Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Розподілена універсальна система для розпаралелювання завдань
Назва теми

КвРКІ 022017.22.02.36 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент III курсу, група КІ2с-22-2 _____ Валерія КОС
Підпис — Ініціали, прізвище

Керівник _____ Богдан САВЕНКО
Підпис, дата — Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата — Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис — Ініціали, прізвище

«12» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Кос Валерії Вікторовні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Розподілена універсальна система для розпаралелювання завдань

Керівник проекту (роботи) Савенко Б.О., д.ф.н.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області її структурних та функціональних особливостей

Проєктування програмно-технічного засобу

Програмно-апаратна реалізація та тестування програмно технічного засобу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Діаграма прецедентів

Діаграма активності

Діаграма компонент

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІС		

7. Дата видачі завдання «10» 01 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування розподіленої універсальної системи для розпаралелювання завдань	01.04.2025	виконано
5	Робота над розділом 3 – проєктування розподіленої універсальної системи для розпаралелювання завдань	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Валерія КОС
Підпис Ініціали, прізвище

Керівник роботи _____ Богдан САВЕНКО
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КРКІ 022017.02.36 ПЗ	Пояснювальна записка	70		
			Графічні матеріали			
2		КРКІ 022017.02.36 Е8	Діаграма прецендентів	1		
3		КРКІ 022017.02.36 Е8	Діаграма активності	1		
4		КРКІ 022017.02.36 Е8	Діаграма компонент	1		

КвРКІ 022017.22.02.36 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Кос		12.06.25
Перевір.		Савенко		12.06.25
Н. контр.		Кисіль		12.06.25
Затв.		Павлова		12.06.25

Відомість проекту

Літера	Аркуш	Аркушів
У	1	1

ХНУ, КІ2с-22-2

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Розподілена універсальна система для розпаралелювання завдань».

Автор роботи: Кос Валерія Вікторівна.

Керівник роботи: Савенко Богдан Олегович.

Пояснювальна записка: 70 с., 28 рис., 1 табл., 4 дод., 40 джерел.

Графічна частина: 3 креслення.

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, ДЕЦЕНТРАЛІЗАЦІЯ, САМООРГАНІЗАЦІЯ, WEB-ІНТЕРФЕЙС, ЗАДАЧІ.

Метою дипломної роботи є проєктування та реалізація програмної системи для децентралізованого виконання обчислювальних задач із можливістю автоматичного вибору центра, підтримкою паралелізму та централізованого моніторингу через веб-інтерфейс.

Об'єктом дослідження є програмна архітектура розподіленої обчислювальної системи.

Предметом дослідження є алгоритми самоорганізації, механізми обміну задачами між вузлами, а також структура веб-засобів адміністративного управління системою.

У ході виконання роботи було застосовано методи моделювання програмної архітектури, побудови діаграм UML, реалізації WPF- і ASP.NET Core-додатків, розробки механізмів взаємодії між компонентами на основі WebSocket. Було розроблено систему, що підтримує як прості арифметичні, так і обчислювально складні задачі з можливістю динамічного перерозподілу навантаження, самоорганізацією, автоматичного вибору центра і ручного втручання через веб-панель.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційних технологій супроводжується стрімким зростанням обсягів даних і складності задач, які потребують обчислення. У зв’язку з цим особливої актуальності набувають розподілені обчислювальні системи, здатні виконувати паралельні обчислення на різних вузлах мережі. В умовах, коли традиційні централізовані підходи мають обмежену масштабованість і уразливість до відмов, виникає потреба у створенні децентралізованих систем, які характеризуються гнучкістю, надійністю та автономністю.

Децентралізовані підходи до розподілення навантаження дозволяють забезпечити стійкість до збоїв, автоматичне балансування ресурсів і масштабованість без суттєвої зміни архітектури. Особливо цінною буде можливість кожного вузла не лише отримувати задачі, а й координувати роботу інших вузлів у разі зміни ролей. Такий підхід відкриє широкі можливості як для освітніх цілей (демонстрація розподілених алгоритмів), так і для практичного використання в системах обробки даних невеликого або середнього масштабу.

Метою даної роботи є розробка універсальної розподіленої обчислювальної системи, в основі якої лежить децентралізована архітектура з можливістю автоматичного вибору центра, виконання задач у фоновому режимі та адміністрування через веб-інтерфейс. У роботі буде реалізовано як механізми самореєстрації та самоорганізації компонент, так і централізовані засоби керування, що забезпечать повний цикл взаємодії між вузлами та користувачем.

У рамках роботи буде здійснено огляд існуючих архітектур розподілених систем, обґрунтовано вибір інструментальних засобів, спроєктовано структуру взаємодії між компонентами, створено алгоритми виконання задач і обміну статусами, буде розроблено графічні інтерфейси для адміністратора та компоненти-учасника, а також проведено тестування системи в умовах різного навантаження. Отримані результати демонструватимуть функціональну повноту реалізованого рішення та його практичну придатність до використання.

АНАЛІЗ ВІДОМИХ РІШЕНЬ ТА ВИБІР СТРАТЕГІЇ І ЗАСОБІВ РЕАЛІЗАЦІЇ ЗАВДАННЯ

1.1 Огляд існуючих розподілених систем та аналіз сучасних підходів та реалізацій до розпаралелювання завдань

Розподілені системи – це система, що складається з сукупності незалежних комп’ютерів, які об’єднуються між собою в одну мережу. Взаємодія між частинами системи утворює один організм, який може розв’язувати складні задачі набагато швидше. Для того, щоб всі елементи системи працювали узгоджено між собою потрібне правильне узгодження завдань, розподілення ресурсів та правильне використання спільних ресурсів [1-3].

Ознаки:

– система складається з декількох зв’язаних між собою комп’ютерів, які займаються обміном даними через мережу;

– в системі має забезпечуватися доступ до спільних ресурсів, наприклад: пам’ять, обчислювана потужність, сховища та даних між усіма частинами системи;

– система має легко підстроюватися під зростання потреб, шляхом додавання нових комп’ютерів;

– якщо якийсь комп’ютер в системі перестає працювати, то система безперебійно продовжує працювати далі;

– для користувача система має відчуватися як одне ціле, він не має відчувати складність системи.[1]

Для розподілених систем є визначено декілька основних видів архітектур [4].

Однією з них являється багаторівнева архітектура.

Багаторівнева архітектура в розподілених системах являє собою поділ системи на ієрархічні рівні. В даній системі кожен рівень має певні задачі та функції, які він виконує та зону відповідальності. Цей підхід дає можливість доволі ефективно керувати складною системою і дає можливість розподіляти відповідальність між кожною з частин системи.

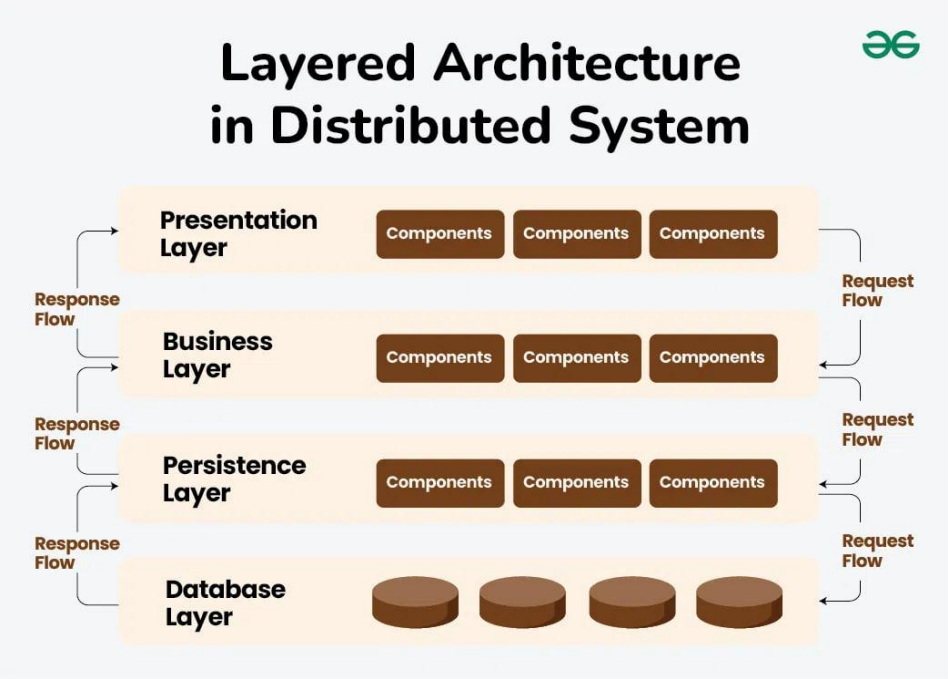


Рисунок 1.1 – Багаторівнева архітектура [1]

Переваги:

– кожен рівень має свої обов’язки, це дозволяє простіше розробляти складні системи;

– система є гнучкою, оскільки зміни на одному рівні мінімально заторгують всі інші;

– компоненти системи можливо використовувати в інших проєктах;

– кожен з рівнів можливо розширити, незалежно від всіх інших.

Недоліки:

– передавання даних між рівнями додає додаткові витрати ресурсів, що може знижувати швидкість системи;

– наявність багатьох рівнів ускладнює архітектуру, так як і спілкування між ними;

– іноді зміни в логіці можуть потребувати зміни коду на багатьох рівнях системи, що може ускладнити розробку.

Найвідомішим прикладом даної архітектури є веб-додатки. У них зазвичай є рівень виводу даних до користувача, рівень бізнес-логіки продукту і рівень що працює з базою даних і відповідає за дані. Також корпоративні системи часто мають розділення між виводом користувачу, бізнес-логікою та керування даними [1].

Архітектура клієнт-сервер – це одна з основних моделей для розподілених систем. Основними компонентами системи є клієнт і сервер. Між ними розподіляється основна логіка та функції. Клієнти надсилають запити для підключення до потрібних функцій чи ресурсів, які має центр цієї системи – сервер, а сервер отримує ці запити та повертає результат на них. Клієнт – це ініціатор, сервер – постачальник функцій та ресурсів. Управління є відповідальністю сервера, а клієнт відображає отримані дані, виводить їх.

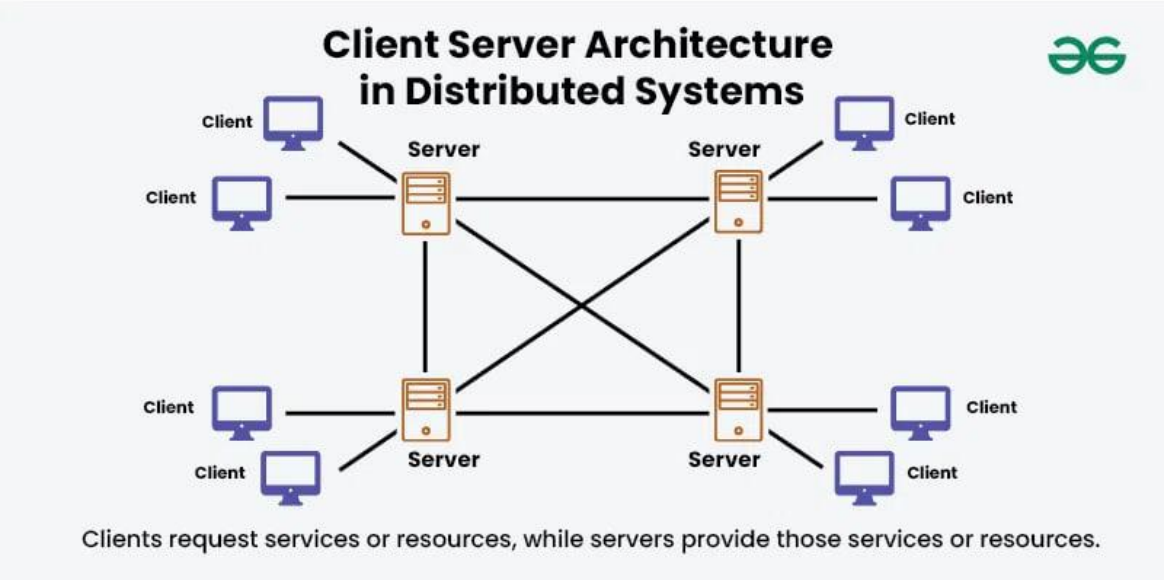


Рисунок. 1.2 - Клієнт-серверна архітектура [1]

Клієнт-серверна архітектура відіграє ключову роль у проєктуванні систем завдяки таким перевагам [5]:

– усі ресурси знаходяться у центрі системи – сервері, що дозволяє легше оновлювати систему, управляти нею тощо;

– система спокійно може зростати, якщо з’являється у цьому потреба і потрібна більша потужність можна додати кілька серверів;

– система дозволяє більш раціонально використовувати ресурси, через те що важкі розрахунки виконує сервер, а клієнт тільки отримує дані та виводить їх користувачу;

– серверна частина може балансувати навантаження, робити резервне копіювання, яке допомагає забезпечити стабільну роботу та мінімальні простої;

– так як система є централізованою, то її можна зробити більш безпечною завдяки шифруванню, авторизації тощо.

Але в даної системи також є і недоліки, такі як:

– якщо сервер ламається, то система зупиняється, що робить її вразливою в центральній частині;

– якщо до сервера забагато запитів, це може призвести до затримок в системі;

– доступ клієнтів до сервера повністю залежить від інтернет-підключення;

– при високому навантаженні збільшується вірогідність появи вузьких місць системи.

Основними прикладами даної архітектури є веб-додатки, де браузери є клієнтами, вони відправляють запити до веб-серверів, для того, щоб отримати дані або веб-сторінки. В системах керування базами даних клієнтські програми відправляють запити до серверів баз даних, для отримання доступу до обробки та оновлення інформації.

Peer-to-Peer (P2P) – це децентралізована мережа, у якій кожен учасник являється і клієнтом, і сервером, тобто надає й отримує ресурси й функції. Від класичної клієнт-серверної архітектури P2P відрізняється тим, що має всі вузли рівноправні: вони можуть самі надсилати й обробляти запити. Цей підхід дозволяє більш ефективно обмінюватися розрахунками, ресурсами та іншими даними без того, щоб мати центр системи.

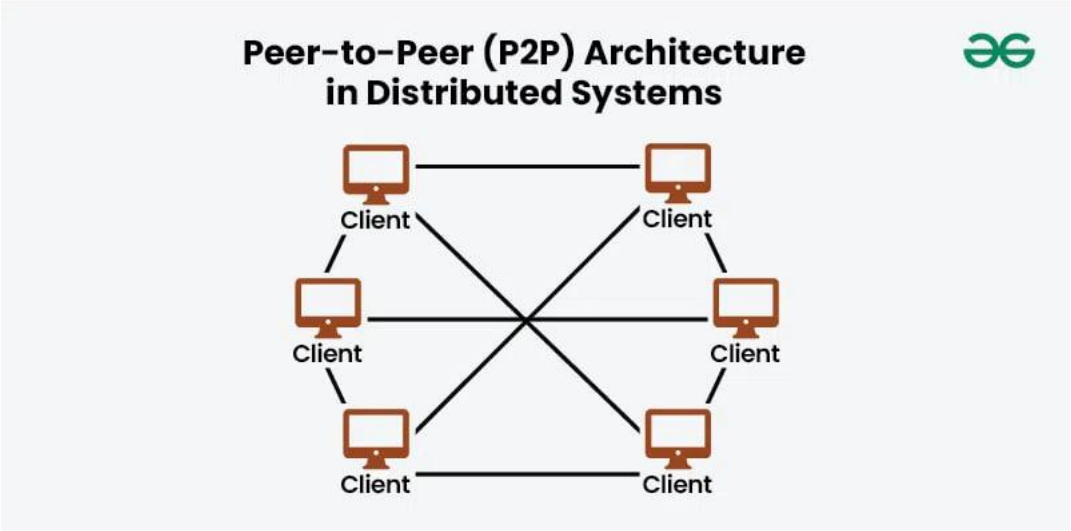


Рисунок 1.3 - Peer-to-Peer архітектура [1]

Основні характеристики Peer-to-Peer (P2P) архітектури[6]:

– вузли системи є повністю децентралізованими;

– мережа може дуже просто зростати, через відсутність центру, який обмежує систему в максимальній кількості ресурсів;

– мережа є дуже стійкою, через відсутність центрального вузла, що може перевантажитись і вся система перестане працювати;

– вузли системи напряму передають один одному дані;

– кожен вузол є автономним, він сам приймає рішення, що робить систему гнучкою.

