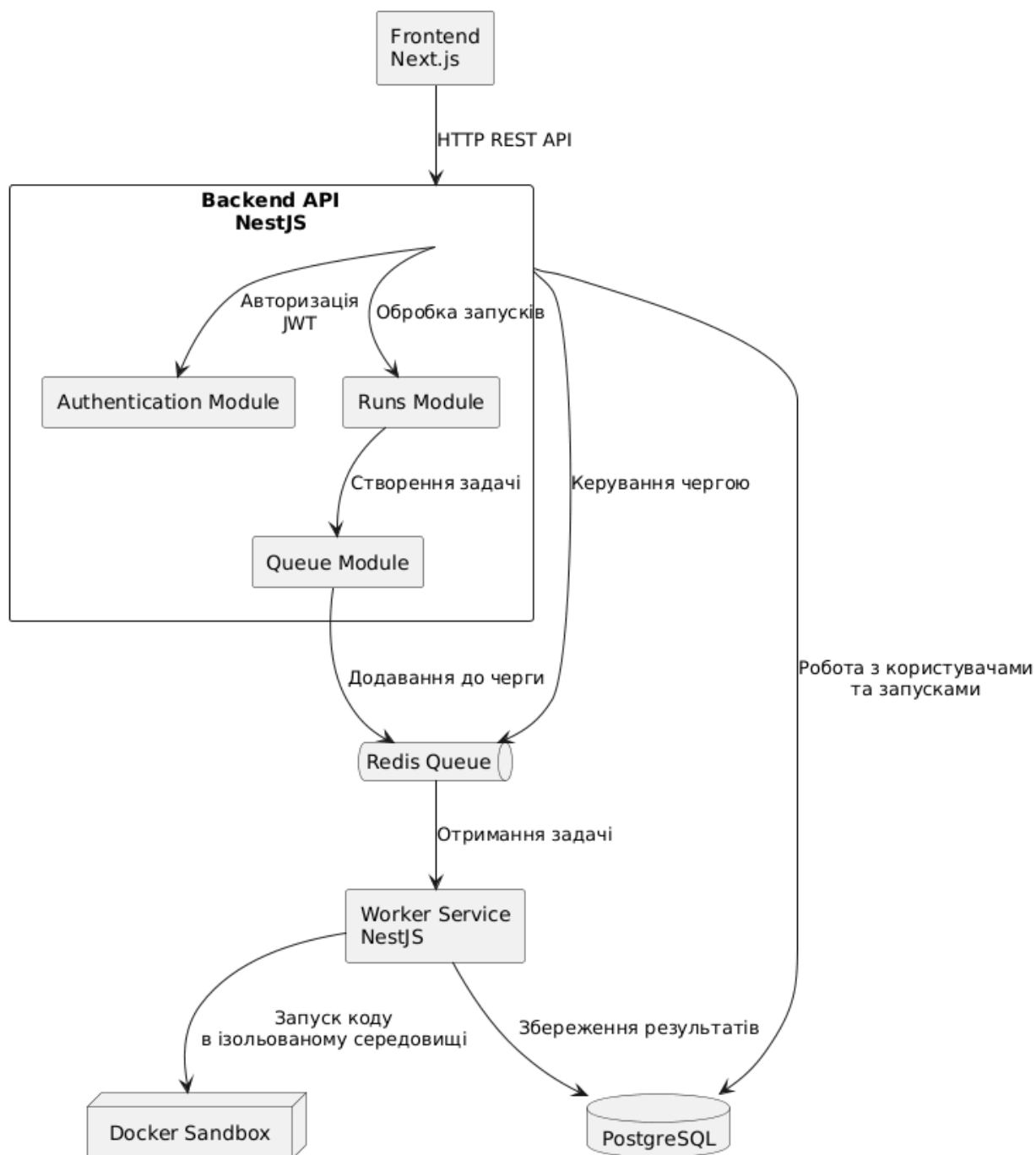


Рис. 2.16. Загальна компонентна структура програмного комплексу

На рис. 2.16 наведено загальну компонентну структуру системи. Кожен компонент виконує окремі функції та взаємодіє з іншими частинами програмного комплексу через визначені інтерфейси.

Центральним елементом архітектури є Backend API. Саме через нього проходять усі запити користувачів до системи.

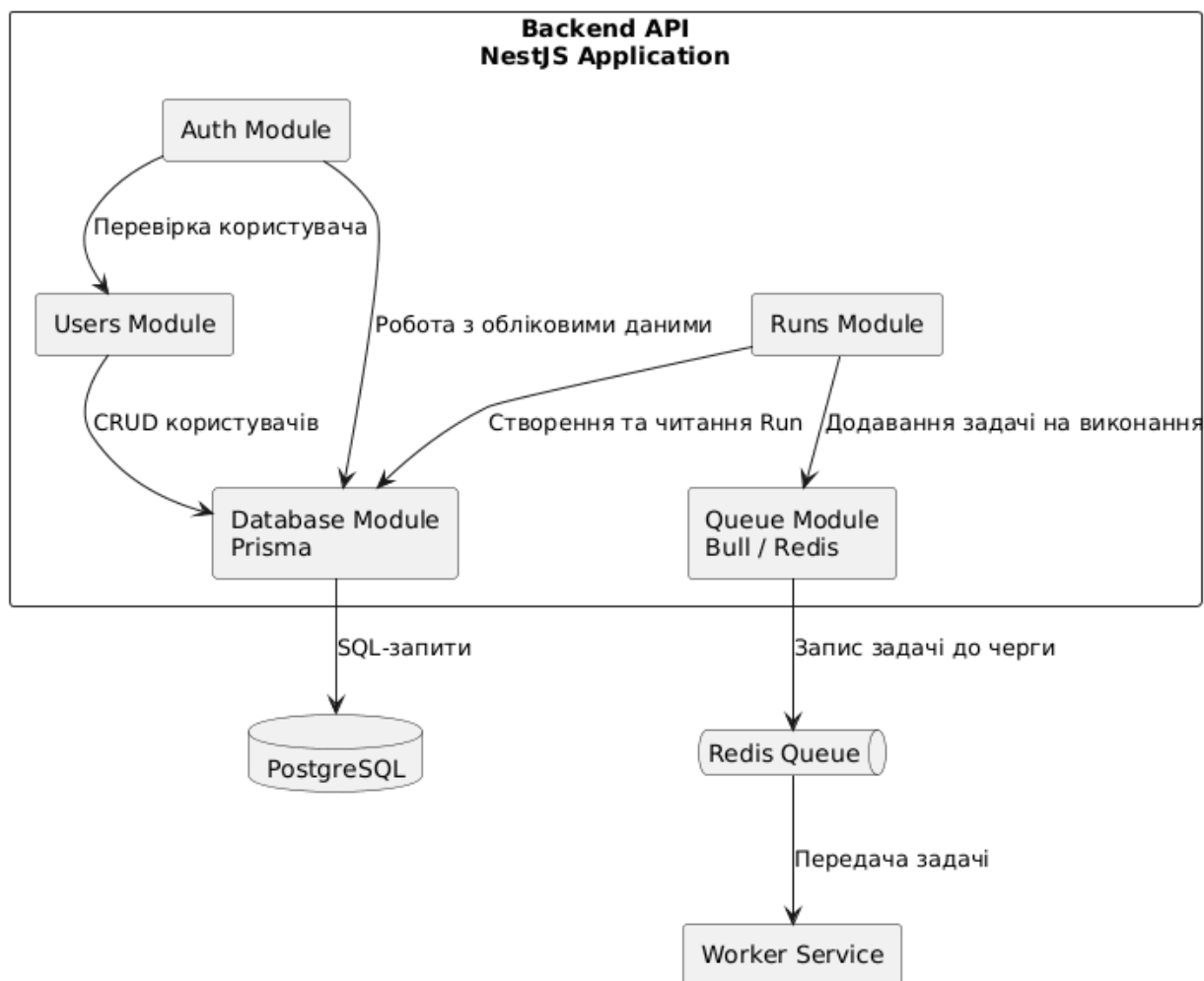


Рис. 2.17. Внутрішня структура Backend API

Backend API виконує функції перевірки вхідних даних, авторизації користувачів, створення запусків програм та передачі задач до черги обробки.

Для реалізації механізму автентифікації використовується окремий модуль авторизації. [30; 31]

Таблиця 2.9

Функції модуля авторизації

Функція	Призначення
Реєстрація	Створення нового користувача
Авторизація	Вхід до системи
Перевірка JWT	Підтвердження доступу
Отримання профілю	Дані поточного користувача

Модуль авторизації забезпечує контроль доступу до функціональних можливостей системи та захист даних користувачів.

Для роботи із задачами використовується модуль Runs.

Таблиця 2.10

Функції модуля Runs

Функція	Призначення
Створення запуску	Формування нової задачі
Отримання результату	Перегляд результатів
Історія запусків	Отримання списку запусків
Зміна статусу	Оновлення стану задачі

Після створення запуску задача передається до модуля черги повідомлень.

Для асинхронної взаємодії між компонентами використовується Redis та Bull.



Рис. 2.18. Схема роботи черги повідомлень

Черга повідомлень дозволяє відокремити процес виконання програм від основного серверного застосунку. Завдяки цьому користувач не очікує завершення роботи програми під час виконання HTTP-запиту.

Окремим компонентом системи є Worker-сервіс.

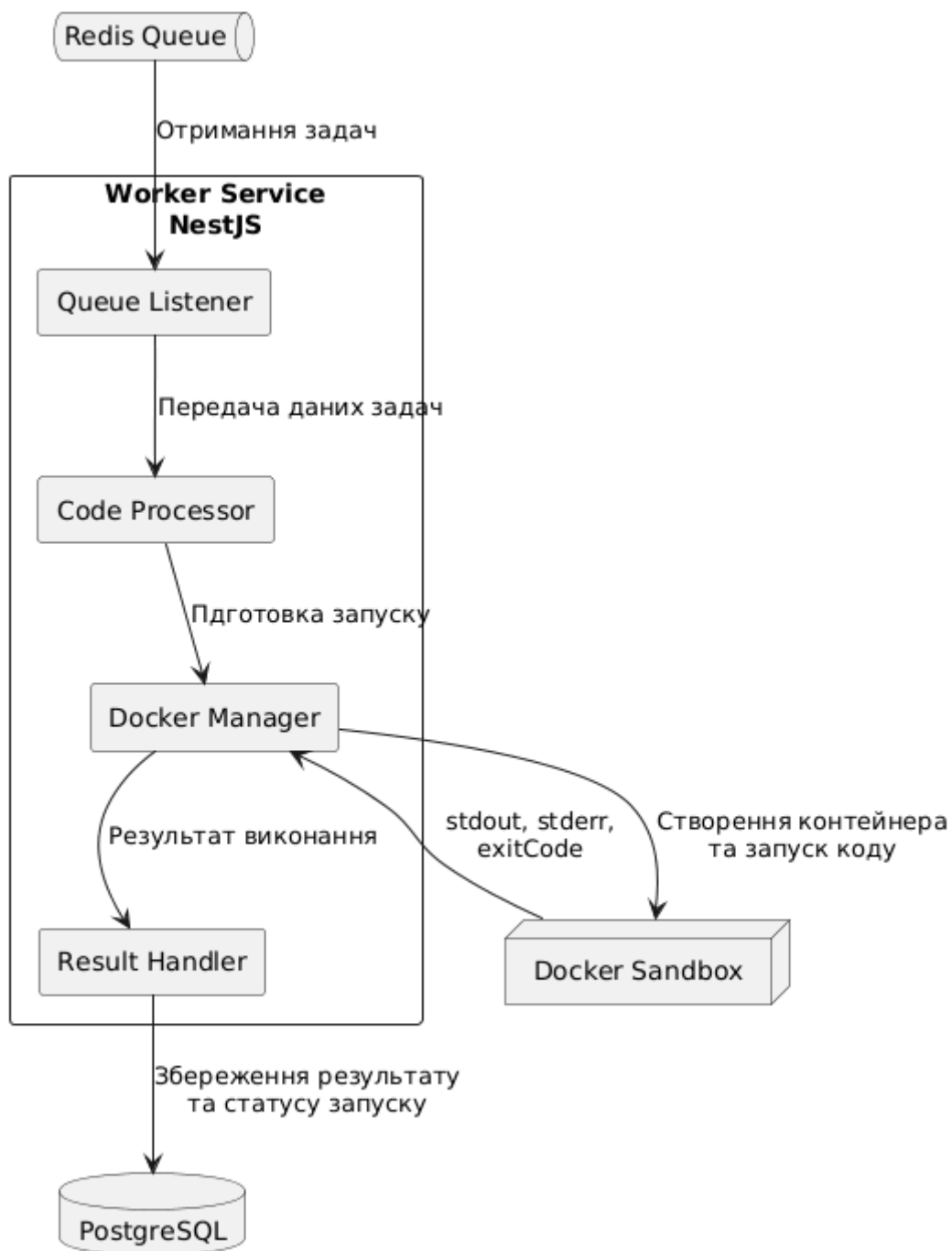


Рис. 2.19. Внутрішня структура Worker-сервісу

Worker виконує обробку задач, запуск контейнерів та збереження результатів виконання програм до бази даних.

Для виконання програм використовується Docker Sandbox.

Функції Docker Sandbox

Функція	Призначення
Ізоляція процесів	Захист сервера
Контроль ресурсів	Обмеження навантаження
Тимчасове середовище	Виконання програми
Автоматичне видалення	Очищення ресурсів

Контейнерне середовище забезпечує ізоляцію виконання користувацького програмного коду від інших компонентів системи.

Для зберігання інформації використовується PostgreSQL.

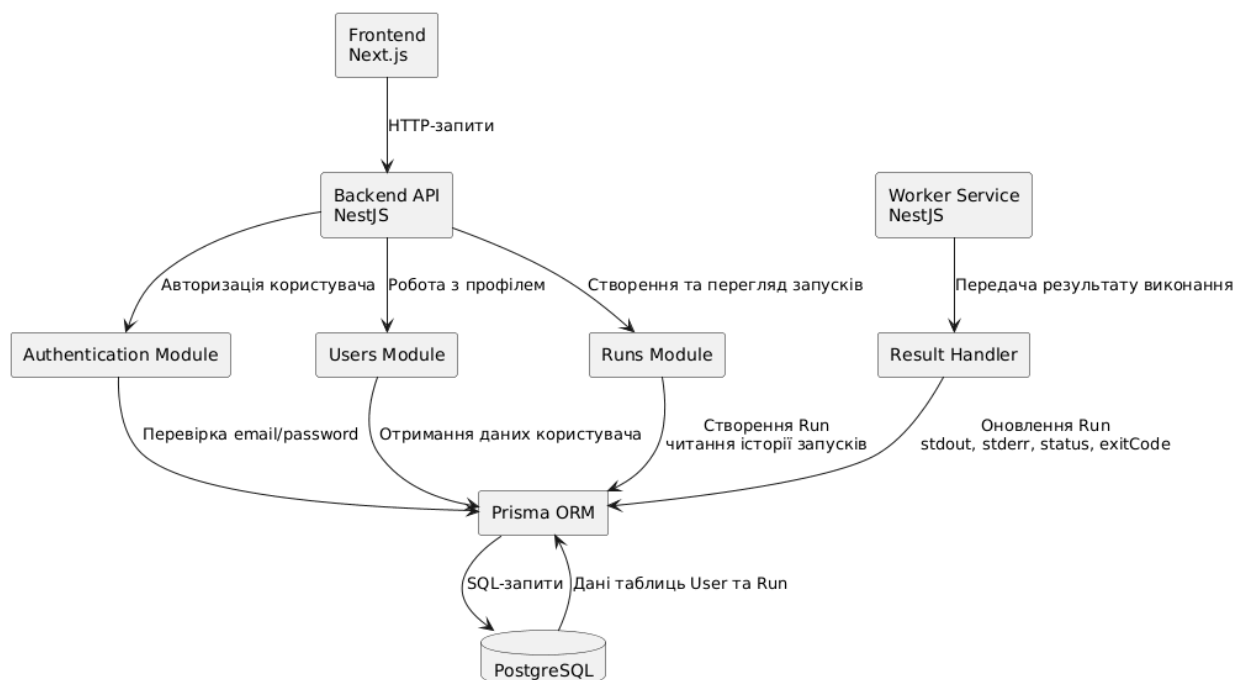


Рис. 2.20. Взаємодія програмних компонентів із базою даних

На рис. 2.20 наведено взаємодію програмних компонентів із базою даних. Серверна частина створює записи про запуск програм, а Worker оновлює результати після завершення виконання.

Таким чином, компонентна структура програмного комплексу забезпечує чіткий розподіл відповідальності між окремими сервісами,

спрощує супровід програмного забезпечення та створює основу для подальшого масштабування системи.

Висновки до другого розділу

У другому розділі було виконано проєктування мікросервісу для перевірки та виконання програмного коду в ізольованому середовищі. На основі результатів аналізу предметної області визначено основні сценарії використання системи та сформовано вимоги до її функціональності.

Було побудовано UML-діаграми варіантів використання, компонентів та послідовності взаємодії, що дозволило визначити структуру програмного комплексу та взаємозв'язки між його основними складовими. Запропонована архітектура базується на принципах мікросервісного підходу та передбачає використання окремих компонентів для обробки запитів користувачів, виконання програмного коду та зберігання результатів.