Переваги [1]:

– якщо вузол перестає працювати, то це не впливає на роботу усієї системи і вона продовжує працювати без нього;

– через відсутність центральних серверів система являється набагато дешевшою ніж централізована;

– систему дуже легко збільшувати, чим більше вузлів: тим краще працює система.

Недоліки:

– через відсутність централізованого управління, то систему легше пошкодити;

– продуктивність не є стабільною, все залежить від окремих частин системи;

– систему дуже складно розробляти й складно нею управляти, так як вона є незалежною.

Основним прикладом використання децентралізованої архітектури Peer-to-Peer (P2P) є файлообмінні мережі – наприклад, BitTorrent. Він дозволяє користувачам передавати файли напряму, там кожен з учасників одночасно і завантажує і поширює частини свого файлу, що забезпечує ефективний обмін. В децентралізованих застосунках, що функціонують у P2P-мережах без централізованого контролю, використовують однорангову архітектуру для того, щоб зберігати дані, робити обчислення тощо.

Архітектура, керована подіями – це архітектурний підхід, у якому основними елементами системи являється подія. Через події відбувається обробка даних і управління логікою даної системи. У цій архітектурі її компоненти взаємодіють між собою за допомогою генерації й обробки подій. Події відображають зміни стану чи певні дії, що відбувають у даній системі[3].

Архітектура має такі переваги [7]:

– система швидко реагує на зміни: вона спрацьовує, коли відбувається будь-яка подія;

– систему легко можна розширювати за допомогою додавання нових компонент;

– система обробляє любі події негайно, що робить її дуже швидкою та ефективною;

– система, в якій відбувається обмін подіями, є децентралізованою, що дозволяє простіше управляти нею.

Недоліки архітектури:

– систему складно реалізовувати;

– у системі складно гарантувати чи події виконуються у правильній послідовності;

– система асинхронна і децентралізована, її складно тестувати і реалізовувати.

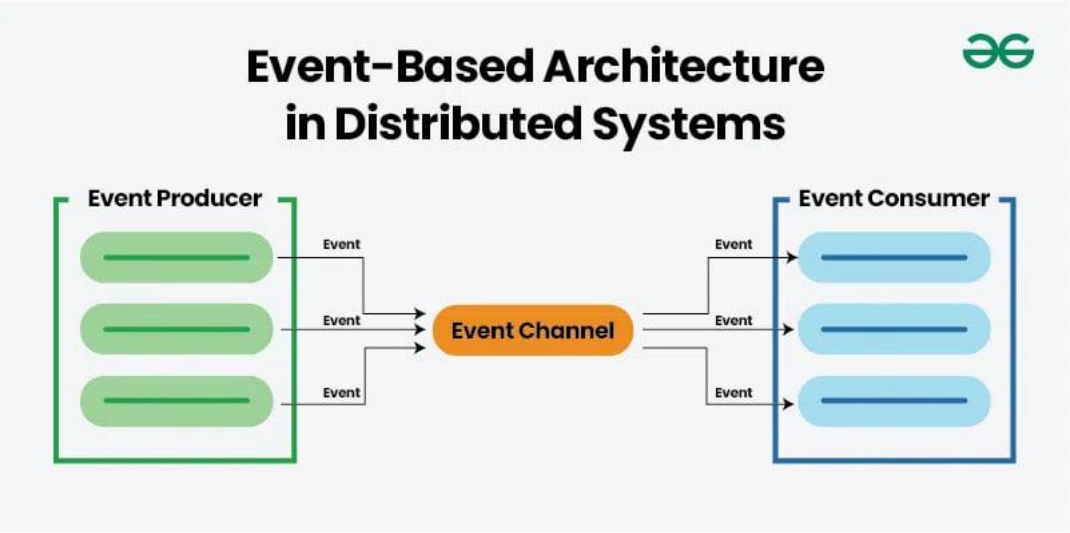


Рисунок 1.4 - Архітектура, керована подіями [1]

Найчастішими прикладами застосування даної архітектури є аналітика в реальному часі, наприклад торгові платформи на фінансових ринках, для швидкої обробки ринкових подій та прийняття рішень. Також системи інтернет-речей застосовують її для того, щоб реагувати на дані від різних датчиків та пристроїв у реальному часі. Систему можна використовувати для виявлення шахрайства, у фінансовому секторі й вона допомагає оперативно виявляти аномальну активність та реагувати на потенційні загрози дуже швидко.

1.2 Вибір технологій для реалізації розподіленої системи

Під час розробки розподіленої універсальної системи для розпаралелювання завдань ключовим аспектом є вибір відповідних технологій, які забезпечать масштабованість, надійність, гнучкість та ефективність взаємодії між компонентами системи [8]. Враховуючи сучасні вимоги до програмних рішень і завдання, які ставляться перед системою, було обрано стек технологій, що включає

C# як основну мову програмування, а також ASP.NET Core MVC та WPF для реалізації різних частин системи.

Основною мовою реалізації проєкту є C#[9] (C-Sharp) – це об'єктно-орієнтована мова програмування, що була розроблена компанією Microsoft, як частина платформи .NET. Вона створює поєднання синтаксичної суворості та гнучкості, що властива сучасним мовам, та інтеграцію з інструментами розробки і інфраструктурою Windows[10].

Ключовими перевагами C# являються [11]:

– багатофункціональність та сучасність: C# підтримує імперативний, об'єктно-орієнтований та функціональний типи програмування. В останніх версіях мови (це C# 9.0 і вище) з'явилися records, pattern matching, асинхронні потоки, nullable reference types та інші корисні інновації [9];

– асинхронне програмування: підтримка async/await в мові програмування C# дозволяє ефективно реалізовувати паралельні та розподілені обчислення, які є критично важливими для запланованої системи [12];

– великий набір бібліотек: використання .NET Standard і .NET Core відкриває доступ до дуже великої кількості бібліотек і NuGet-пакетів. Пакети для роботи з мережею, базами даних, безпекою, обробкою даних також доступні і багато іншого;

– інтеграція з Microsoft технологіями: мова програмування ідеально підходить для створення як десктопних застосунків (через WPF/WinForms), так і для веб-додатків (через ASP.NET), а також сервісів і створення API;

– висока продуктивність: завдяки оптимізованій віртуальній машині CLR та компіляції JIT, програми на C# швидко працюють та споживають керовану кількість ресурсів.

Синергія з платформою .NET також дає можливість легко організовувати взаємодію між різними типами клієнтів, працювати з веб-сервісами, базами даних та забезпечувати високий рівень безпеки через механізми аутентифікація й авторизації [13].

Для створення десктопного клієнта, який встановлюється на кожному комп’ютері в мережі, використовується WPF (Windows Presentation Foundation) [14], який дозволяє створювати зручний і гнучкий інтерфейс користувача. WPF (Windows Presentation Foundation) обрано для створення десктопної клієнтської програми, яка встановлюється на кожному комп’ютері мережі [15]. WPF дає можливість створювати якісні, адаптивні та візуально красиві інтерфейси з використанням XAML-розмітки [16]. Завдяки потужній підтримці Model-View-ViewModel патерну[17], WPF дозволяє легко відокремити логіку представлення від логіки обробки даних, що значно полегшує тестування, збільшення обсягу та підтримку клієнтського програмного забезпечення [18]. Окрім цього, WPF має можливість апаратного прискорення, яка забезпечує плавну графіку та високу продуктивність при роботі зі складними візуальними компонентами[19].

Для організації зв’язку між комп’ютерами в мережі використовується WebSocket-протокол. WebSocket – це мережевий протокол, який забезпечує повнодуплексне, двостороннє з'єднання між клієнтом і сервером поверх одного TCP-з'єднання [20]. На відміну від традиційних HTTP-запитів, де кожен запит має бути ініційованим клієнтом, WebSocket дозволяє обом сторонам (тобто клієнту і серверу) ініціювати обмін даними у будь-який момент часу без потреби повторного відкриття з'єднання.

Переваги WebSocket:

– якщо з’єднання було встановлене, то воно залишається відкритим, що дає змогу легко обмінюватися інформацією між клієнтом та сервером.

– дає меншу затримку, оскільки не має витрат на підключення для кожного запиту [21].

– зв’язок іде у 2 сторони, тобто клієнт і сервер можуть спокійно обмінюватися повідомленнями між собою [22].

– дуже ефективно підходить для систем, де потрібно обмінюватися даними миттєво – в момент реального часу.

Для запланованої системи, WebSocket можна буде використати для:

– постійного зв’язку між десктопними клієнтами, встановленими на кожному ЕОМ мережі та центральним сервером;

– оперативного обміну службовими повідомленнями (наприклад, оновлення інформації про навантаження, підтвердження отримання та завершення завдань) [23];

– передавання статусу виконання обчислень та реакції на перевантаження: якщо один вузол повідомляє про високе навантаження, сервер може перенаправити завдання на інший, менш завантажений ЕОМ;

– організації базових механізмів самоорганізації системи – зміна "центру" обробки, автоматичне перепідключення клієнтів, вибір ведучого вузла тощо.

ASP.NET Core MVC – це високопродуктивний, кросплатформений фреймворк від Microsoft для створення веб-додатків, RESTful API та серверної логіки[24]. Реалізує шаблон Model-View-Controller (MVC), який чітко розділяє дані, представлення і бізнес-логіку, що сприяє модульності, легшому тестуванню і підтримці коду [25].

Переваги ASP.NET Core MVC [26]:

– кросплатформеність: ASP.NET Core працює на Windows, Linux і macOS, це дозволяє розгортати систему на будь-якому серверному середовищі.

– висока продуктивність – завдяки оптимізаціям та використанню Kestrel-сервера фреймворк демонструє одну з найкращих продуктивностей серед веб-фреймворків;

– гнучкість налаштування маршрутизації: через Endpoint Routing система може чітко обробляти запити до контролерів, дій і API;

– інтеграція з DI-контейнером (Dependency Injection): нативна підтримка впровадження залежностей дозволяє легко керувати життєвим циклом об'єктів та масштабувати систему;

– безпека: вбудовані засоби автентифікації та авторизації (за допомогою Identity, OAuth, JWT) дозволяють створити захищені веб-інтерфейси з обмеженим доступом;

– підтримка сучасних інструментів розробки: фреймворк легко поєднується з Entity Framework Core для роботи з БД, підтримує Blazor, SignalR, WebSockets, та є сумісним з популярними фронтенд-фреймворками (React, Angular)[27].

У проєкті ASP.NET Core MVC буде потрібен для реалізації:

– закритого веб-інтерфейсу з автентифікацією, який надає адміністративний доступ до системи [28];

– виведення інформації про стан системи: кількість підключених клієнтів, рівень навантаження (процесор, пам’ять), статус виконання задач;

– створення REST API, які викликаються десктопними клієнтами для обміну даними;

– управління завданнями, що надсилаються на виконання в розподілену мережу;

– інтеграції з WebSocket-підсистемою для живого моніторингу та керування розподіленням навантаження.

ASP.NET Core MVC – це ключовий компонент серверної частини системи, він забезпечує гнучкий і розширюваний механізм керування розподіленими обчисленнями [29].

Щоб забезпечити адаптивний, сучасний інтерфейс для веб-додатку буде використано Bootstrap. Bootstrap – це популярний фронтенд-фреймворк із відкритим кодом, розроблений компанією Twitter, який дозволяє швидко створювати адаптивні та зручні інтерфейси користувача [30]. Він містить набір готових HTML, CSS і JavaScript-компонентів для верстання інтерфейсів, сумісних з усіма сучасними браузерами та пристроями.

Bootstrap надає такі переваги, як:

– адаптивність – вбудована сіткова система забезпечує адаптацію інтерфейсу під різні розміри екранів (ЕОМ, планшети, смартфони) [31];

– уніфіковані стилі – стильова система Bootstrap дозволяє швидко оформити елементи (кнопки, форми, таблиці, панелі тощо), забезпечуючи професійний вигляд без додаткової кастомізації [32];

– інтеграція з JavaScript – підтримка інтерактивних елементів (модальні вікна, сповіщення, випадаючі меню) без необхідності писати код "з нуля" [33];

– велика спільнота та документація – Bootstrap має потужну екосистему з великою кількістю шаблонів, UI-компонентів та інструкцій [34].

Також Bootstrap має гарну взаємодію з ASP.NET Core [35]. Він легко інтегрується у Razor Views (cshtml-файли) ASP.NET Core MVC. Для його підключення достатньо приєднати CSS і JS-файли з CDN або встановити Bootstrap через NuGet або NPM. Це дає змогу чітко контролювати вигляд і поведінку інтерфейсу без потреби у складному JavaScript-коді.

Завдяки використанню Bootstrap, веб-інтерфейс системи виглядає сучасно та лаконічно, являється адаптивним під будь-який екран й дозволяє адміністратору інтуїтивно взаємодіяти з усіма функціями системи[36].

Загалом, обрані технології доповнюють одна одну, утворюючи надійну, гнучку та масштабовану платформу для побудови розподіленої універсальної системи, що відповідає сучасним вимогам до продуктивності, безпеки та зручності користування.

1.3 Формування вимог до розподіленої універсальної системи для розпалалелювання завдань та розробка постановки задачі

Розподілена універсальна система для розпаралелювання завдань буде створена у формі десктопної програми. Ця програма має встановлюватись на будь-яку кількість електронних обчислювальних пристроїв. Між даними програмами на різних електронних обчислювальних пристроях має бути налагоджений зв’язок. Даний зв’язок буде реалізовано за допомогою веб-сокетів.

Кожен з електронних обчислювальних пристроїв, що мають запущену програму буде мати доступ до закритого веб-сайту з авторизацією всередину. Сайт не повинен мати публічної частини. Після успішної авторизації користувач буде мати доступ до внутрішньої частини сайту, де буде відображатися інформація

розподіленої системи: кількість електронних обчислювальних пристроїв мережі даної системи (тобто ті, на яких встановлена дана програма), навантаження кожного електронного обчислювального пристрою (пам’ять, процесор) та можливість запуску різних обчислень, які мають бути виконані у розподіленій системі.

У ці обчислення входить: різні математичні задачі, які можна розпаралелити на підзадачі, які будуть обраховуватися на різних вузлах мережі, а результат збирається в центрі мережі.

При виконанні обчислень система має враховувати завантаженість кожного електронних обчислювальних пристрою і якщо завантаженість більше допустимої, то такі пристрої мають ігноруватися.

Проєкт має бути реалізовано мовою програмування C#, використовуючи фреймворки ASP.NET Core MVC та WPF. Для обміну інформацією в системі буде використано формат JSON, що забезпечує легкість обробки між компонентами системи.

1.4 Висновки до першого розділу

У першому розділі було розглянуто різні архітектурні моделі, що використовуються для того, щоб реалізувати систему розпаралелювання обчислень. Було визначено переваги та недоліки даних архітектур. Було проаналізовано наступні архітектури: багаторівнева, клієнт-серверна, децентралізована (Peer-to-Peer) та архітектура, керована подіями.

Із всіх перелічених видів архітектур було обрано децентралізовану архітектуру (P2P) для розроблювальної системи. Дана модель дозволить досягнути високого рівня відмовостійкості, масштабованості та незалежності кожного компонента системи. Замість того, щоб мати серверний один вузол, до якого всі компоненти звертаються, як клієнти, система функціонуватиме, базуючись на самоорганізації. Щоб розподіляти задачі будь-який вузол може стати центром по

розподіленню задач між вузлами, враховуючи їх поточне навантаження. Центр зміщується, враховуючи поточне навантаження. Коли це навантаження перевищується у центрі обирається найменш навантажений вузол, що стане новим центром.

Також було проаналізовано та обрано сучасні технології для реалізації системи. Як мову програмування з потужною підтримкою асинхронного коду та багатопотоковості було обрано C#. Щоб створити захищений адміністративний веб-інтерфейс було обрано фреймворк ASP.NET Core MVC. Для того, щоб створити десктопну програму для компонент з багатим інтерфейсом, було обрано WPF. Щоб забезпечити швидкий обмін даними, протоколом постійного двостороннього зв'язку між компонентами було обрано технологію WebSocket. І для того, щоб швидко побудувати гарний та адаптивний веб-інтерфейс на закритому сайті було обрано фреймворк Bootstrap. Ці засоби реалізації дозволять створити надійну та гнучку систему, яка може легко розширюватися, підтримувати розпаралелювання розрахунків та динамічне балансування навантаження в даній системі. Також буде реалізовано можливість адміністрування системи, не відключаючи при цьому самоорганізацію.

На основі цього розділу було визначено основні технічні вимоги для майбутньої розподіленої системи, визначені основні функції та сформульовано постановку задачі.

АРХІТЕКТУРА РОЗПОДІЛЕНОЇ УНІВЕРСАЛЬНОЇ СИСТЕМИ

2.1 Загальна архітектура розподіленої системи

Після формування вимог до розподіленої універсальної системи для розпаралелювання завдань та розробка постановки задачі, наступним кроком стала побудова загальної архітектури розподіленої системи.

Базовою архітектурною моделлю було обрано архітектуру децентралізованої системи - P2P.

Децентралізована архітектура системи або архітектура peer-to-peer – це такий принцип побудови системи, при якому всі вузли мають рівний статус між собою [37]. Тобто немає основного елементу, від якого залежать всі інші. Така структура потребує самоорганізації. Вся обробка і координація системи здійснюється всіма вузлами.

Одна з сильних сторін такої архітектури є відсутність єдиної точки відмови. Отже, вихід з ладу одного або декількох вузлів не зупинить роботу всієї системи. Мережа буде автоматично адаптуватися до змін і забезпечить відмовостійкість та самовідновлення.

В архітектурі універсальної розподіленої системи для розпаралелювання завдань реалізовано модель керованої децентралізації. В рамках такого гібридного підходу, хоча всі вузли є рівноправними по реалізації та можливостям, один з цих вузлів тимчасово виконує функцію центру системи. Такий центр розподіляє завдання, збирає результати розрахунків, аналізує навантаження всіх вузлів. Це має деякі ознаки централізації, але основна відмінність від централізованої системи, полягає у тому, що ці ролі є динамічними: центр не є жорстко закріпленим, він може змінюватись у реальному часі на основі аналізу завантаженості вузлів.

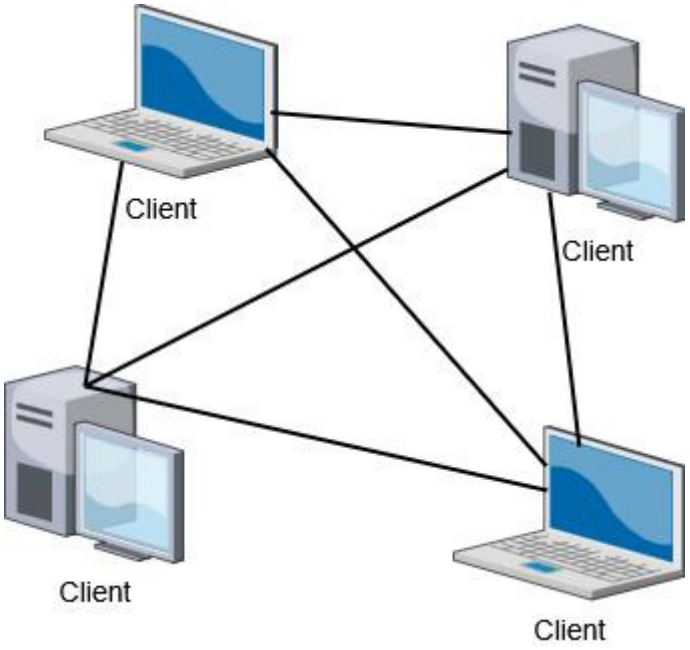


Рисунок 2.1 – Архітектура розподіленої децентралізованої системи P2P

Система реалізована з використанням повнозв’язного WebSocket-з’єднання. Кожен вузол встановлює прямий канал зв’язку з іншими. Така структура дозволяє здійснювати обмін даними про свою завантаженість, в режимі реального часу без посередництва центра.

Процес самоорганізації є одним з найважливіших елементів системи. Кожен з вузлів регулярно надсилає свій статус іншим учасникам. Поточна центральна компонента займається аналізом отриманої інформації. У випадку перевищення граничного навантаження (80% використання центрального процесора) центральна компонента ініціює передачу управління на інший вузол з найменшим поточним навантаженням. Такий алгоритм забезпечує динамічну рівновагу у системі та стійкість до перевантаження.

Система має два типи компонент: центральний і виконавчий.

Центральна компонента відповідає за розподіл обчислень між різними вузлами системи та обчислення фінального результату.

Виконавча компонента отримує розрахунки, виконує їх та відправляє результати назад.

Вузол являється WPF десктопною програмою, що встановлюється на електронний обчислювальний пристрій [38].

Також у системі є адміністративний веб-сайт, що стоїть на одному пристрої з компонентою системи. Адміністратор має можливість зайти на закритий веб-сайт, захищений авторизацією, де він може переглядати інформацію розподіленої системи, запускати розрахунки, виключати клієнтів із системи, отримувати результат розрахунків. Адміністративний веб-сайт є ASP.NET Core MVC веб-додатком [39-40].

В кожної з компонентів систем є своя архітектура та логіка. Веб-сайт адміністратора побудований за принципом MVC.

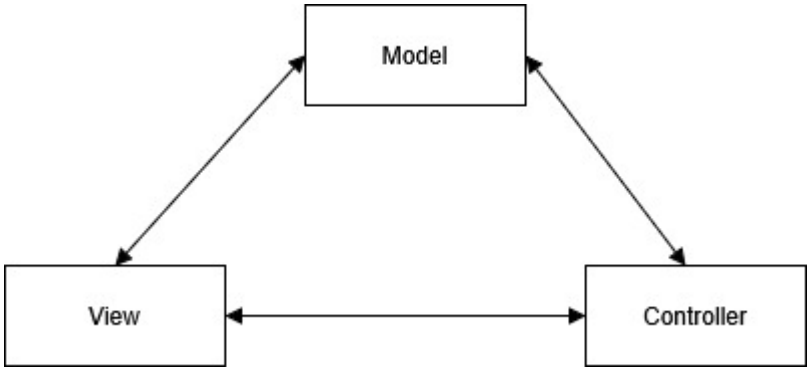


Рисунок 2.2 – Архітектура розподіленої системи клієнт-сервер

MVC (Model-View-Controller) – є архітектурним шаблоном для структурування коду веб-додатків. Для веб-сайту даний архітектурний шаблон дозволяє розділити обов’язки наступним чином:

– Модель (Model) – відповідає за основні сутності цієї системи. Додатково використовуються сервіси, що забезпечують бізнес-логіку роботи з даними моделі для таких задач, як відображення завантаженості компонентів системи; розрахункові задачі, що будуть виконуватися компонентами; статус завдань тощо.

– Представлення (View) – відповідає за відображення даних для адміністратора системи через веб-інтерфейс, а саме, список підключених компонентів, що являються вузлами розподіленої системи, навантаження цих

вузлів, завдання та їх статус виконання. Також адміністратор, використовуючи веб-інтерфейс має можливість запускати розрахунки, виключати елементи мережі тощо.

– Контролер (Controller) – відповідає за обробку запитів від адміністратора, що користується веб-інтерфейсом, взаємодію з моделями використовуючи бізнес-логіку сервісів, передачу даних до представлень, які будуть відображатися на веб-сторінці.

Таким чином, на центральному сервері MVC забезпечує чіткий поділ обов'язків між обробкою даних, бізнес-логікою та інтерфейсом користувача, що спрощує розробку, тестування і підтримку системи.

Наступним кроком було вирішено створити діаграму класів для розподіленої системи.

Діаграма класів являється однією з основних діаграм у мові моделювання UML (Unified Modeling Language). Ця діаграма використовується для відображення, проєктування та точного опису структури розроблювальної системи. Діаграма відображає класи, атрибути, методи та зв'язки між цими класами. Це дозволяє поглянути на програмну систему з абстрактного рівня, зрозуміти створену архітектуру і покращити її при необхідності.

Основними елементами діаграми класів є класи, інтерфейси, зв'язки (асоціація, успадкування, агрегація, композиція).

Класи та інтерфейси відображені у вигляді прямокутника, який поділяється на декілька частин. У цих частинах вказується ім'я даного класу, його атрибути, методи та рівень доступу.

Класи мають між собою зв'язки. Вони є різних видів: асоціація, успадкування, агрегація та композиція. Асоціація вказує на деяку взаємодію між класами. Композиція – сильний зв'язок між класами, де один з них являється частиною іншого. Клас, що є частиною іншого, не може існувати без основного класу. Агрегація - це слабкий зв'язок між класами, ці класи можуть існувати незалежно один від одного. Успадкування показує ієрархію класів.

Для проєктування розподіленої децентралізованої системи діаграма класів є дуже важливим кроком. Вона дозволяє зрозуміти зв'язки між ключовими сутностями системи, визначити правильну архітектуру для правильної взаємодії елементів системи між собою і всередині.

Далі буде представлено діаграму класів для компоненти системи.

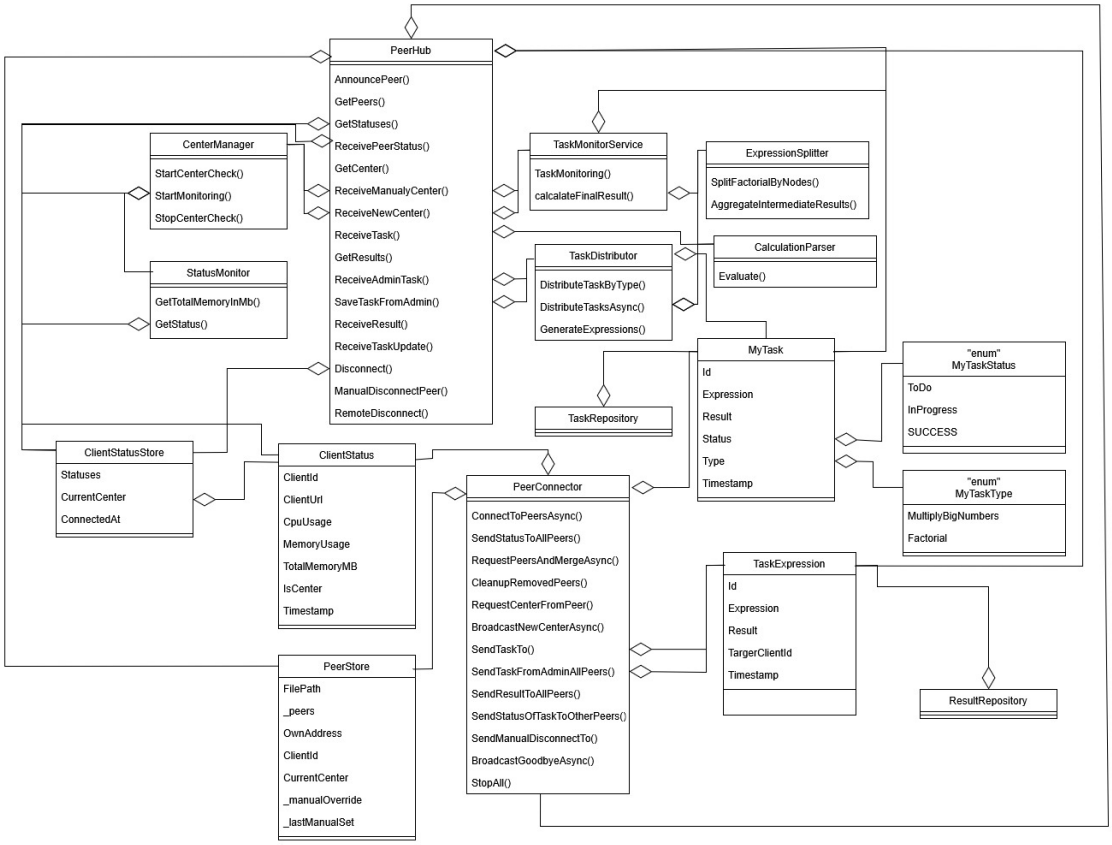


Рисунок 2.3 – Діаграма класів для компоненти

За децентралізоване з'єднанням між вузлами відповідають класи PeerHub, PeerConnector та PeerStore.

PeerConnector відповідає за відправлення інформації потрібним вузлам. Він містить та керує всіма з'єднаннями у системі. Він виконує основне керування системою між вузлами. Інформацію про вузли, поточний центр та підключення зі сторони адміністратора він отримує з PeerStore.

PeerHub містить функції, які інші вузли можуть викликати один в одного. Він допомагає отримати статуси, інформацію про всі вузли, підключені до системи.

Також він відповідає за отримання центру, перемикання центру, отримання задач, результатів, завдань від адміністратора, а також дозволяє відключатися від системи.

MyTask це модель для задач, які центр буде розпаралелювати між компонентами, щоб створити сильну розподілену систему. Для даної моделі також є перелічування для статусів задачі та типу задачі. Перелічування для статусів задачі дає можливість зрозуміти, на якому етапі життєвого циклу знаходиться задача, а перелічування для типу задачі дає розуміти як цю задачу розбивати на підзадачі.

TaskExpression являється моделлю для розпаралелених частин задачі (підзадач), які отримують компоненти від центра. Компоненти такі задачі обраховують, записують в них результат та відправляють всім вузлам. Центр, перевіряє кількість отриманих частин та обраховує ці результати в один кінцевий результат. Після цього центр відправляє даний результат всім іншим вузлам і вузол, який має з'єднання з адміністративним веб-сайтом відправить йому результат.

MyTask зберігає інформацію про себе в MyTaskRepository. TaskExpression зберігає інформацію про себе в ResultRepository. Це локальна база даних SQLLite, яка є в кожного компонента. Вона дозволяє зберігати, переглядати та використовувати інформацію про задачі та розрахунки, ділитися нею між вузлами та відправляти її за запитом адміністративного сайту.

Для того, щоб розподілити вираз задачі на підзадачі, які можна розпаралелити, використовується клас TaskDistributor. Він розділяє вираз на підзадачі, та виділяє доступні вузли, які мають допустиму навантаженість та розподіляє підзадачі між ними. Коли приходять результати з вузлів, він з'єднує всі проміжні результати в кінцевий. Для цього він використовує допоміжний клас, як ExpressionSplitter. TaskMonitorService – це сервіс який при зміні центру слідкує за тим, щоб поставлені задачі були виконані і розрахунки не загубилися при цьому. Коли вузол отримує підзадачу у вигляді json-об'єкту, то він визначає і обраховує вираз зі строки за допомогою класу CalculationParser.

ClientStatus, є моделлю, яка відображає навантаження кожного клієнта, що дозволить системі визначати, куди потрібно перенаправляти розрахункові задачі для ефективнішого їх виконання.

ClientStatusStore зберігає у собі інформацію: про статуси клієнтів, хто є центром та хто коли підключився, що дозволяє правильно переносити центр.

StatusMonitor відповідає за статус конкретного вузла, він збирає інформацію про завантаженість оперативної пам’яті та центрального процесора. Також він збирає інформацію про те, скільки всього оперативної пам’яті має вузол.

CenterManager відповідає за перевірку того, чи конкретний вузол має допустиме навантаження, щоб ефективно бути центром. Якщо навантаженість переходить дозволену межу, центр відправляє сигнал для перемикання до іншого, найменш навантаженого, доступного вузла.