У процесі проєктування розроблено структуру бази даних, яка забезпечує зберігання інформації про користувачів, запуски програм та результати їх виконання. Запропонована модель даних підтримує цілісність інформації та може бути розширена в майбутніх версіях програмного комплексу.

Також було розроблено алгоритми роботи системи, що охоплюють процес створення задач на виконання, обробку черги повідомлень, запуск програм у Docker-контейнерах та контроль часу виконання. Реалізовані алгоритми забезпечують безпечне виконання користувацького програмного коду та захист серверної інфраструктури від надмірного споживання ресурсів.

У результаті виконання розділу сформовано повну архітектурну основу програмного комплексу та реалізовано його основні програмні компоненти. Отримані результати дозволяють перейти до опису інтерфейсу користувача, порядку встановлення системи та тестування реалізованого програмного забезпечення.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МІКРОСЕРВІСОМ ДЛЯ ВИКОНАННЯ ПРОГРАМНОГО КОДУ В ІЗОЛЬОВАНОМУ СЕРЕДОВИЩІ

3.1 Порядок встановлення та налаштування системи

Для забезпечення працездатності мікросервісу перевірки та виконання програмного коду необхідно виконати розгортання всіх компонентів програмного комплексу. До складу системи входять клієнтська частина, серверна частина, сервіс обробки задач, база даних PostgreSQL, Redis та Docker-середовище для виконання програмного коду.

Розроблений програмний комплекс орієнтований на розгортання в операційних системах сімейства Linux. Для роботи системи необхідно попередньо встановити Docker Engine та Docker Compose, які використовуються для запуску основних компонентів програмного комплексу.

Під час розгортання кожен компонент запускається в окремому контейнері та виконує власний набір функцій. Такий підхід дозволяє ізолювати сервіси один від одного, спростити процес супроводу програмного комплексу та забезпечити можливість незалежного оновлення окремих компонентів системи. Клієнтська частина забезпечує взаємодію користувача із системою через вебінтерфейс та відповідає за відображення результатів виконання програмного коду. Серверна частина обробляє запити користувачів, здійснює взаємодію з базою даних та координує роботу інших сервісів програмного комплексу.

Redis використовується для організації черги повідомлень та передачі задач між компонентами системи. Завдяки використанню черги забезпечується асинхронна обробка запитів, що дозволяє зменшити навантаження на серверну частину та підвищити загальну продуктивність системи. Worker-сервіс отримує задачі з черги та виконує запуск програмного

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

коду в ізолюваному середовищі Docker із застосуванням встановлених обмежень щодо використання ресурсів.

Для зберігання інформації про користувачів, історію запусків та результати виконання програм використовується база даних PostgreSQL. Взаємодія між усіма компонентами системи здійснюється через внутрішні мережеві механізми Docker, що забезпечує надійний обмін даними та спрощує процес розгортання програмного комплексу на сервері.

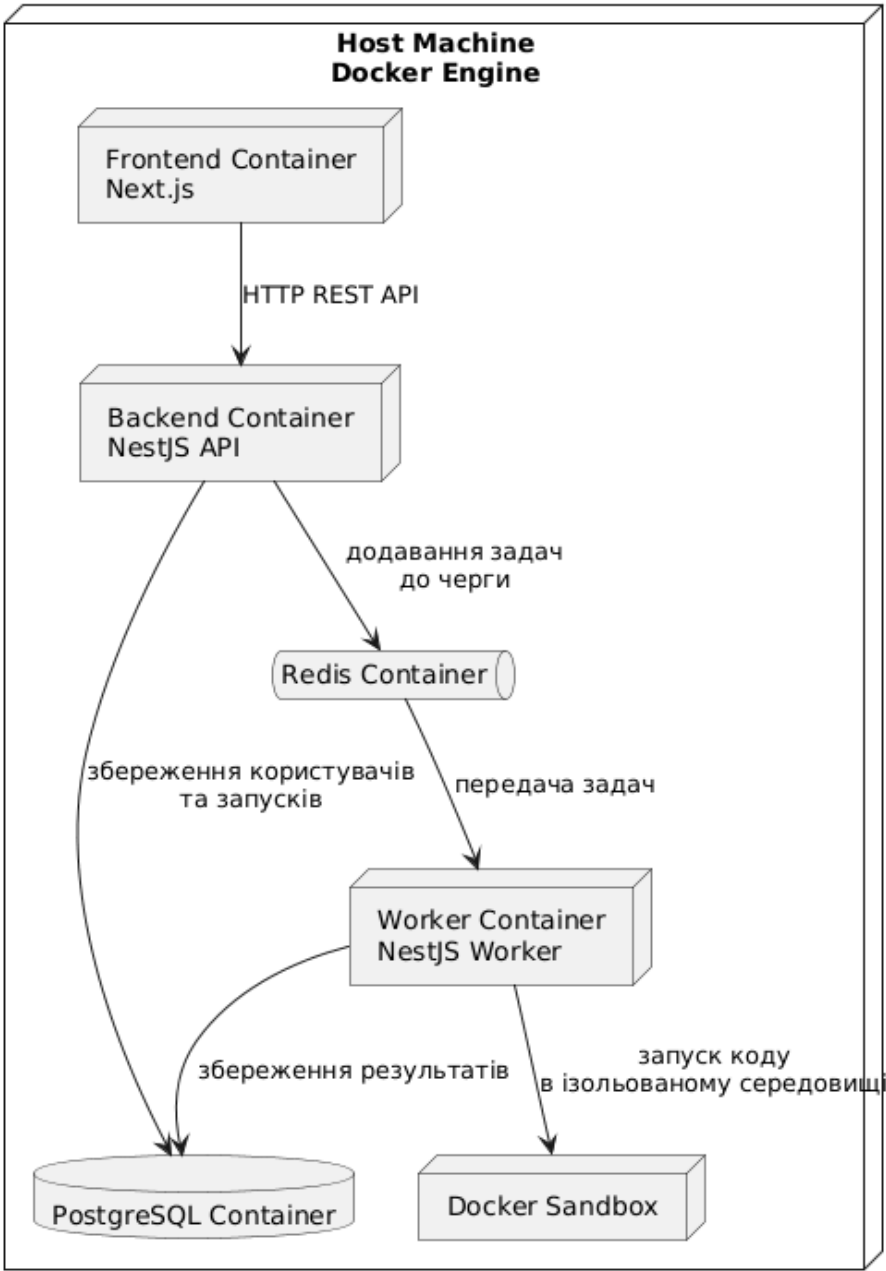


Рис. 3.1. Загальна схема розгортання системи

На рис. 3.1 представлено загальну схему розгортання програмного комплексу та взаємодію його основних компонентів. Усі сервіси функціонують у контейнеризованому середовищі Docker, що забезпечує ізоляцію компонентів, спрощує процес розгортання та підвищує переносимість системи між різними середовищами виконання.

Клієнтська частина реалізована у вигляді окремого контейнера на базі Next.js та забезпечує взаємодію користувача із системою через вебінтерфейс. Користувач має можливість вводити програмний код, створювати нові запуски та переглядати результати виконання програм. Передача даних між клієнтською та серверною частинами здійснюється за допомогою HTTP-запитів до REST API.

Серверна частина реалізована на базі фреймворку NestJS та виконує функції координації роботи всіх компонентів системи. Backend API обробляє запити користувачів, виконує перевірку даних, створює записи запусків у базі даних та формує задачі на виконання програмного коду.

Для організації асинхронної обробки використовується Redis, який виконує роль брокера повідомлень та забезпечує передачу задач між Backend API та Worker-сервісом. Такий підхід дозволяє розділити процес приймання запитів і безпосереднього виконання програмного коду, що позитивно впливає на продуктивність системи.

Worker-сервіс отримує задачі з черги та виконує запуск програм у спеціально створених Docker-контейнерах. Після завершення виконання результати роботи програми зберігаються в базі даних PostgreSQL та стають доступними користувачу через вебінтерфейс. Використання окремого сервісу виконання дозволяє масштабувати систему шляхом збільшення кількості Worker-процесів без зміни логіки роботи інших компонентів.

База даних PostgreSQL використовується для зберігання інформації про користувачів, історію запусків програм та результати виконання. Така архітектура забезпечує централізоване зберігання даних, підтримує цілісність інформації та спрощує супровід програмного комплексу.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Після запуску Docker Compose автоматично створюються контейнери PostgreSQL, Redis, Backend та Worker. Після успішного запуску система стає доступною для використання через веббраузер.

Для підключення до бази даних використовується файл конфігурації середовища.