Наступною буде представлено діаграму класів для веб-сайту адміністратора.

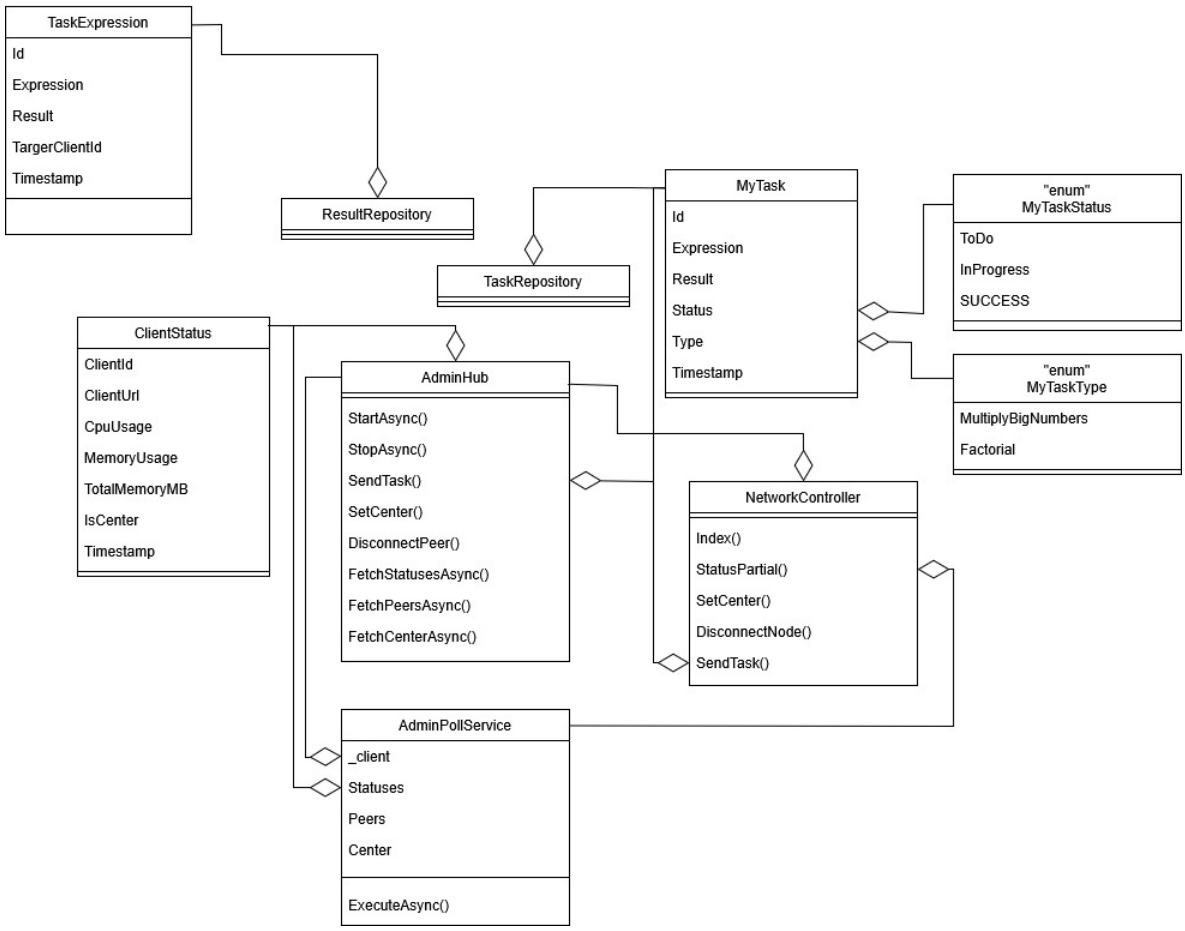


Рисунок 2.4 – Діаграма класів для сайту адміністратора

Веб-сайт дублює моделі MyTask і перелічування для статусу його життєвого циклу та типу задач, модель підзадач TaskExpression та їх репозиторії для доступу до даних TaskRepository та ResultRepository. Також сайт дублює модель клієнтського статусу ClientStatus. Вони відповідають за ті ж функції, що і в компонентах та були описані раніше. У веб-сайта є 3 унікальні класи AdminHub, AdminPollService, NetworkController.

AdminHub містить у собі функції, які дозволяють відправляти задачі, задавати центр, відключати вузол та отримувати інформацію про всі компоненти системи. Ці всі дані він бере з вузла, використовуючи WebSocket з'єднання.

AdminPollService який відповідає за те, щоб періодично викликати методи з AdminHub та отримувати інформацію про підключені вузли, їх статуси та хто є поточним центром. Це дозволяє автоматично оновлювати інформацію на сторінці у реальному часі.

NetworkController відповідає за те, щоб отримуючи запити, вибирати потрібну інформацію і повертати вигляд з потрібною інформацією. Це дозволяє виконувати адміністративні дії з веб-інтерфейсу.

2.2 Функціональні можливості та типи розпаралелювання завдань

Функціональні можливості системи будуть відображені в діаграмі прецедентів.

Діаграма прецедентів – це один з найбільш використовуваних основних типів UML діаграм. Її використовують для того, щоб відобразити функціональні вимоги до системи у вигляді діаграми.

Діаграма показує взаємодію учасників, тобто акторів, із системою використовуючи визначені сценарії використання, тобто прецеденти. Це дозволяє поглянути на систему на рівні її функціоналу і зрозуміти вимоги та можливості даної системи.

Призначення даної діаграми це відображення того, що система має робити, а не те, яким способом вона це робить. Така діаграма дає можливість всім учасникам розробки з різних спеціальностей отримати спільне бачення системи та бачення очікувань з перспективи її користувачів.

Основними елементами діаграми прецедентів є актори – учасники системи, use case або прецеденти, зв'язки між акторами та прецедентами. Зв'язки поділяються на 3 типи відносин: включення або include, розширення або extend, та узагальнення. Актори – це зовнішні сутності системи, які з нею взаємодіють. Актори можуть бути людьми, процесами, системами тощо. Прецедент є описанням функції системи чи поведінки системи, які вона має зробити на запит актора.

Для розроблювальної розподільної децентралізованої системи діаграма дозволяє чітко структурувати та ідентифікувати функціонал, виявити та прибрати дублювання (щоб покращити архітектуру), спростити аналіз поставлених вимог та стати основним кроком для подальшої розробки. Також діаграма є важливим кроком на етапах уточнення вимог та кращого розуміння технічного завдання та тестування функціоналу системи.

На діаграмі (Додаток А) відображено 3 актори: "Адміністратор", "Центральна компонента" та "Виконувальна компонента", останні 2 генералізуються в актора під назвою "Компонента". Дані актори представляють різні ролі у децентралізованій системі й мають унікальні сценарії взаємодії.

"Компонента" робить наступні функції:

– Відправляє інформацію про її додавання усім компонентам системи. Тобто при запуску вона відправляє сигнал всім для того, щоб повідомити про своє перебування і стати частиною системи. Кожна компонента, що вже є у системі отримує інформацію про під'єднання нової компоненти та записує собі у файл, який містить список елементів системи з їх URL адресами. Так елементи системи дізнаються один про одного і можуть обмінюватися інформацією між собою.

– Відправляє інформацію про її від'єднання від системи, коли відключається. Кожна компонента системи отримує цю інформацію, та змінює

файл зі списком вузлів системи і більше не відправляє інформацію на цю адресу, до цього вузла.

– Кожна компонента, після того, як стає елементом системи, починає ділитися своєю завантаженістю з іншими вузлами цієї системи. Завантаженість відправляється у формі статусу даного вузла та містить завантаженість центрального процесора у відсотках та кількість використаної оперативної пам’яті у Мб. Кожен з вузлів отримує завантаженість системи один одного та записує всю інформацію у файл. Тобто кожен елемент системи знає завантаженість всіх елементів системи.

Після цього “Компонента” поділяється на 2 основні ролі “Центральна компонента” та “Виконавча компонента”.

“Центральна компонента” виконує наступні функції:

– Формує завдання, що будуть розподілені між “Виконавчими компонентами”. Для цього він перевіряє завантаженість всіх компонент та відправляє завдання згідно з нею. Якщо комп’ютер навантажений більше допустимого, то його система пропускає. Інші ж комп’ютери сортуються згідно завантаженості та отримують завдання згідно цього порядку. Найбільш навантажені вузли отримають їх останніми і у меншій кількості. Завдання являють собою розрахунки.

– Отримує результати завдань. Центр отримує результати розрахунків від інших компонент та виводить їх.

– Перевіряє чи завантаженість її системи є допустимою. Якщо ні, то тоді починається зміна центру. Центр обирає найменш навантажену компоненту системи та переносить центр на неї. Сам минулий центр перестає виконувати всі функції центра, та стає “Виконавчою компонентою”, а елемент, що був обраний як новий центр, змінює свою роль від “Виконавчої компоненти” до “Центральної компоненти”.

“Виконавча компонента” виконує наступні функції:

– Отримує завдання від центру. Від центру приходять розрахунки які потрібно виконати. Але для початку їх треба перетворити з тексту у виконавчу форму, яка дозволить виконувати ці розрахунки.

– Виконує завдання. Компонента обраховує отримане завдання та видає результат.

– Відправляє результат до центру системи, щоб він міг вивести його.

Останнім актором системи являється “Адміністратор”. Він може робити ручне управління працюючої системи.

“Адміністратор” має функції:

– Логін. Адміністратор має доступ до закритого веб-сайту. Щоб отримати доступ до управління системою він має спочатку авторизуватися, використовуючи логін та пароль. Якщо вони правильні, то він буде переведений на сторінку де будуть відображені дані системи, а його токен буде збережений у сховищі веб-браузеру.

– Перегляд компонент системи. Адміністратор може переглядати вузли, що активні у системі. Ці дані він отримує з файлу, який міститься на вузлі, що працює на його комп’ютері. Файл, що містить посилання на всі вузли системи.

– Перегляд статусів компонентів системи. Адміністратору будуть відображатися всі статуси всіх вузлів системи. Дані будуть отримані також з файлу з вузла на його комп’ютері. Файл, що містить інформацію про статуси компонент всієї системи.

– Вручну змінювати центр системи.

– Вручну відключати компоненту системи.

– Вручну обирати які завдання, в якій кількості має розпаралелювати та обраховувати система.

Система розрахована на розпаралелювання задач та обрахування їх за допомогою вузлів. Для цього були обрані складні обчислення, які можна розпаралелити між комп’ютерами і дістати результат.

Одним із базових є додавання та віднімання великих чисел. Ці числа будуть перевищувати діапазон стандартних типів даних та використовувати BugInteger.

Приклад:

$$123456789123456789 + 987654321987654321 - 184354221957624523 \quad (2.1)$$

Центр формує деяку кількість виразів для яких випадково обрані великі числа. Після цього він відправляє їх вузлам, у вигляді json-об'єкта, які у свою чергу розбивають рядок, переводять числа в BigInteger, та виконують потрібну операцію. Після цього компонента повертає результат.

Більш складною задачею є обчислення факторіалу.

Формула факторіалу:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \quad (2.2)$$

Обчислення факторіалу у системі можна розпаралелити за допомогою діапазонів. Формула для виконання обчислень факторіалу у системі буде виглядати наступним чином:

$$n! = \prod_{i=1}^{n} i = (1 \cdot 2 \cdot \ldots \cdot k_1)(k_1 + 1 \cdot \ldots \cdot k_2) \ldots (k_{m-1} + 1 \cdot \ldots \cdot n) \quad (2.3)$$

де $k_1, k_2, \ldots, k_{m-1}$ – є межами піддіапазонів, на які розбивається множник n!. Вони визначають, де закінчується один шматок розрахунків і починається інший;

m – це кількість вузлів, між якими розподіляється обчислення. Тобто n! розбивається на m часткових добутків.

Така формула дозволяє розділити завдання на піддіапазони та роздати їх різним вузлам. Наприклад, якщо потрібно обчислити факторіал 100 і у системі є 4 вузли. Центр створить наступні діапазони: для 1 вузла від 1 до 25, для 2 вузла від 26 до 50, для 3 вузла від 51 до 75, та для 4 від 76 до 100. Кожна з компонент виконає

розрахунки добутку даних діапазонів та поверне результат на сервер. Центр у свою чергу збере їх та об’єднає, розрахувавши добуток часткових результатів. Цей добуток можна далі розпаралелити при потребі між вузлами.

В результаті буде отримано один цикл множення на вузол, при більшому масштабі системи розрахунки будуть проводитись ще швидше та простіше.

У системі можна розпаралелити суму чисел у великому масиві.

Формула:

$$S = \sum_{i=1}^{n} x_i, \tag{2.4}$$

де x_i – це елементи масиву.

Центр отримує масив чисел, які потрібно між собою додати (наприклад 10 000 чисел). Центр ділить масив на підчастини, відповідно до кількості доступних вузлів з допустимим навантаженням. Кожен з вузлів отримує свій підмасив у вигляді json-об’єкту та команду того, що потрібно з цими числами робити. Компонента сумує числа та повертає результат до центру. Центр збирає всі часткові добутки та додає між собою за формулою:

$$S = \sum_{j=1}^{m} S_i, \tag{2.5}$$

де S_i – це часткова сума j-тої компоненти;

m – кількість активних вузлів.

В результаті отримано швидкі обрахунки дуже великих масивів, завдяки розподіленню їх між вузлами.

У системі можна розпаралелити задачу пошуку мінімального та максимального значення у великому масиві.

Для реалізації цього масив поділяється на підмасиви, які розподіляються між активними компонентами системи. Наприклад: масив з 10 000 чисел розбивається на 10 частин по 1 000 елементів. Кожен з компонент знаходить мінімум або

максимум (залежно від команди, що прийшла у від адміністратора). Потім вони повертають значення до центральної компоненти. Компонента збирає усі результати вузлів, та обчислює глобальний мінімум або максимум.

В результаті шукане число у даному масиві можна знайти набагато швидше, використовуючи механізм розпаралелювання обрахунків частин масиву між вузлами.

У системі можна реалізувати перевірку парності великої кількості чисел.

З великого масиву чисел потрібно знайти парні числа. Щоб розпаралелити таку задачу, потрібно розбити масив на кількість вузлів і кожен з них виконає операцію перевірки на парність. Вузол формує список парних чисел та повертає результат до центру, для того, щоб об'єднати результати усіх вузлів в один. Центр отримує список парних чисел.

У системі можна реалізувати обчислення середньоквадратичного відхилення (СКВ). СКВ – це статистична метрика, що показує ступінь розкиду даних відносно середнього значення. Для того, щоб реалізувати розпаралелювання СКВ буде обчислюватися для масиву чисел.

Формула:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}\,, \tag{2.6}$$

де x_i - елементи масиву;

$\bar{x}$ - середнє арифметичне масиву;

n - кількість елементів.

Спочатку центр отримує великий масив чисел (наприклад на 10 000 елементів). Після цього він обчислює середнє арифметичне, за наступною формулою:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.7}$$

Це також можна зробити розподілено: розіслати частини масиву вузлам, кожен з вузлів поверне часткову суму та кількість елементів. Цент підсумує результати та обчислить $\bar{x}$.

Після того, як середнє арифметичне було обчислене, центр знову ділить масив на підмасиви. Кожний вузол отримує свою частину та середнє значення $\bar{x}$.

Вузли обчислюють часткову суму квадратів відхилень, за формулою:

$$partial = \sum_{i=1}^{k}(x_i - \bar{x})^2 \tag{2.8}$$

І після розрахунків повертають результат центру, разом з кількістю елементів, які було оброблено. Центр у свою чергу збирає всі результати обчислень та обчислює повне середньоквадратичне відхилення.

В результаті було розпаралелено статистичну задачу швидко та ефективно знайдено результат. Дана задача може слугувати підґрунтям для складніших статистичних операцій.

Складнішим завданням, яке можна реалізувати в універсальній розподіленій системі є обчислення скалярного добутку.

Наприклад, потрібно обчислити скалярний добуток двох числових векторів, які мають однакову довжину.

Формула:

$$\vec{a} = [a_1, a_2, a_3], \qquad \vec{b} = [b_1, b_2, b_3],$$
$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i \cdot b_i \tag{2.9}$$

Таке завдання можна прямолінійно розподілити між вузлами, так як кожен добуток $a_i \cdot b_i$ можна обчислити незалежно.

Спочатку центр отримує два вектори. Після цього він розділить їх на піддіапазони. Кожен з вузлів обчислить часткову суму. Після цього центр збере результати та підсумує їх. В результаті буде отримано скалярний добуток.

Дана задача є практичною і використовується у реальному житті: в комп'ютерній графіці, машинному навчанні, статистиці тощо. Розподілена система дозволяє виконувати розрахунки даної задачі набагато швидше.

Одним з більш складних завдань, які можна реалізувати в децентралізованій універсальній розподіленій системі є щільне множення матриць.

Наприклад, необхідно помножити дві щільні матриці однакового розміру (100×100, 250×250 тощо). Щільні матриці – це матриці, в яких переважна більшість елементів більша за 0.

Якщо матриця A – розміром $M \times N$, а матриця B – розміром $N \times P$, то результатом множення матриць C, буде матриця розміром $M \times P$, де:

$$C[i][j] = \sum_k^n A[i][k] \cdot B[k][j], \tag{2.10}$$

де i - індекс рядка в результуючій матриці C (від 1 до M);

j - індекс стовбця в результуючій матриці C (від 1 до P);

k - індекс перемножуваних елементів у рядку матриці A і стовпці B.

Кожен елемент матриці може обчислюватися незалежно тому його легко розділити на завдання для вузлів.

Центр розділяє матрицю A по рядках. Після цього він відправляє кожному вузлу один або кілька рядків матриці A (залежно від кількості наявних активних вузлів) та повну матрицю B. Після цього вузол обчислює відповідну частину результуючої матриці C. Результат відправляє вузол до центру, де центр формує повну матрицю результату. Важливо також при відправці завдання від вузла до центру вказувати який це рядок матриці, щоб отримавши результат центр міг зрозуміти в якому вигляді має бути результуюча матриця.

Такі обчислення часто використовуються у графіці, машинному навчанні тощо. Тобто дана задача використовується в реальному житті і розподілена система пришвидшує обрахунки великих матриць.

В результаті, використання діаграми прецедентів забезпечує ясне бачення системи для розробника, технічного персоналу та користувачем, покращує розробку та розуміння технічного завдання. У системі функціонують 3 актори: “Адміністратор”, “Центральна компонента” та “Виконувальна компонента”, які мають великий функціонал. Сама система працює децентралізовано та незалежно від зовнішнього впливу і тільки адміністратор може вручну управляти нею та переглядати дані системи, якщо це потрібно.

В результаті у цьому підрозділі було визначено перелік типових задач, які можна ефективно реалізувати використавши паралельні обчислення. До таких задач належать:

– додавання та віднімання великих чисел, що виконуються за межами стандартних типів даних;

– перевірка парності великої кількості чисел;

– обчислення факторіалу, розділеного на добутки діапазонів чисел, які обчислюються незалежно;

– сума елементів великого масиву, яка розподіляється між вузлами та об'єднується на центральному рівні;

– пошук мінімального або максимального значення в масиві шляхом локального аналізу частин;

– обчислення середньоквадратичного відхилення;

– обчислення скалярного добутку векторів;

– щільне множення матриць.

Кожен з типів задач було переглянуто зі сторони обчислювальної складності, структури вхідних даних для цієї задачі, алгоритму для розподілу між вузлами і механізму зібрання кінцевого результату. Дані приклади показують, що система має підтримку функціональності не лише початкового рівня, а й більш складного

рівня. Система легко масштабується під більш складні задачі і при цьому не потрібно змінювати принципову логіку.

2.3 Реалізація самоорганізації в архітектурі системи

Для того, щоб система працювала правильно та ефективно важливо правильно змоделювати взаємодію між компонентами цієї системи. Щоб системи могла добре адаптуватися до змін у кількості вузлів, зростання навантаження та змін під час роботи системи є важливим використання та реалізація самоорганізації такої системи.

Кожна компонента підтримує постійний зв’язок з іншими компонентами. Це забезпечує обмін навантаженням вузлів (статусами вузлів) та швидкий обмін обчислювальних завдань та їх результатів. Такий зв’язок реалізується за допомогою технології WebSocket. Він дозволяє мати мінімальні затримки при передачі завдань і значно підвищує стабільність і відмовостійкість системи при умові динамічної зміни доступних ресурсів.

Кожен компонент системи відправляє свій статус всім іншим компонентам. Статус включає в себе завантаженість комп’ютера, а конкретніше завантаженість процесора та використання оперативної пам’яті. Статус являється важливою частиною самоорганізації системи, так як виходячи з нього визначається центр системи, та кількість задач, яку буде виконувати компонент і чи буде взагалі він їх виконувати. На основі цих даних формується уявлення про поточний стан всієї розподіленої децентралізованої системи.

Одним з найважливіших факторів ефективної розподіленої децентралізованої системи є можливість правильно балансувати навантаження. Використовуючи інформацію про статус (навантаження) кожної доступної компоненти, вузол, що виконує роль центра системи, обирає ефективну стратегію для розподілу завдання. На даний момент реалізовано стратегію на основі вибору найменшого навантаження, та не допуск найбільш завантажених вузлів до роботи

над завданнями. Дана стратегія є достатньо простою для впровадження і обслуговування системи, але при цьому всьому забезпечує достатньо високу ефективність у сценаріях змінної кількості вузлів системи і не стабільним навантаженням. В майбутньому системи може реалізувати більш складну стратегію розподілу завдань, наприклад розподіл за пріоритетами, складністю та критичністю поставлених завдань.

Зміщення центра працює наступним чином: компонента, що являється центром, перевіряє кожні 15 секунд чи її завантаженість не перевищує допустимий поріг. Допустимою завантаженістю головного процесора являється до 80%. Якщо центр перевищив допустиму норму, то починається процес зміщення центра. Центр обирає найменш навантажений компонент та робить його новим центром. Центр передає йому команду, яка почне маркувати компонент, як новий центр, а сам стирає дані, які вказують, що він центр та стає звичайною компонентою. Новий центр починає перевіряти свою завантаженість і чи не перевищує вона ліміт, та відправляє завдання іншим компонентам.

Завдання генеруються центром та розпаралелюються між компонентами у вигляді тексту, який потім ці компоненти зчитують, перетворюють на вираз та обчислюють його та повертають результат. Це дозволяє реалізувати прозорий, стандартний процес взаємодії між усіма компонентами розподіленої децентралізованої системи.

Завдання діляться між компонентами, що відсортовані за завантаженістю і найбільше завдань отримають ті, що найменш навантажені. Якщо компонент має завантаженість більшу ніж допустима (80%) використання центрального процесора, то такий компонент пропускається і не отримує задачі від нього, які потрібно обчислювати. Центр сам розсилає деяку кількість завдань через деяку кількість часу. По дефолту 30 виразів кожні 5 секунд. Але дані показники можуть змінюватися виходячи з типу задачі.

Однією з важливих переваг системи, що самоорганізовується, також є автоматичне відновлення після збоїв. Якщо один з вузлів системи відключається,

то система миттєво оновлює список активних компонентів системи і відключає пошкоджений вузол, що дозволяє більше не розподіляти завдання до даного вузла. Система являється дуже доступною та відмовостійкою без потреби зовнішнього втручання адміністратора.

Варто зазначити, що адміністратор даної системи має можливість втрутитися в неї вручну, щоб скоригувати роботу системи, що є самоорганізованою. Він може вручну обрати типи виконуваних завдань, їх перерозподілення, примусово змінити центр та відключити вузли від системи. Таким чином адміністратор має можливість налаштувати систему під конкретне середовище або поставлені задачі. Це забезпечує можливість гнучкості даної системи під різні випадки.

2.4 Висновки до другого розділу

У другому розділі було проведено проєктування архітектури для розподіленої універсальної системи для виконання розпаралелювання завдань. В результаті аналізу була сформована основна логіка побудови та взаємодії між елементами системи, щоб забезпечити гнучкість та масштабованість.

Основною особливістю проєктованої системи є реалізація керованої децентралізованої архітектури (P2P), у якій усі вузли мають рівноправні можливості, але на деякий час один з цих вузлів виконує роль центру – координатор розподілу задач та той хто збирає їх результати. Особливим є те, що центр не є постійною структурою (якщо центр перевантажено, то він змінюється). Це дозволяє досягнути рівномірного розподілу навантаження та уникнення вузького місця системи.

Компоненти самостійно передають інформацію про стан завантаженості системи іншим її учасникам з допомогою WebSocket-з’єднання. Це створює можливість для взаємного моніторингу. Цю інформацію центр використовує для прийняття рішень, використовуючи принцип мінімізації навантаження.

У цьому розділі також було описано архітектуру адміністративного веб-сайту. Ця архітектура реалізована за шаблоном MVC (Model-View-Controller) з використанням ASP.NET Core. Дані параметри дозволяють розділити логіку з управління даними, їхнє представлення та взаємодію з користувачем. Адміністратор має можливість бачити інформацію про стан системи, змінювати центр, виключати компоненти, обирати типи виконуваних завдань.

Було побудовано та розглянуто діаграму класів, яка відобразила структуру системи для компоненти та адміністративного сайту. Також було побудовано діаграму прецедентів. Дана діаграма описала основні функції кожного учасника даної системи. Ключовими ролями є адміністратор, центральна компонента і виконавча компонента. Кожна частина системи має свої функціональні обов’язки, що полегшує розуміння логіки роботи і можливостей системи для розробника та для кінцевого користувача.

АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РОЗПОДІЛЕНОЇ УНІВЕРСАЛЬНОЇ СИСТЕМИ

3.1 Основні алгоритми системи

Основні алгоритми, закладені у функціонування запропонованої системи, спрямовані на забезпечення самоорганізації, адаптивного балансування навантаження, відмовостійкості та гнучкого управління. Архітектура системи поєднує децентралізовану модель взаємодії з централізованим адмініструванням і побудована таким чином, щоб кожен з вузлів міг брати участь у прийнятті рішень, зберігати копії даних і самостійно виконувати завдання.

Система складається з двох головних логічних частин: компонента (WPF-додаток), що розгортається на кожному вузлі, виконує обчислення, взаємодіє з іншими вузлами та адміністративний веб-сайт (ASP.NET Core MVC), що працює поруч з однією з компонент і забезпечує централізовану візуалізацію стану системи та ручне управління. Усі взаємодії в системі відбуваються за допомогою WebSocket-з’єднань, а дані передаються у вигляді JSON-структур. Кожна компонента зберігає інформацію про інших учасників системи у своїй локальній базі, що дозволяє зберігати автономність і зменшує залежність від одного вузла. Усі ці процеси супроводжуються частинами діаграмами активності, що показана у Додатку Б.

Ключовими алгоритмами в системі є:

– самореєстрація вузлів у мережі;
– обмін статусами та моніторинг навантаження;
– динамічне обрання центра системи;
– ручна зміна центра адміністратором;
– відключення вузла із системи;
– прийом, розподіл та виконання задач (обчислень);
– відновлення задач при зміні центра.

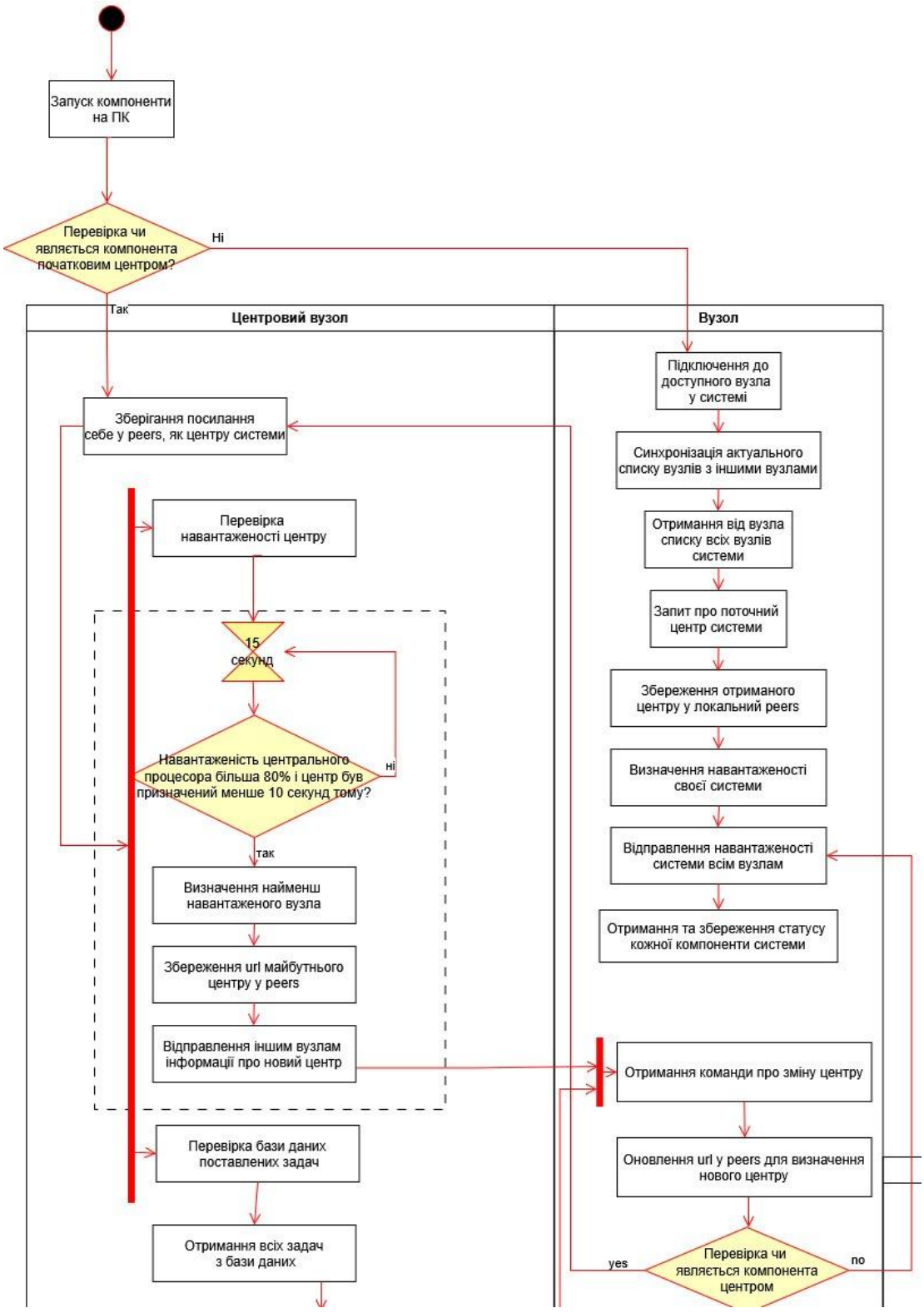


Рисунок 3.1 – Діаграма активності. Частина визначення та перенесення центру системи

Самореєстрація вузлів у мережі(відображена на Рисунку 3.1) функціонує наступним чином. Коли нова компонента з'являється у системі відбувається наступне: спочатку вузол перевіряє чи він являється початковим центром. Якщо він являється центром, то входить в стан центрового вузла, якщо не являється, то він продовжує роботу у стані звичайного вузла.