Таблиця 3.1

Основні параметри конфігурації системи

Параметр	Призначення
DATABASE_URL	Підключення до PostgreSQL
REDIS_HOST	Адреса Redis
REDIS_PORT	Порт Redis
JWT_SECRET	Ключ авторизації
API_PORT	Порт Backend API

Зберігання конфігураційних параметрів у змінних середовища дозволяє підвищити безпеку системи та спрощує перенесення програмного комплексу між різними середовищами розгортання.

Після завершення встановлення користувач може відкрити вебінтерфейс системи та перейти до роботи з програмним комплексом.

Таким чином, процес розгортання системи є достатньо простим та виконується за допомогою контейнеризації Docker. Використання Docker Compose дозволяє автоматизувати запуск усіх необхідних компонентів програмного комплексу, мінімізувати витрати часу на налаштування та спростити процес адміністрування системи. Завдяки використанню контейнерів забезпечується однакове середовище виконання для всіх сервісів незалежно від конфігурації сервера. Такий підхід підвищує надійність роботи програмного комплексу, спрощує його перенесення між різними середовищами та дозволяє швидко виконувати оновлення окремих компонентів системи. [32; 33; 34; 35; 36; 37]

3.2 Структура інтерфейсу вебзастосунку

Інтерфейс користувача є одним із ключових компонентів програмного комплексу, оскільки саме через нього здійснюється взаємодія користувача з усіма функціями системи. Під час розробки особлива увага приділялась простоті використання, зрозумілій навігації та швидкому доступу до основних функціональних можливостей сервісу. [38; 39]

Клієнтська частина реалізована за допомогою фреймворку Next.js та мови програмування TypeScript. Для введення програмного коду використовується редактор Monaco Editor, який забезпечує підсвічування синтаксису та зручну роботу з програмним кодом.

Після відкриття вебзастосунку користувач потрапляє на головну сторінку системи.

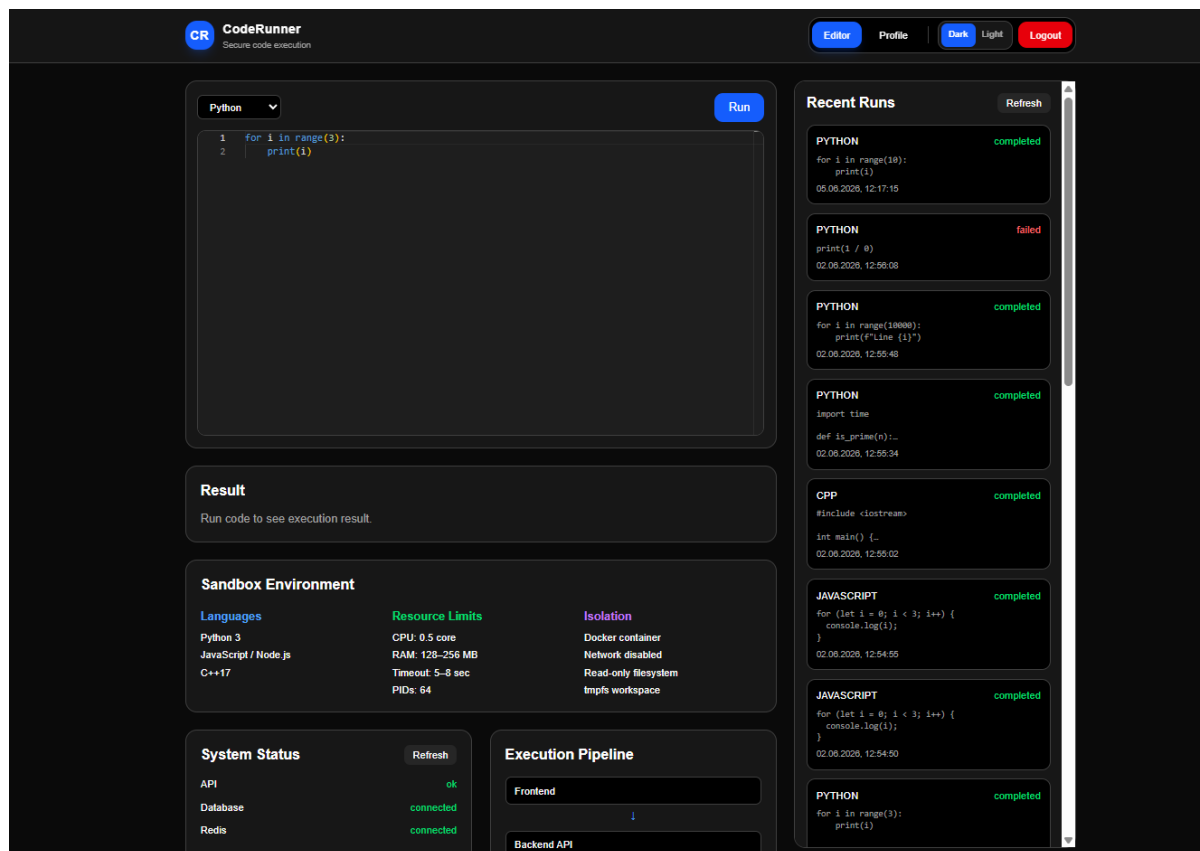


Рис. 3.2. Головна сторінка програмного комплексу

На рис. 3.2 представлено головну сторінку системи. Вона містить основну інформацію про сервіс та забезпечує перехід до функцій авторизації та роботи з програмним кодом.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

Для доступу до функціоналу системи користувач повинен пройти процедуру авторизації.

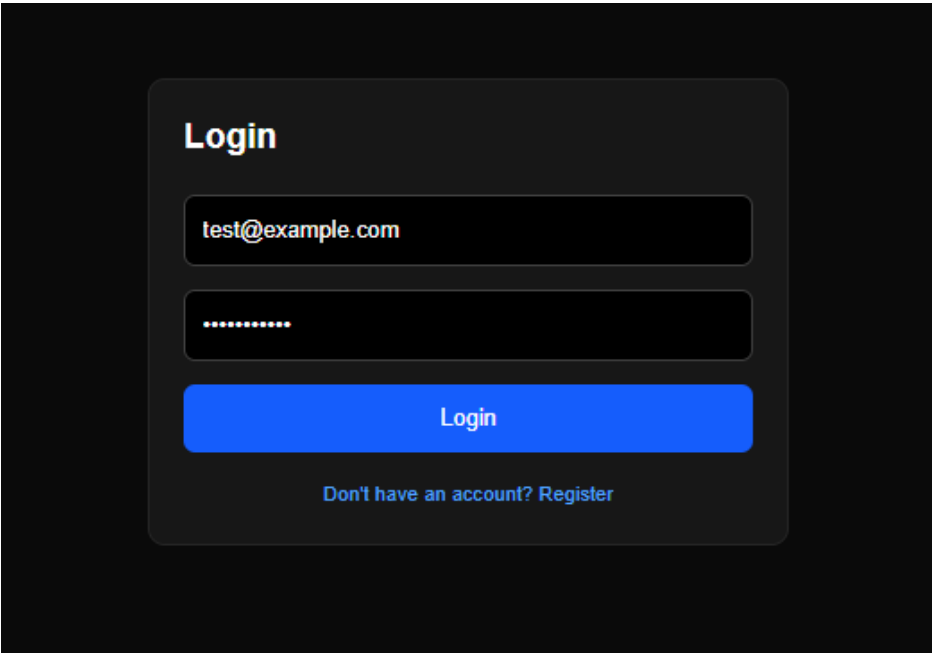


Рис. 3.3. Сторінка авторизації користувача

Сторінка авторизації містить поля для введення адреси електронної пошти та пароля. Після успішної перевірки облікових даних користувач отримує доступ до функціоналу системи.

Для нових користувачів передбачено механізм реєстрації.

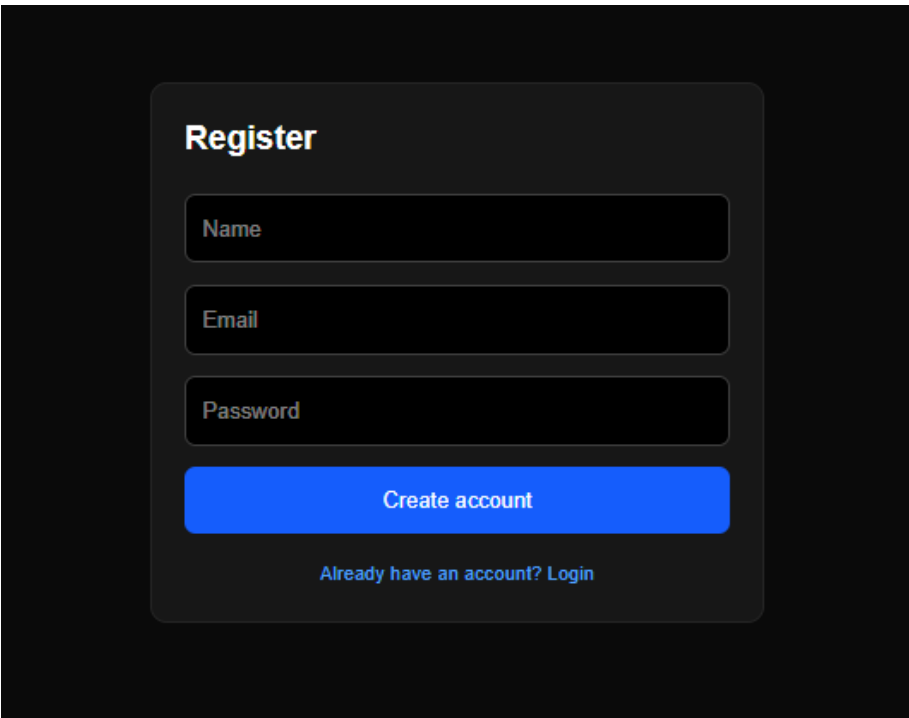


Рис. 3.4. Сторінка реєстрації користувача

На рис. 3.4 наведено форму створення нового облікового запису. Після заповнення необхідних полів інформація передається до серверної частини та зберігається в базі даних.

Основною сторінкою програмного комплексу є сторінка виконання програмного коду. У центральній частині сторінки розташований редактор програмного коду, а також елементи керування запуском програм.

Для введення програм використовується редактор Monaco Editor.

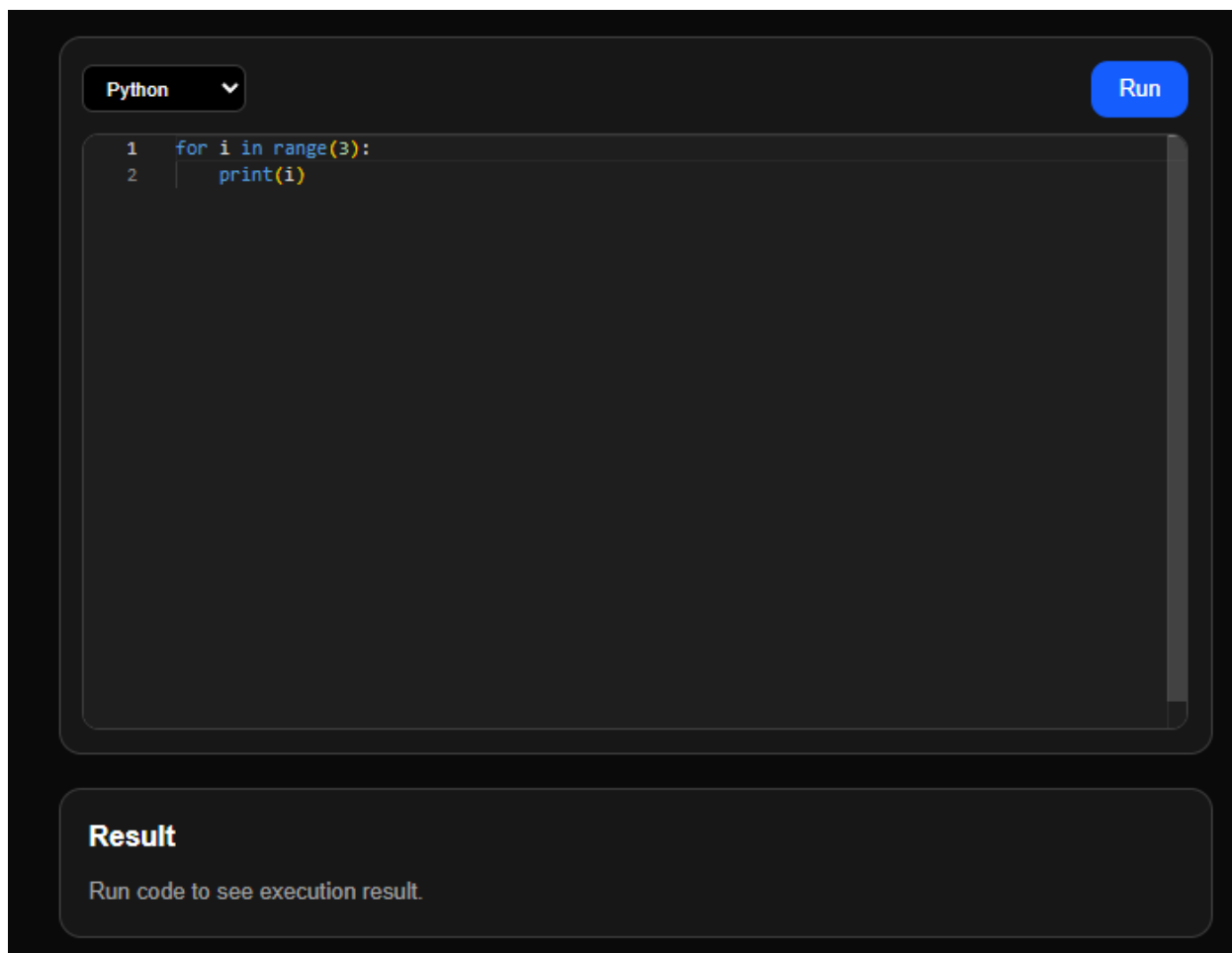


Рис. 3.5. Редактор програмного коду Monaco Editor

Редактор підтримує підсвічування синтаксису, автоматичне форматування тексту та інші можливості, які спрощують написання програмного коду.

Можливості програмного комплексу не обмежуються лише виконанням програмного коду. Для кожного користувача передбачено окрему сторінку профілю, на якій відображається статистика використання системи.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

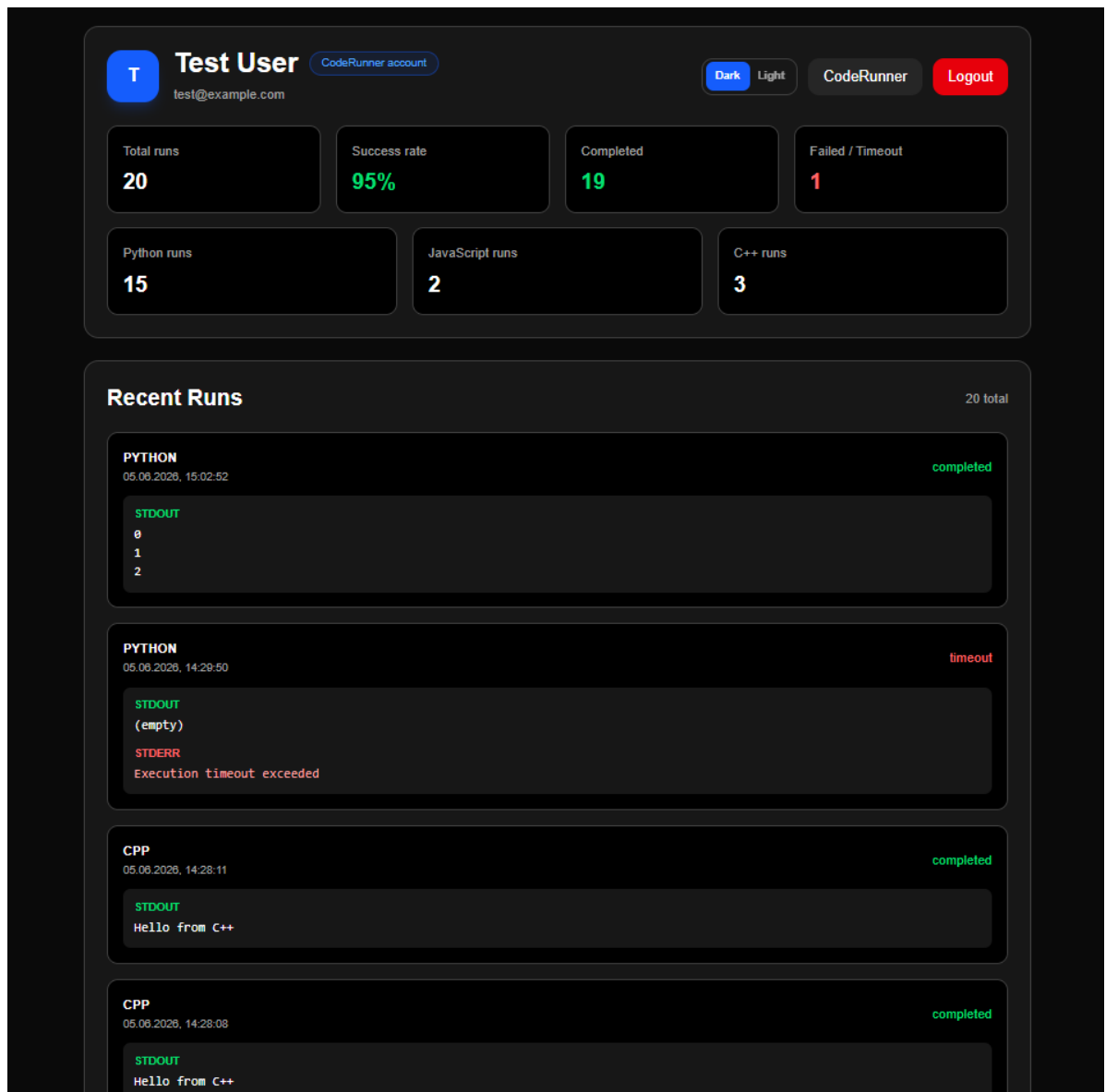


Рис. 3.6. Сторінка профілю користувача

На рис. 3.6 представлено сторінку профілю користувача. Вона містить основну інформацію про обліковий запис, загальну кількість виконаних запусків, відсоток успішних виконань, а також статистику використання різних мов програмування. Додатково користувач може переглядати історію своїх останніх запусків без необхідності переходу до окремого журналу виконання.