Якщо вузол є центровим, то він зберігає своє посилання у peers, як поточний центр системи та у списку вузлів що вже є у системі. Після цього він запускає таймер перевірки завантаженості центру. Кожних 15 секунд спрацьовує наступний алгоритм:

– спочатку перевіряється чи центр був встановлений вручну. Якщо так, то пропускається 1 перевірка для зміни центру;

– далі перевіряється коли був встановлений центр, якщо це було менше 10 секунд тому, то пропускається перевірка для зміни центру;

– після цього збирається стан системи поточного вузла, тобто його завантаженість;

– далі йде перевірка чи завантаженість системи є допустимою, тобто менше 80% завантаженості центрального процесора;

– якщо завантаженість допустима, то центр не змінюється і алгоритм зупиняється до наступної перевірки;

– якщо завантаженість більше допустимої, то запускається алгоритм зміни центру;

– при зміні центру теперішній центр отримує вузол, який являється найменш завантаженим. Якщо такий вузол знайдено, то він зберігається як новий центр у локальній базі. Після цього минулий центр зупиняє таймер перевірки центру та повідомляє всі інші вузли про новий центр;

– при отриманні такого сигналу, кожен вузол перевіряє чи являється він новим центром;

– якщо вузол не являється новим центром, то він просто оновлює дані в локальному peers. Він зберігає посилання нового центр;

- якщо вузол являється новим центром, то спочатку вузол зберігає час його встановлення та починає таймер перевірки центру.

Якщо вузол є звичайним, то виконується наступний алгоритм дій:

– вузол підключається до доступного вузла, беручи інформацію peers;

– вузол зберігає своє посилання у peers у списку вузлів, що вже є у системі;

– після цього проводиться синхронізація списку вузлів з іншими вузлами. При синхронізації вузли додають посилання нового вузла у свій peers;

– після цього вузол запитує про поточний центр системи в іншого, та зберігає цю інформацію у свій peers;

– далі він запускає таймер відправки завантаженості всім вузлам своєї системи кожних 5 секунд;

– вузол також отримує завантаженість інших вузлів та зберігає цю інформацію та відображає її.

Адміністративний веб-сайт у свою чергу запускається на електронній обчислювальній машині поряд з компонентною (як це зображено на рисунку 3.2). Всю інформацію він отримує від даної компоненти. Веб-сайт встановлює WebSocket з’єднання з компонентною та відправляє запити до неї по таймеру (кожні 5 секунд) щоб мати актуальну інформацію про систему. Компонента надає йому наступні дані з своєї локальної бази даних, які веб-додаток потім відображає для користувача:

– інформацію про всі вузли що функціонують у системі, актуальну завантаженість вузлів у вигляді завантаженості головного процесора у відсотках та оперативної пам’яті у мегабайтах. Також він запитує інформацію скільки в кожного вузла оперативної пам’яті, щоб правильно відобразити графік завантаженості для користувача;

– інформація про те, хто являється поточним вузлом. Дана інформація відображається на карточці вузла, для того, щоб наочно побачити, яка є завантаженість цього вузла і як вона змінюється.

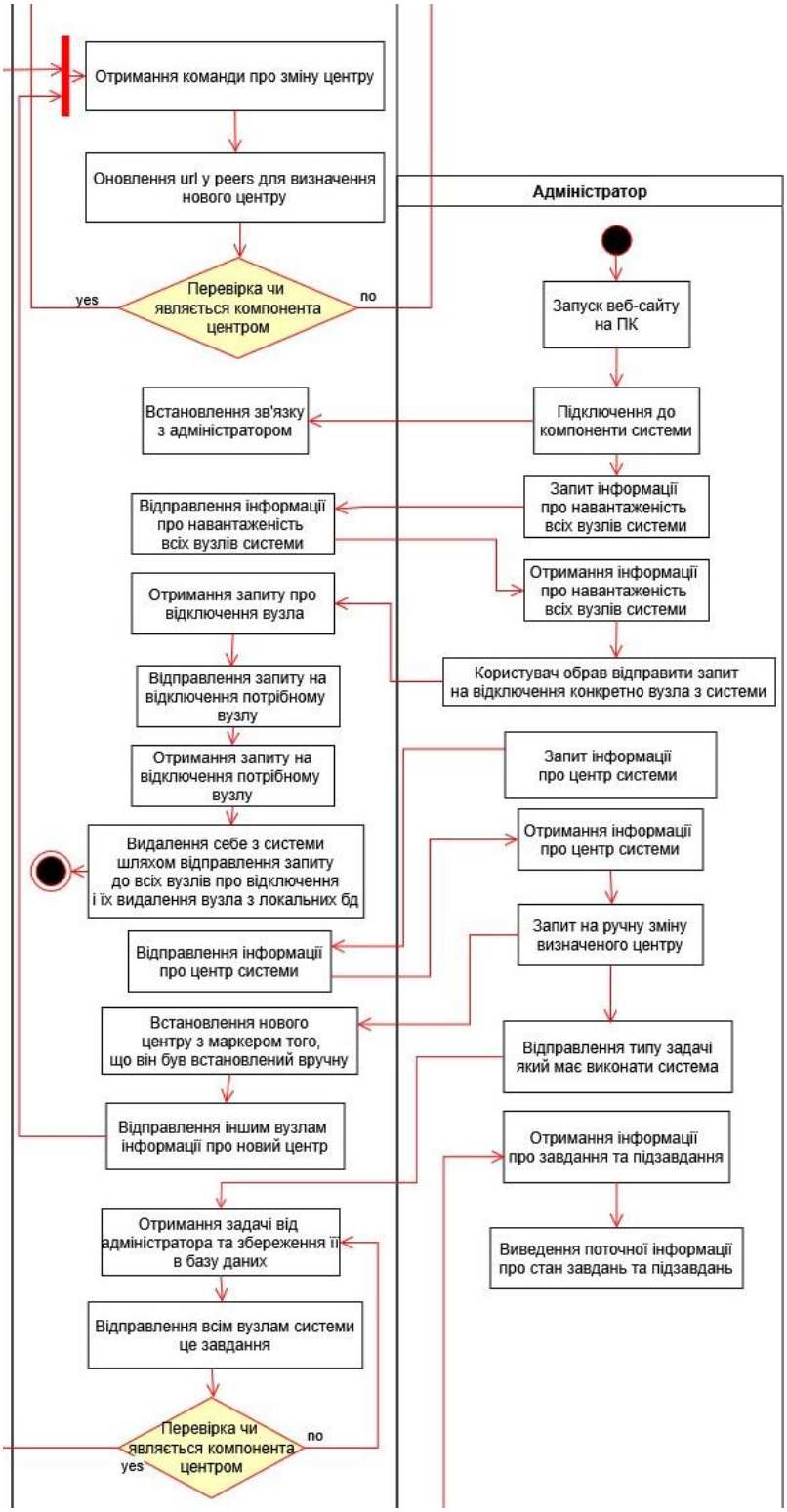


Рисунок 3.2 – Діаграма активності. Частина дій між адміністративним веб-сайтом та компонентою

Всі вузли відображені в інтерфейсі у вигляді карточок. Кожна карточка відображає графіки з завантаженістю оперативної пам’яті та центрального

процесора, назву електронної обчислювальної машини, посилання по якому до вузла підключаються і також вузол який являється центром, має іконку центра зверху, щоб його можна було легко відрізнити від інших. Також карточка має кнопки “Зробити центром” і “Відключити вузол”. Вони дають можливості управляти системою вручну.

При натисненні кнопки “Зробити центром” запускається алгоритм ручної зміни центру. Він складається з наступних дій:

– вузол отримує від адміністративного веб-сайту запит на ручну зміну центру та ідентифікатор потрібного вузла;

– отриманий вузол встановлює маркер, що новий центр був отриманий вручну;

– після цього він зберігає новий центр у локальний peers;

– далі вузол відправляє всім іншим вузлам команду на оновлення центру та посилання на новий центр;

– далі кожен вузол зупиняє таймер перевірки задач та центру і оновлює інформацію про новий центр, попередній центр зупиняється;

– далі кожен вузол перевіряє чи являється він новим центром;

– якщо вузол є новим центром, то він починає таймер перевірки завантаженості центру та перевіряє список задач і статус їх виконання, та запускає не виконані задачі або збирає результат виконаних.

При натисненні кнопки “ Відключити вузол” запускається алгоритм ручного відключення вузла від системи. Він складається з наступних дій:

– вузол отримує від адміністративного веб сайту запит на вузла, та ідентифікатор потрібного вузла;

– отриманий вузол перевіряє чи є він вузлом який потрібно відключити;

– якщо поточний вузол є вузлом який має бути відключеним, то він відправляє всім вузлам у системі, що його потрібно видали з локального списку активних вузлів в системі peers;

– всі вузли закривають з’єднання з видаленим вузлом;

– далі вузол зупиняє таймери для відправки статусу всім іншим вузлам та синхронізації між вузлами і закриває додаток на електронній обчислювальній машині;

– якщо поточний вузол не є вузлом, який має бути відключеним, то він відправляє потрібному вузлу команду, що він має відключитися від системи. Вузол отримує цю команду та проводить ті ж самі дії, що у і випадку коли відключений вузол був той, хто отримав початковий запит від адміністратора.

Також у веб-інтерфейсі для користувача відображаються типи задач, які можна запустити у систему. Для того, щоб почати процес обчислення задачі, спрацьовує наступний алгоритм (Рисунок 3.3):

– користувач обирає тип задачі та відправляє його до компоненти;

– компонента отримує задачу від адміністратора та зберігає її у свою локальну базу даних;

– після цього завдання відправляється всім вузлам у системі, щоб вони також зберегли його в своїй локальній базі даних з статусом “Зробити”;

– також кожна компонента перевіряє чи являється вона центром. Якщо так, то центр змінює статус задачі на “В прогресі”, зберігає оновлення та відправляє це усім вузлам, щоб вони також оновили свій статус в локальній базі і після цього починається процес розподілу задачі по типу;

– спочатку система аналізує до якого типу відноситься задача і як правильно її розподілити на підзадачі. Після цього вибираються вузли системи, що мають допустиме навантаження і не є дуже завантаженими. Завантаженість центрального процесора має бути менша ніж 80% та в оперативній пам’яті має бути завантажено менше ніж 1 гігабайт;

– після цього підзадачі розподіляються згідно з кількістю оптимальних вузлів та відправляються у вигляді json об’єкту з ідентифікатором задачі і виразом, що є строкою;

– вузол отримує підзадачу від центру, розбиває строку на вираз, обчислює даний вираз, зберігає результат в базу даних та відправляє результат підзадачі всім вузлам, щоб вони його зберегли у локальну базу;

– коли центр отримує результат підзадачі, то він перевіряє чи всі підзадачі для задачі були отримані. Якщо ні, то він нічого не робить. Якщо так, то всі проміжні результати обраховуються в кінцевий результат. Після цього результат зберігається, а статус задачі змінюється на “Зроблено”. Нові дані зберігаються у локальну базу даних та відправляються всім іншим вузлам, щоб вони також могли оновити дані по задачі;

– вузол відправляє результат задачі до адміністративного веб-сайту і він своєю чергою відображає результати про поточний стан задач для користувача. Також він виводить проміжні результати, що обчислювали вузли та інформацію хто їх виконав та коли.

При зміні центру під час виконання завдання важливо звернути увагу на те, щоб він міг продовжити виконувати вже почату задачу та почати виконувати нові, якщо такі є.

Для цього в системі передбачений наступний алгоритм, що показаний на Рисунку 3.4:

– компонент отримує з локальної бази даних список всіх задач, що не готові. Після цього кожна задача перевіряється чи має вона статус “Не розпочата” або “В прогресі”.

– якщо задача має статус “Не розпочата”, то встановлюється статус, що вона “В прогресі” та розподіляється між вузлами, як було описано вище;

– якщо задача має статус “В прогресі”, то перевіряється чи всі проміжні результати були отримані. Якщо так, то обраховується кінцевий результат, якщо ні, то центр продовжує чекати.

Це дозволяє не губити задачі і бути впевненим, що всі задачі будуть виконані.

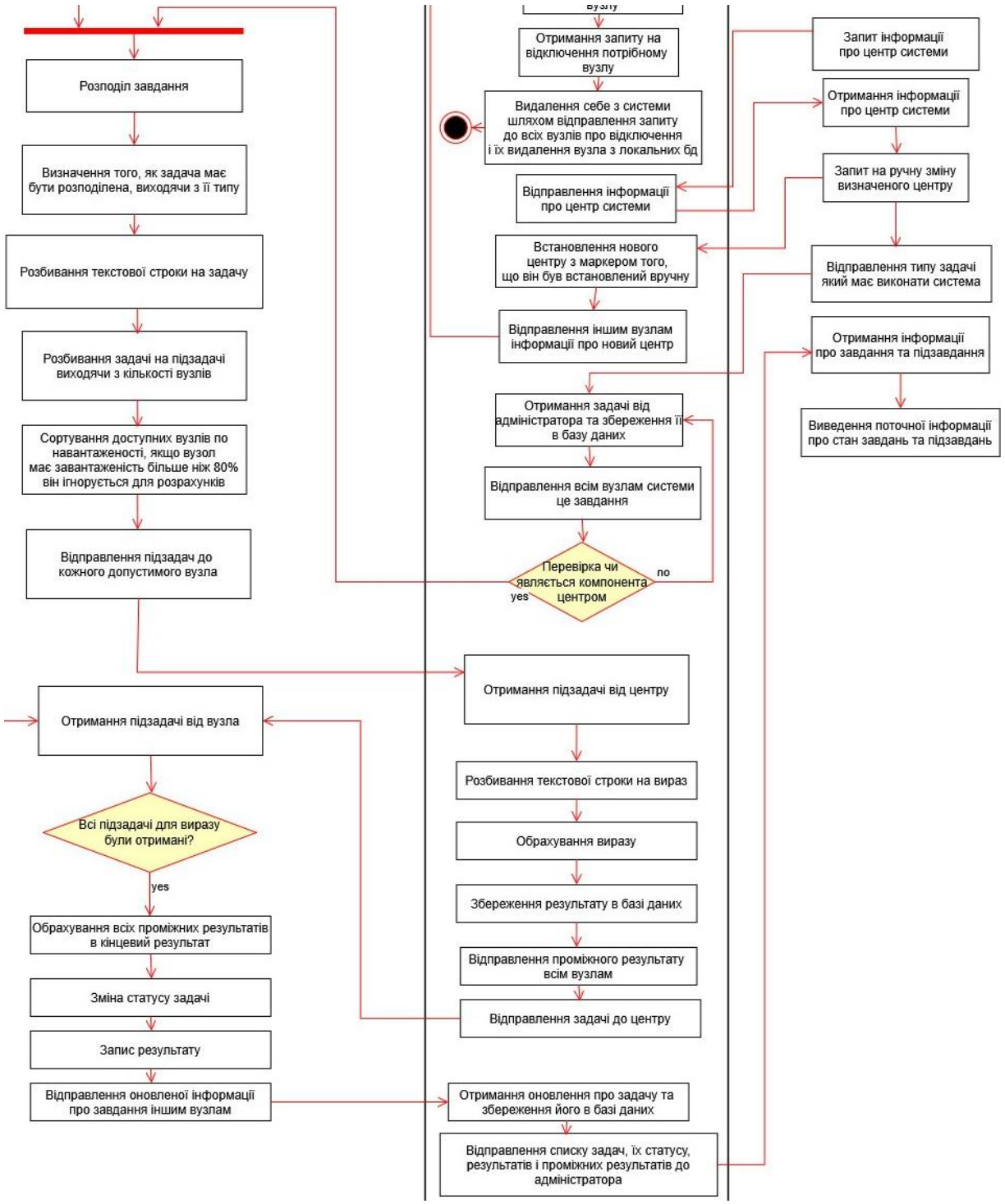


Рисунок 3.3 – Діаграма активності. Частина дій, що відповідають за відправку задач від адміністративного веб сайту до компоненти

Рисунок 3.4 – Діаграма активності. Частина дій, що відповідають за відправку задач від адміністративного веб сайту до компоненти

Тобто, основні алгоритми, реалізовані в системі, дозволяють повністю забезпечити її функціональність у децентралізованому середовищі. Система є самоорганізованою, гнучкою, стійкою до збоїв і при цьому зручною для адміністрування. Алгоритмічне забезпечення дозволяє реалізувати повноцінне управління розподіленим середовищем без втрати задач, із підтримкою як автономної, так і ручної координації.