Для більш детального аналізу результатів роботи програм у системі реалізовано окрему сторінку перегляду інформації про конкретне виконання.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

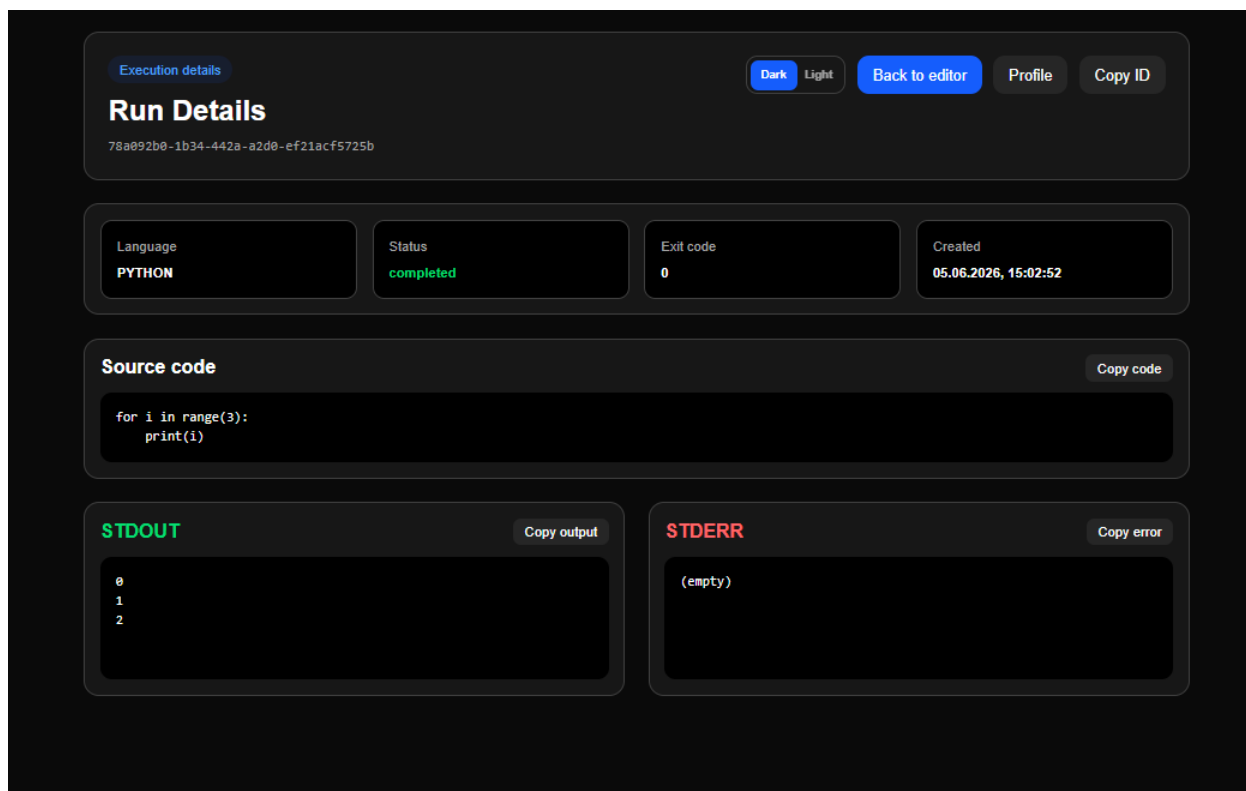


Рис. 3.7. Сторінка детальної інформації про виконання програми

На рис. 3.7 наведено сторінку детальної інформації про виконання програми. На ній відображаються ідентифікатор запуску, мова програмування, статус виконання, код завершення процесу та дата створення задачі. Також користувач має можливість переглядати вихідний програмний код, стандартний потік виведення (STDOUT) та потік повідомлень про помилки (STDERR), що значно спрощує аналіз результатів виконання програм.

3.3 Тестування роботи програмного комплексу

Після завершення реалізації програмного комплексу було проведено тестування основних функціональних можливостей системи. Метою тестування є перевірка коректності роботи всіх компонентів програмного комплексу, оцінка стабільності функціонування системи та підтвердження відповідності реалізованого функціоналу поставленим вимогам. [40]

Тестування виконувалося на розгорнутому середовищі, що включало клієнтську частину, серверну частину, Redis, PostgreSQL, Worker-сервіс та Docker-середовище виконання програмного коду.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

Основна увага приділялась перевірці процесу виконання програмного коду, роботі механізму ізоляції, контролю часу виконання та обробці помилок.

Для перевірки коректності роботи системи було сформовано набір тестових сценаріїв.

Таблиця 3.2

Тестування основного функціоналу системи

№	Тестовий сценарій	Очікуваний результат	Результат
1	Авторизація користувача	Успішний вхід до системи	Успішно
2	Створення запуску програми	Створення задачі	Успішно
3	Виконання коректної програми	Отримання результату	Успішно
4	Отримання історії запусків	Відображення списку запусків	Успішно
5	Збереження результатів до БД	Дані збережені	Успішно

Результати тестування показали коректну роботу основних функцій програмного комплексу.

Першим тестом було виконання простої програми мовою Python.

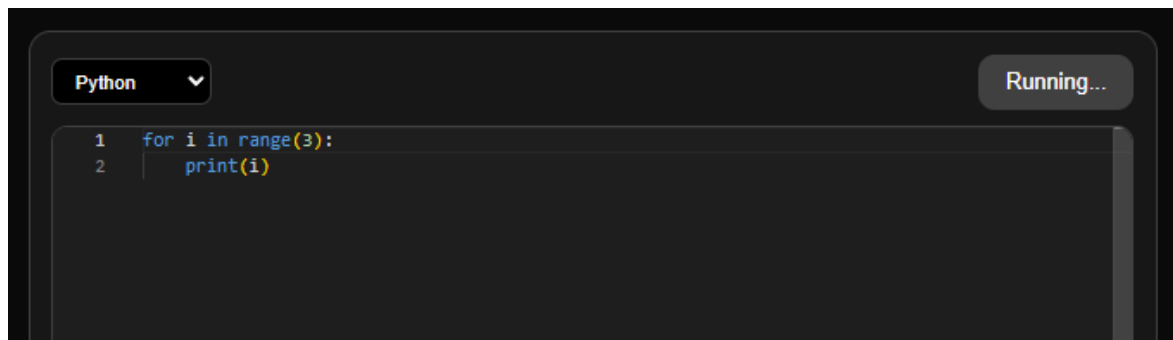


Рис. 3.8. Тестування програми

На рис. 3.8 представлено процес тестування програмного коду через вебінтерфейс розробленого програмного комплексу. Для перевірки працездатності системи було використано просту програму мовою Python, яка виводить послідовність чисел у стандартний потік виведення. Після натискання кнопки запуску код передається на серверну частину системи, де створюється задача на виконання та передається до черги обробки. Даний тест дозволяє перевірити коректність взаємодії між клієнтською частиною, сервером та сервісом виконання задач.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

Result

Run ID: 5eb2ff8b-163f-4780-88b3-f995662cc573 | Status: completed

STDOUT

0

1

2

STDERR

(empty)

Exit code: 0

Рис. 3.9. Результат виконання програми

На рис. 3.9 представлено результат виконання найпростішої програми. Система успішно передала код до середовища виконання, виконала програму та повернула результат користувачу.

Наступним етапом було тестування роботи з математичними обчисленнями.

Source code

Copy code

```
import math

radius = 5

area = math.pi * radius ** 2
circumference = 2 * math.pi * radius

print(f"Radius: {radius}")
print(f"Area: {area:.2f}")
print(f"Circumference: {circumference:.2f}")
```

STDOUT

Copy output

Radius: 5

Area: 78.54

Circumference: 31.42

STDERR

Copy error

(empty)

Рис. 3.10. Тестування арифметичних операцій

Отримані результати повністю співпали з очікуваними значеннями, що підтверджує коректність передачі та виконання програмного коду.