3.2 Структура та склад програмного забезпечення

Структура створеного програмного забезпечення була відображена у діаграмі компонентів, яку можна знайти у додатку В.

Основними компонентами діаграми є “Component” та “Web Site”, що відображають вузол системи та адміністративний веб-сайт.

Для вузла системи в діаграмі є такі основні компоненти: модель та сервіси. Компонента моделі має в себе моделі задач та підзадач, та модель статусу системи. Моделі задач та підзадач, отримують інформацію з репозиторію, який у свою чергу бере інформацію з бази даних.

Компонента сервісів відповідає за основну логіку додатку. Вона містить такі компоненти, як сервіс статусів, сервіс вузлів та сервіс задач.

Сервіс статусів системи відповідає за збір інформації завантаженості системи у вигляді моделі для статусу системи. Також сервіс відповідає за перевірку завантаженості центра і логіку зміни її при перевищенні завантаженості вузла.

Сервіс вузлів є основною керуючою компонентою мережі. Він відповідає за з’єднання вузлів між собою, отримання інформації від них, відправку інформації до них, відправку сигналів про дії, які потрібно виконати. Сервіс вузлів отримує використання структури даних від моделі задач, а інформацію про вузли системи він отримує від компоненти сховища.

Сервіс задач займається основною логікою виконання та розподілення задач на підзадачі. Отримує дані з компоненти моделей задач та підзадач.

Компонента помічник отримує інформацію від усіх інших компонентів системи для виведення інформації в файл логів системи. Також ця компонента отримує інформацію про локальну адресу вузла і передає її для використання у системі.

Для адміністративного веб-сайту в діаграмі є такі основні компоненти: модель, контролер, представлення, сервіси.

У моделі міститься статус системи і моделі задач та пізадач. Вони виконують ті ж функції, що і в компоненті вузла.

У контролері є компонента, що відповідає за авторизацію та аутентифікацію, та контролер мережі, що відповідає за зв’язок між користувачем та логікою разом з даними. Контролер мережі бере інформацію, чи може користувач мати доступ до неї у компоненти авторизації. Також інформацію з вузла він бере використовуючи компоненту хабу, який у свою чергу отримує її з хабу вузлів.

Компонента представлення має в себе компоненту логіну, що обмінюється інформацією з компонентою авторизації. Також представлення має в себе відображення статусу навантаженості вузлів; задач, що виконуються у системі та підзадач. Останні три компоненти спілкуються з контролером мережі.

Склад створеного програмного забезпечення був відображений у діаграмі розгортання, яку можна на рисунку 3.5.

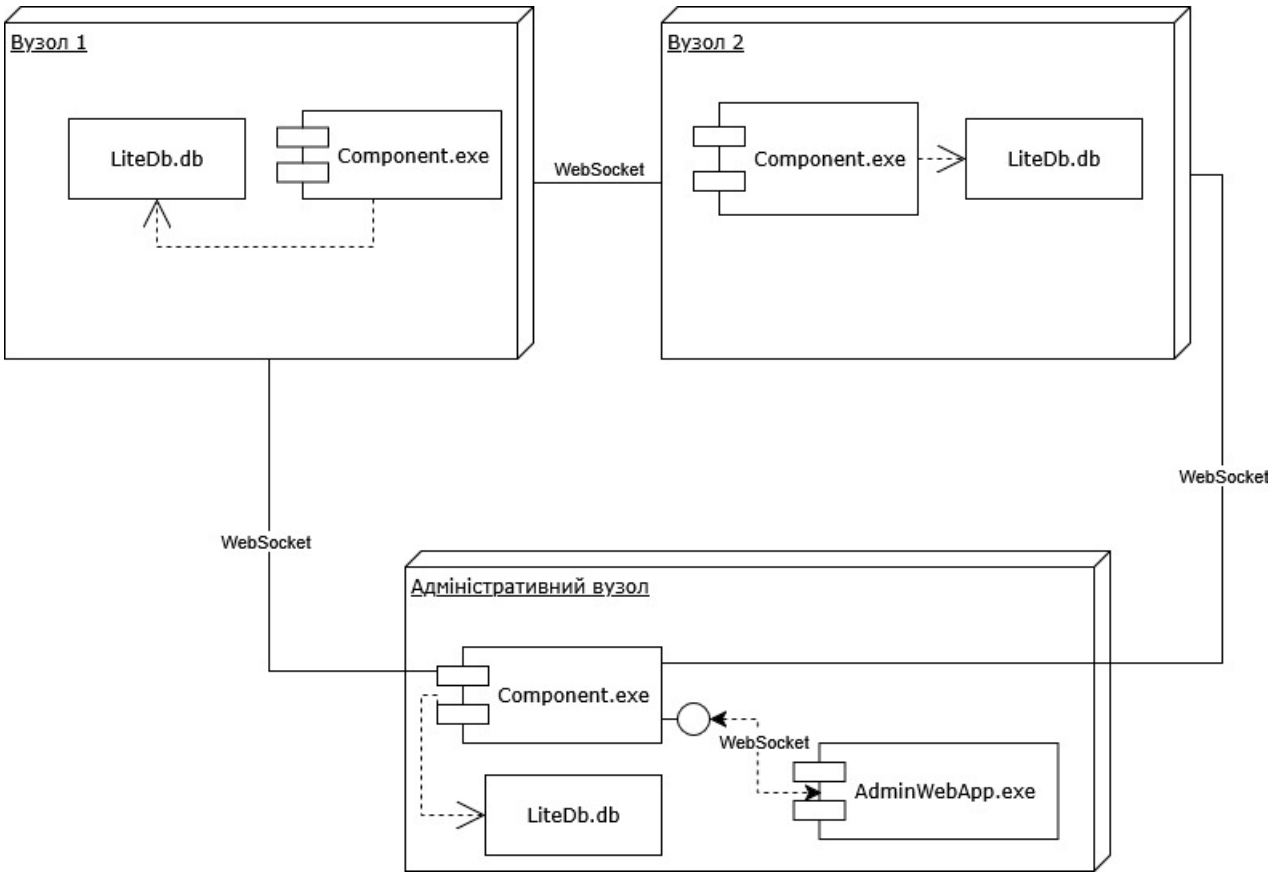


Рисунок 3.5 – Діаграма розгортання

Кожен вузол – це електронний обчислювальний пристрій. На даній діаграмі “Вузол” 1 та “Вузол 2” є звичайними учасниками системи, а “Адміністративний вузол” поєднує в собі компоненту та веб-сайт.

На всіх вузлах встановлено WPF додаток та локальна база даних, в якій цей додаток зберігає всю необхідну інформацію про систему. Також кожен вузол має WebSocket з’єднання з іншими вузлами.

Адміністративний вузол додатково має в себе веб-додаток ASP.NET Core MVC. Він також має WebSocket з’єднання з вузлом, що знаходиться на тій же електронній обчислювальній машині, що і він.

Дана діаграма також відображає симетрію вузлів, тобто всі мають однакові можливості, окрім вузла, що має адміністративний веб-додаток.

Вимоги для обчислювальних вузлів відображені в Таблиці 1.1.

Таблиця 1.1 – Вимоги для обчислювальних вузлів

№	Параметр	Мінімум	Рекомендовано
1.	Операційна система	Windows 10 або вище	Windows 10/11 Pro
2.	Процесор	2 ядра, 1.5 ГГц	4 ядра, 2.4+ ГГц
3	Оперативна пам’ять	2 ГБ	4–8 ГБ
4	Вільне місце на диску	500 МБ	1+ ГБ
5	.NET Runtime	.NET 6/7	Остання стабільна версія .NET Core
6	Додатково	Порт TCP відкритий для WebSocket	Стабільне LAN або Wi-Fi з’єднання

Якщо вузол має також бути адміністративним, тобто на електронно обчислювальну машину буде також встановлений адміністративний веб-сайт, то для вузла будуть такі додаткові вимоги:

– встановлений веб-сервер ASP.NET Core;

– встановлений браузер (для відкриття локального інтерфейсу);

– встановлено IIS Express або kestrel-сервер;

– бажано: 8+ ГБ RAM, SSD.

Сформовані вимоги є помірними, що робить можливим розгортання системи навіть у лабораторному або навчальному середовищі.

3.3 Реалізація веб-базованого інтерфейсу для взаємодії з системою

У межах створеної децентралізованої обчислювальної системи реалізовано веб-базований інтерфейс, що дозволяє адміністраторам здійснювати моніторинг стану вузлів, перегляд результатів обчислень, запуск задач та керування центром системи. Даний інтерфейс виступає в ролі додаткового, але не обов’язкового, засобу керування, що функціонує поверх самоорганізованої мережі компонент.

Веб-інтерфейс побудований на основі:

– фреймворку ASP.NET Core MVC для побудови серверної логіки веб-інтефейсу;

– фреймворку Bootstrap для реалізації адаптивного та гнучкого користувацького інтерфейсу;

– бібліотеки SignalR для встановлення прямого двостороннього зв’язку між веб-додатком і виконавчою компонентою;

– формату даних JSON – як основний формат обміну повідомленнями між веб-сайтом і системою.

Рисунок 3.6 – Макет адміністративного інтерфейсу веб-додатку. Вхід у систему

Веб-сайт захищений авторизацією і автентифікацією та є закритим і не має публічної частини. Спочатку адміністратор має увійти до системи використовуючи свій логін і пароль. Інтерфейс входу в систему можна побачити на рисунку 3.6.

Якщо логін та пароль були правильними, то користувач переходить на головну сторінку веб-сайту. Макет цієї сторінки представлений на рисунку 3.7.

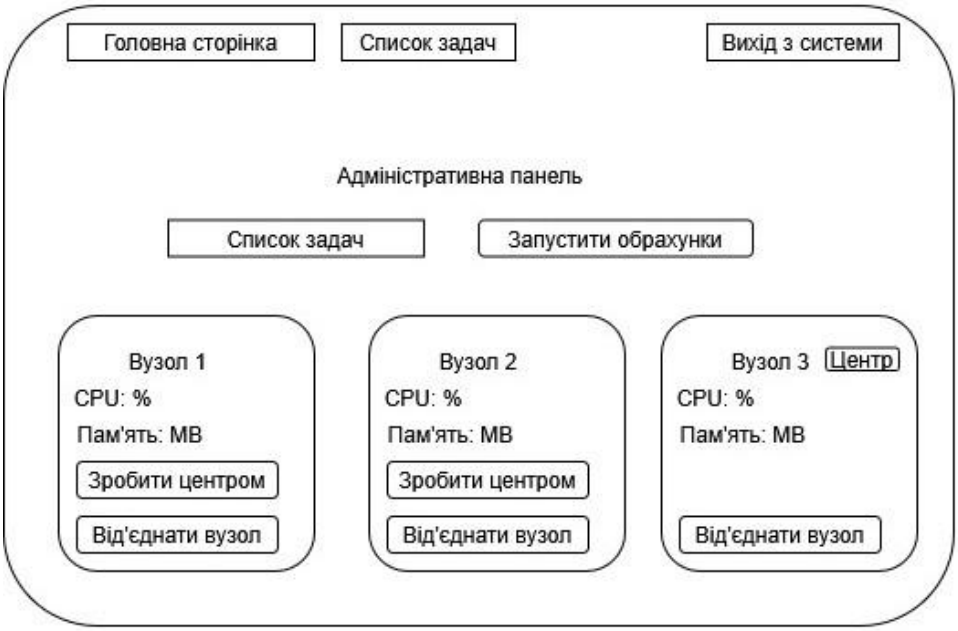


Рисунок 3.7 – Макет адміністративного інтерфейсу веб-додатку. Головна сторінка

На головній сторінці користувач має наступну функціональність:

– перегляд списку активних вузлів у вигляді карточок. У даному списку він може побачити актуальне навантаження кожного з вузлів, час останнього оновлення, ідентифікатор електронної-обчислювальної машини та посилання даного вузла по якому всі інші вузли з ним спілкуються. Також один з вузлів відображає поточний центр спеціальним маркером;

– можливість керувати центром системи вручну. Кожна карточка (окрім поточного центру) містить кнопку, яка дозволяє призначити новий центр мережі. При натисненні кнопки користувач бачить, що маркер центру перейшов з минулого центру на вказаний;

– можливість виключити вузол мережі. Кожна карточка містить кнопку, яка дозволяє виключити вузол мережі. При натисненні її користувач бачить, що вузол зник зі списку – це означає вузол більше не є активним;

– можливість обрати тип задачі у списку типів задач та відправити її у мережу для розподілення між вузлами та швидкого і ефективного її обрахування.

Використовуючи навігатор сайту користувач може перейти до списку задач, що були виконані чи виконуються мережею. Інтерфейс сторінки статусів задач відображений на рисунку 3.8.

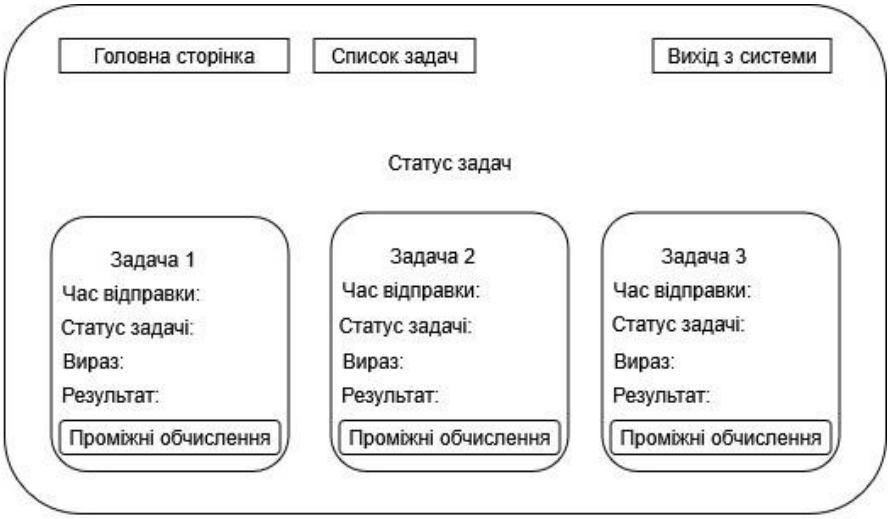


Рисунок 3.8 – Макет адміністративного інтерфейсу веб-додатку. Сторінка статусів задач

Сторінка статусів задач відображає задачі в системі у вигляді карточок. Кожна з них містить інформацію про час відправки задачі у мережу, статус цієї задачі, вираз, який обчислюється та результат цієї задачі, якщо такий наявний. Також кожна з цих карточок містить кнопку, що дозволяє переглянути проміжні обчислення, які обчислюються або обчислювались вузлами в мережі. При натисканні на цю кнопку користувач переходить на сторінку відображення проміжних обчислень для конкретної задачі. Інтерфейс сторінки проміжних обчислень можна побачити на рисунку 3.9.

Рисунок 3.9 – Макет адміністративного інтерфейсу веб-додатку. Сторінка проміжних обчислень

Сторінка проміжних обчислень відображає список обчислень у формі таблиці. Заголовок сторінки містить інформацію про час відправки задачі у мережу, статус цієї задачі, вираз який обчислюється та результат цієї задачі, якщо такий наявний. У таблиці міститься інформацію про кожне проміжне обчислення. У

таблицю входить вираз проміжного обчислення, ідентифікатор вузла, що її вирішував, результат виконання обчислення.

Також веб-інтерфейс містить кнопку виходу з системи. Після цього, щоб отримати доступ у систему, адміністратор має знову ввести логін та пароль.

3.4 Приклади застосування системи

Робота розподіленої системи починається з запуску вузлів на різних електронних обчислюваних машинах. Якщо вузол є центральним, то в програмі виводиться надпис “Центральний вузол” поряд з ідентифікатором електронної обчислювальної машини, як можна побачити на рисунку 3.10.