Для перевірки механізму обробки синтаксичних помилок було використано програму з навмисною помилкою.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

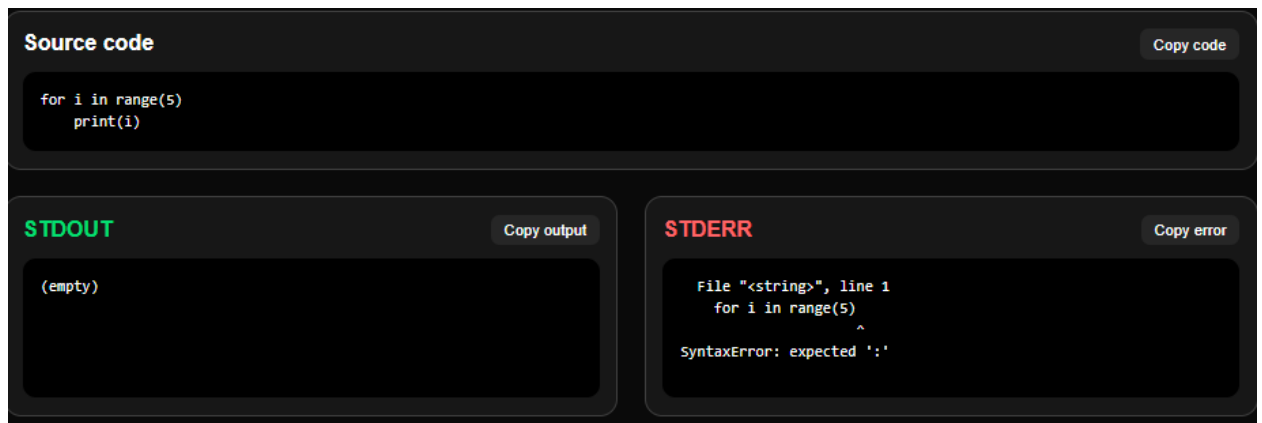


Рис. 3.11. Тестування синтаксичної помилки

На рис. 3.11 показано результат тестування механізму обробки синтаксичних помилок. Для перевірки було навмисно використано програмний код із порушенням синтаксису мови Python. Після виконання задачі система коректно виявила помилку та відобразила детальне повідомлення, отримане від інтерпретатора. Це дозволяє користувачу швидко визначити причину помилки та внести необхідні виправлення до програмного коду.

Наступним тестом було моделювання помилки виконання.

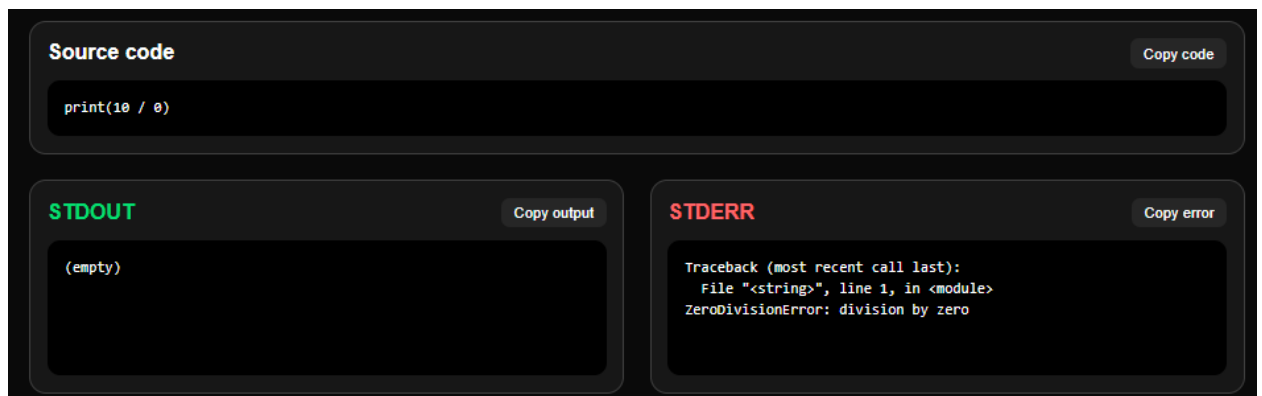


Рис. 3.12. Тестування Runtime Error

На рис. 3.12 представлено приклад тестування помилки виконання програми (Runtime Error). Для перевірки працездатності системи було використано програмний код, який викликає ділення на нуль під час виконання. У результаті система успішно перехопила виняткову ситуацію та відобразила відповідне повідомлення про помилку без аварійного завершення роботи сервісу.

Особливу увагу було приділено перевірці механізму обмеження часу виконання.

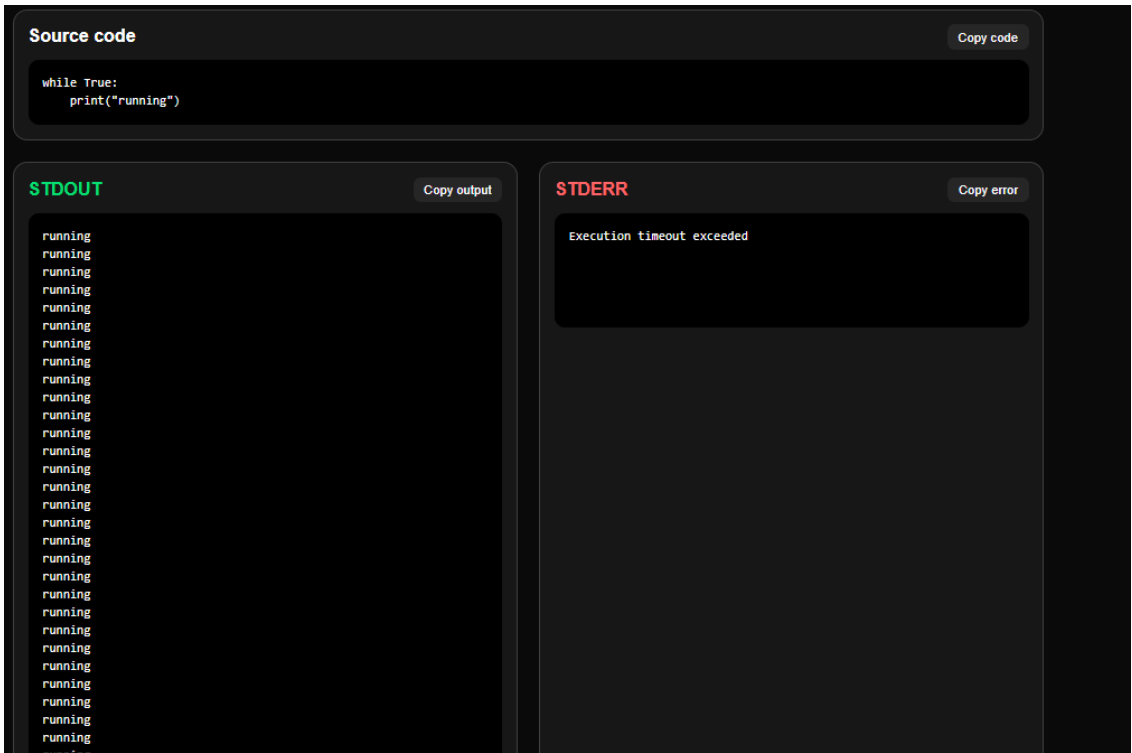


Рис. 3.13. Тестування перевищення часу виконання

Під час виконання даної програми система виявила перевищення встановленого ліміту часу та автоматично завершила контейнер. Користувачу було повернуто повідомлення про перевищення часу виконання.

Для перевірки роботи історії запусків було створено декілька послідовних задач.

Таблиця 3.3

Результати тестування системи

Категорія тестування	Кількість тестів	Успішно
Авторизація	3	3
Робота API	5	5
Виконання коду	8	8
Обробка помилок	2	2
Робота Docker	1	1
Робота БД	2	2

У ході тестування критичних помилок, які унеможливають використання програмного комплексу, виявлено не було. Усі основні функціональні можливості системи працюють відповідно до поставлених вимог.

Таким чином, результати тестування підтверджують працездатність розробленого мікросервісу та можливість його використання для безпечного виконання програмного коду в ізольованому середовищі.

Висновки до третього розділу

У третьому розділі було розглянуто особливості встановлення, налаштування та використання розробленого програмного комплексу. Наведено опис процесу розгортання системи із застосуванням контейнеризації Docker, що забезпечує спрощення налаштування середовища та підвищує зручність супроводу програмного забезпечення.

Виконано детальний опис інтерфейсу користувача, який забезпечує взаємодію із системою, введення програмного коду, запуск задач на виконання та перегляд отриманих результатів. Реалізований інтерфейс дозволяє користувачам ефективно працювати із сервісом без необхідності додаткових знань щодо внутрішньої архітектури програмного комплексу.

У процесі тестування було перевірено всі основні функціональні можливості системи, включаючи створення запусків програмного коду, роботу черги задач, виконання програм у Docker-контейнерах, збереження результатів у базі даних та механізми обробки помилок. Отримані результати підтвердили коректність роботи програмного комплексу та відповідність реалізованого функціоналу поставленим вимогам.

Проведене тестування продемонструвало стабільну роботу системи при виконанні коректних програм, обробці помилок синтаксису та виконання, а також при виникненні ситуацій перевищення допустимого часу виконання. Це підтверджує ефективність використання контейнеризації та асинхронної обробки задач у розробленому програмному комплексі.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено мікросервіс для перевірки та виконання програмного коду в ізольованому середовищі. У ході дослідження проаналізовано особливості побудови систем виконання програмного коду, розглянуто сучасні підходи до контейнеризації та ізоляції процесів, а також досліджено існуючі програмні рішення в даній предметній області.

На основі проведеного аналізу було обрано мікросервісну архітектуру, яка дозволяє розділити функціональність системи між незалежними компонентами та забезпечує можливість подальшого масштабування програмного комплексу. Запропонована архітектура включає клієнтську частину, серверний API, систему чергування задач, сервіс виконання програмного коду та базу даних для зберігання інформації про користувачів і результати запусків.

Під час виконання роботи спроектовано структуру бази даних, реалізовано REST API для взаємодії між компонентами системи, створено механізм автентифікації користувачів на основі JWT та реалізовано систему збереження історії запусків програм. Для обробки задач використано Redis та Bull, що забезпечує асинхронне виконання запитів та зменшує навантаження на серверну частину застосунку.

Особливу увагу приділено питанням безпеки. Виконання користувацького коду здійснюється в ізольованих Docker-контейнерах із обмеженням використання процесора, оперативної пам'яті та мережевих ресурсів. Додатково реалізовано контроль часу виконання програм, що дозволяє запобігати зависанню системи внаслідок нескінченних циклів або надмірного споживання ресурсів.

У межах роботи розроблено вебзастосунок, який підтримує виконання програм мовами Python, JavaScript та C++, відображення результатів виконання, перегляд історії запусків і роботу з обліковими записами

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

користувачів. Також реалізовано автоматизовану перевірку проєкту за допомогою GitHub Actions та виконано тестування основних функціональних можливостей системи.

Результати проведеного тестування підтвердили коректність роботи всіх ключових компонентів програмного комплексу. Система успішно приймає програмний код користувача, виконує його в ізольованому середовищі, зберігає результати в базі даних та забезпечує їх відображення через вебінтерфейс.

Отримані результати демонструють ефективність використання сучасних підходів до розробки програмного забезпечення, зокрема мікросервісної архітектури, контейнеризації та асинхронної обробки задач. Розроблений програмний комплекс забезпечує розділення відповідальності між компонентами системи, що спрощує подальшу підтримку, тестування та розширення функціональних можливостей. Використання Docker-контейнерів та черги повідомлень Redis дозволяє підвищити надійність і безпечність виконання користувацького програмного коду, а також створює основу для подальшого масштабування системи в умовах зростання навантаження.

Практична цінність роботи полягає у створенні програмного комплексу, який може бути використаний як основа для освітніх платформ, онлайн-компіляторів, систем автоматизованої перевірки програмних рішень та інших інформаційних систем, що потребують безпечного виконання програмного коду.

Подальший розвиток проєкту може бути пов'язаний із підтримкою додаткових мов програмування, впровадженням автоматичної перевірки тестових наборів, розширенням можливостей масштабування системи, інтеграцією засобів моніторингу та логування, а також удосконаленням механізмів ізоляції середовища виконання.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 618 p.
2. Richardson C. Microservices Patterns. Shelter Island : Manning Publications, 2018. 520 p.
3. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p.
4. Docker Inc. Docker Documentation. URL: <https://docs.docker.com> (дата звернення: 03.06.2026).
5. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal. 2014. No. 239. P. 2–7.
6. Turnbull J. The Docker Book: Containerization is the New Virtualization. James Turnbull, 2023. 394 p.
7. NestJS Documentation. URL: <https://docs.nestjs.com> (дата звернення: 03.06.2026).
8. Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 03.06.2026).
9. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs> (дата звернення: 03.06.2026).
10. Node.js Documentation. URL: <https://nodejs.org/docs/latest/api> (дата звернення: 03.06.2026).
11. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs> (дата звернення: 03.06.2026).
12. Redis Documentation. URL: <https://redis.io/docs> (дата звернення: 03.06.2026).
13. Bull Queue Documentation. URL: <https://docs.bullmq.io> (дата звернення: 03.06.2026).
14. Monaco Editor Documentation. URL: <https://microsoft.github.io/monaco-editor> (дата звернення: 03.06.2026).

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Judge0 Documentation. URL: <https://judge0.com> (дата звернення: 03.06.2026).
16. OWASP Foundation. OWASP Top 10 Web Application Security Risks. URL: <https://owasp.org> (дата звернення: 03.06.2026).
17. NIST. Application Container Security Guide. Gaithersburg : National Institute of Standards and Technology, 2017.
18. Red Hat. Container Security Guide. URL: <https://www.redhat.com> (дата звернення: 03.06.2026).
19. Kubernetes Documentation. URL: <https://kubernetes.io/docs> (дата звернення: 03.06.2026).
20. Linux Foundation. Introduction to Containers and Kubernetes. URL: <https://training.linuxfoundation.org> (дата звернення: 03.06.2026).
21. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2018. 432 p.
22. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
23. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.
24. Pressman R., Maxim B. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2019. 880 p.
25. UML Resource Page. URL: <https://www.uml.org> (дата звернення: 03.06.2026).
26. OMG Unified Modeling Language Specification. URL: <https://www.omg.org/spec/UML> (дата звернення: 03.06.2026).
27. PostgreSQL Performance Documentation. URL: <https://wiki.postgresql.org> (дата звернення: 03.06.2026).
28. Mozilla Developer Network. HTTP Overview. URL: <https://developer.mozilla.org> (дата звернення: 03.06.2026).
29. RFC 9110. HTTP Semantics. Internet Engineering Task Force, 2022.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

30. RFC 8259. The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force, 2017.
31. JWT Official Introduction. URL: <https://jwt.io/introduction> (дата звернення: 03.06.2026).
32. Open Container Initiative. Runtime Specification. URL: <https://opencontainers.org> (дата звернення: 03.06.2026).
33. Google Cloud. Best Practices for Container Security. URL: <https://cloud.google.com> (дата звернення: 03.06.2026).
34. Microsoft Learn. Secure Application Development. URL: <https://learn.microsoft.com> (дата звернення: 03.06.2026).
35. Amazon Web Services. Containerized Applications on AWS. URL: <https://aws.amazon.com> (дата звернення: 03.06.2026).
36. IBM Cloud Documentation. Containers and Kubernetes. URL: <https://cloud.ibm.com/docs> (дата звернення: 03.06.2026).
37. Oracle. Database Concepts Guide. URL: <https://docs.oracle.com> (дата звернення: 03.06.2026).
38. Google Developers. REST API Design Guide. URL: <https://developers.google.com> (дата звернення: 03.06.2026).
39. W3C. Web Architecture Standards. URL: <https://www.w3.org> (дата звернення: 03.06.2026).
40. ISO/IEC 25010:2023. Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models.

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

ДОДАТОК А

Файл docker-compose.yml:

```
services:
  redis:
    image: redis:7-alpine
    container_name: coderunner-redis
    ports:
      - "6379:6379"

  postgres:
    image: postgres:16-alpine
    container_name: coderunner-postgres
    environment:
      POSTGRES_USER: coderunner
      POSTGRES_PASSWORD: coderunner
      POSTGRES_DB: coderunner
    ports:
      - "5433:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

ДОДАТОК Б

Файл schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
}

model User {
  id          String    @id @default(uuid())
  email       String    @unique
  passwordHash String
  name        String?
  createdAt   DateTime  @default(now())
}
```

```

    updatedAt    DateTime @updatedAt
    runs         Run[]
  }

  model Run {
    id           String    @id @default(uuid())
    language     String
    code         String
    stdout       String?
    stderr       String?
    status       String
    exitCode     Int?
    createdAt    DateTime @default(now())
    userId       String?
    user         User? @relation(fields: [userId], references: [id],
onDelete: Cascade)
  }

```

		Закревський С.			ІПЗ.КР.Б-121.26-ПЗ	Арк.
		Морозов А.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		76