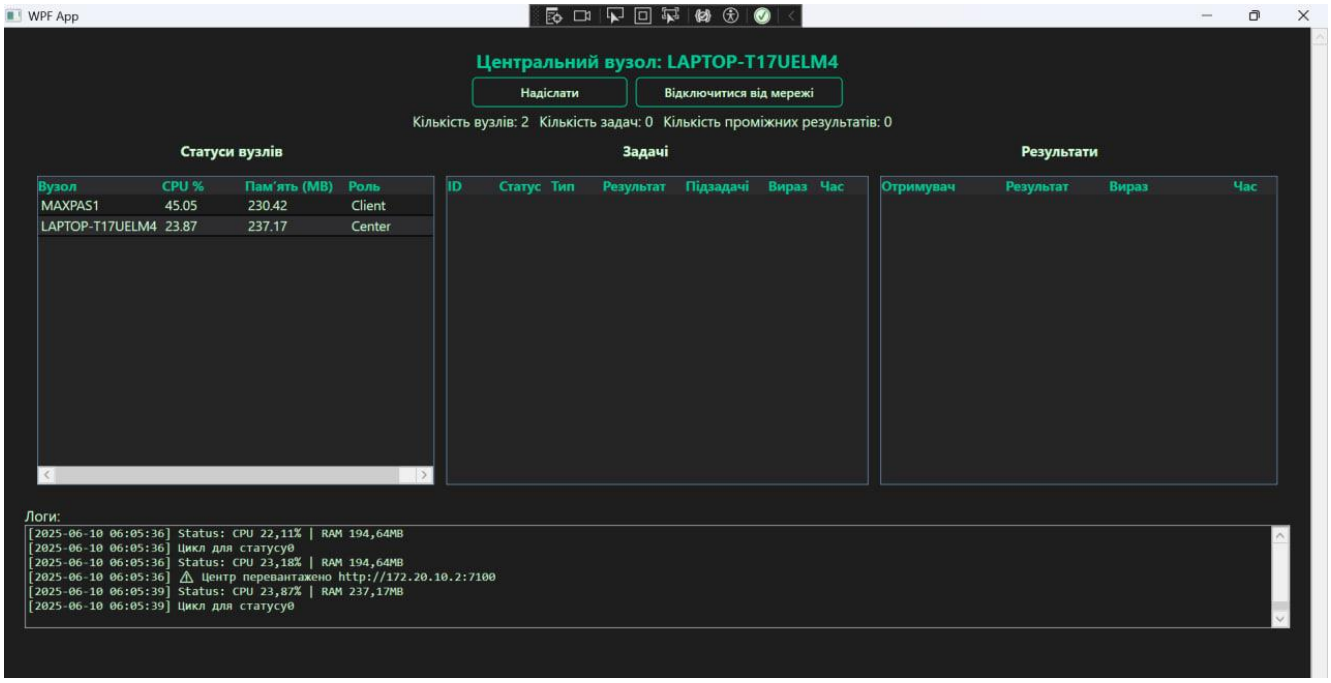


Рисунок 3.10 – Запуск вузла. Центральний вузол

Якщо вузол не є центральним, то виводиться надпис “Вузол”, як це можна побачити на рисунку 3.11. Програма виводить три таблиці: статуси вузлів, задачі, результати. В статусах вузлів можна побачити навантаження центрального процесора в відсотках, оперативну пам’ять в мегабайтах та роль вузла про те чи є

на цей час конкретний вузол роль центра чи клієнта системи. Також виводиться зверху кількість вузлів, що є у системі, кількість задач та кількість проміжних результатів.

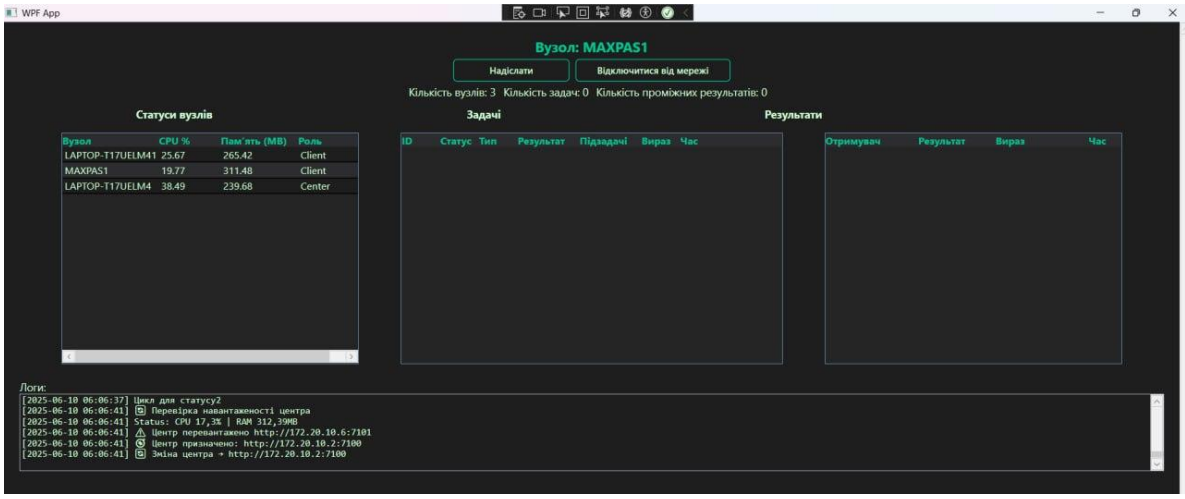


Рисунок 3.11 – Запуск вузла. Вузол який не є центром

Запустити задачу для виконання в мережі може тільки адміністратор. Для цього адміністративний веб-сайт має бути запущений поряд з вузлом розподіленої системи. Щоб отримати доступ до керування розподіленою системою адміністратор повинен авторизуватися. Для цього він має ввести свій логін та пароль на сторінці, що зображена на рисунку 3.12.

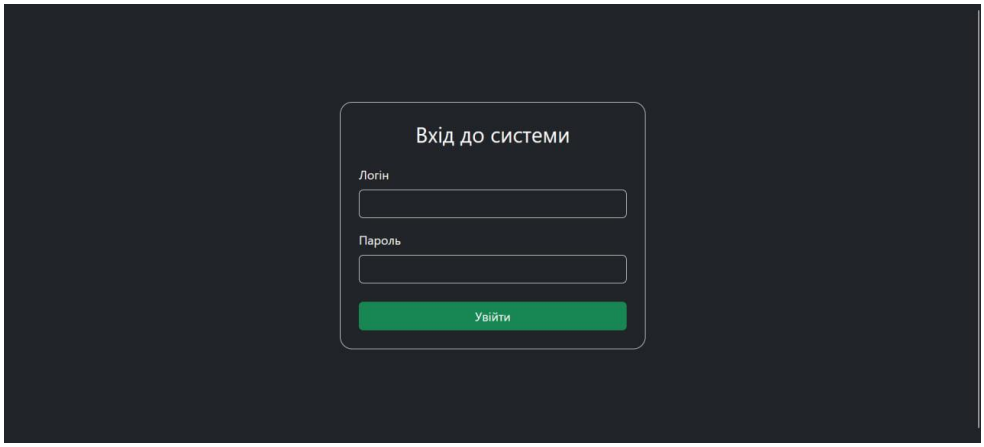


Рисунок 3.12 – Сторінка авторизації адміністратора

При успішній авторизації користувач переходить на сторінку статусів вузлів системи. На сторінці, що зображено на рисунку 3.13, виведений список вузлів, що активні в мережі. Кожний вузол показує свій ідентифікатор, посилання, завантаженість головного процесора та оперативної пам’яті у вигляді графіків, що показують відсоток завантаженості від загальної кількості. Вузол, який є центром, має маркер, який дозволяє легко його відрізнити від всіх інших. Також кожна карточка має кнопки “Зробити центром” та “Від’єднати вузол”. При від’єднанні вузла програмний вузол, який було від’єднано закривається на електронній обчислювальній машині.

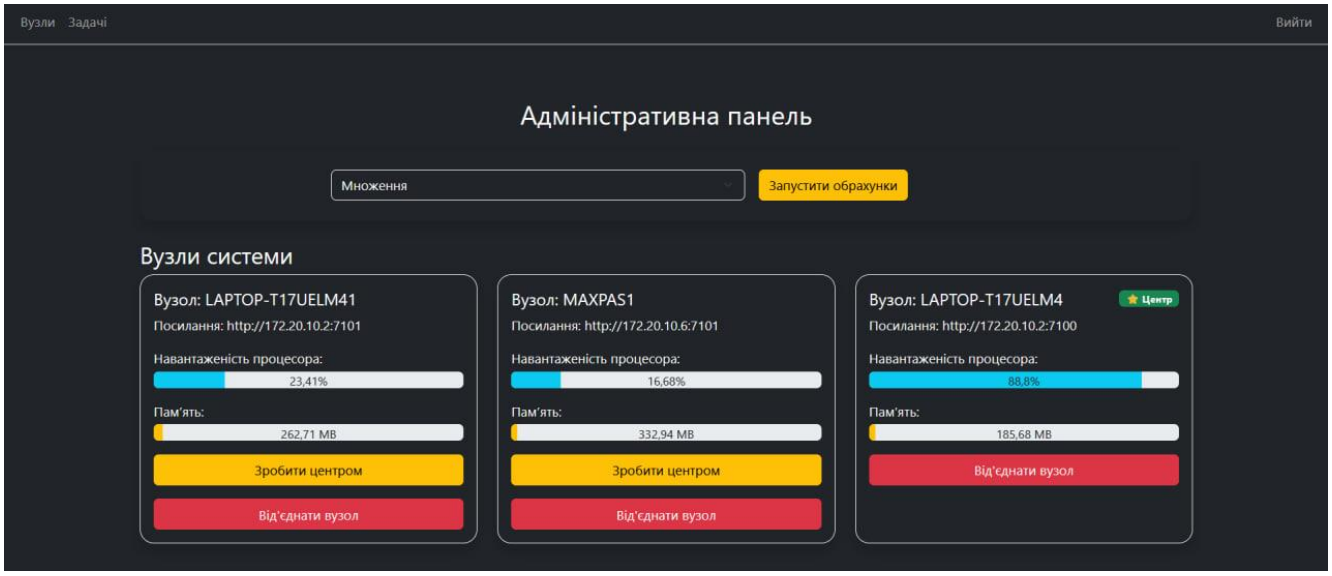


Рисунок 3.13 – Сторінка авторизації адміністратора

При натисненні на кнопку “Зробити центром” для середнього вузла він став центром вручну. Результат відображений на рисунку 3.14.

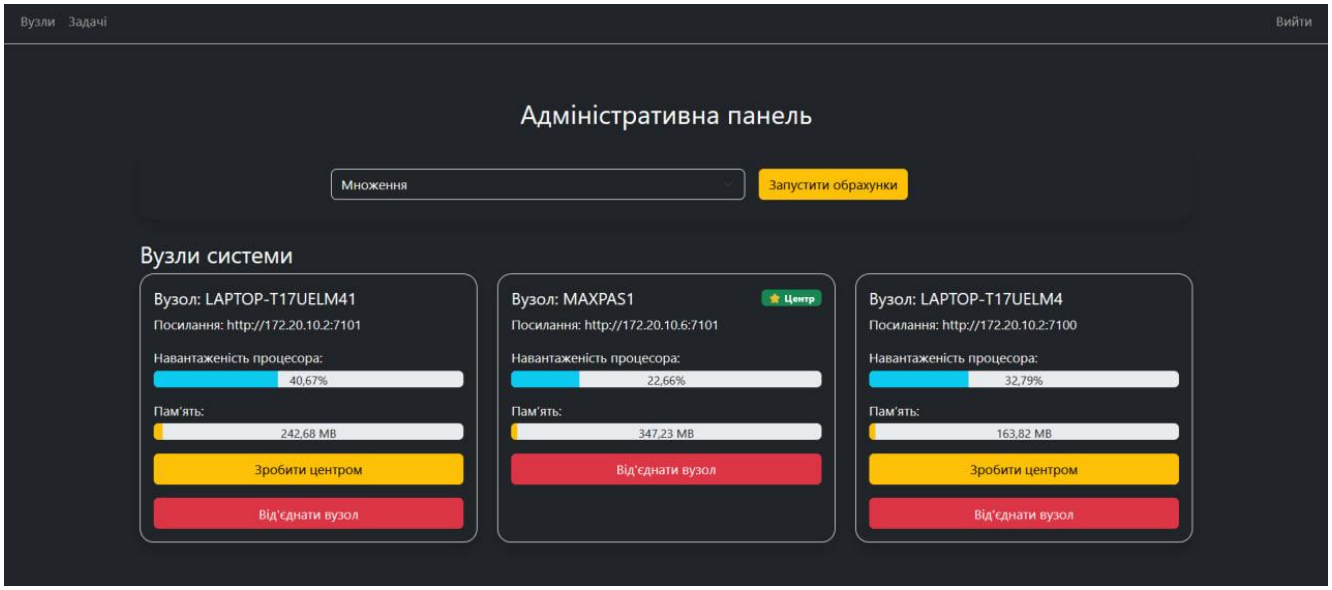


Рисунок 3.14 – Зміна центру вручну

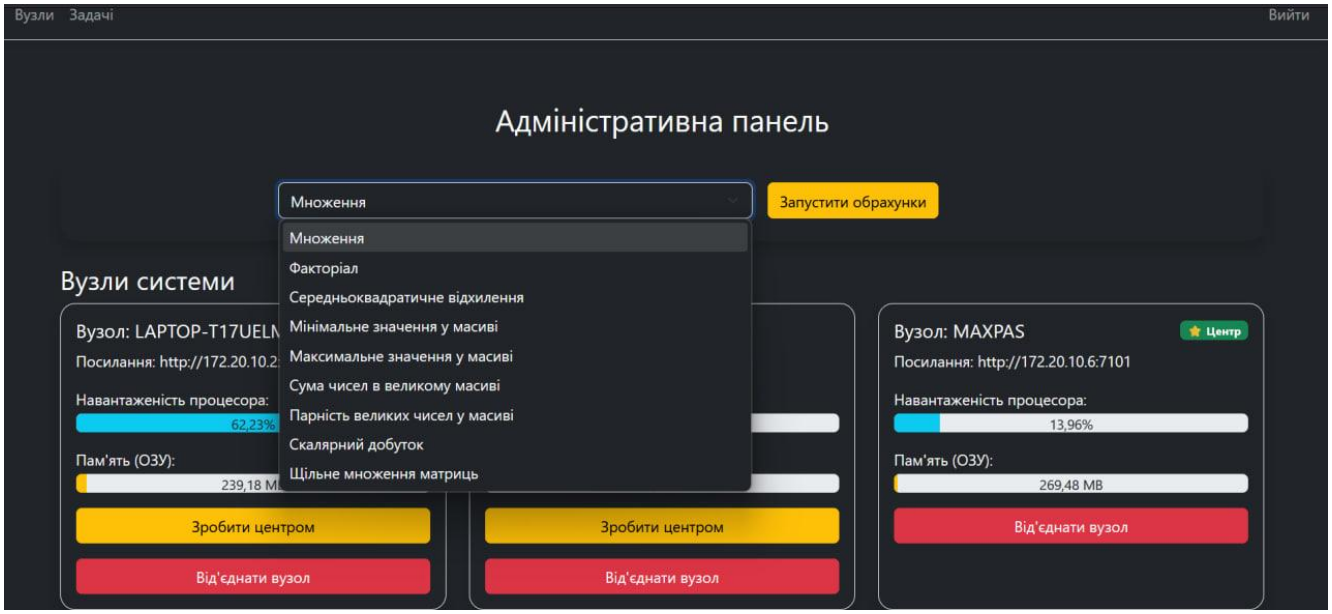


Рисунок 3.15 – Запуск задачі у розподілену систему

Також на даній сторінці можна запустити обрахунки, як це можна побачити на рисунку 3.15. Щоб обрати тип обрахунків є спадний список. Після обрання типу задачі “Множення великих чисел” користувач натискає кнопку “Запустити обрахунки” і задача запускається в систему.

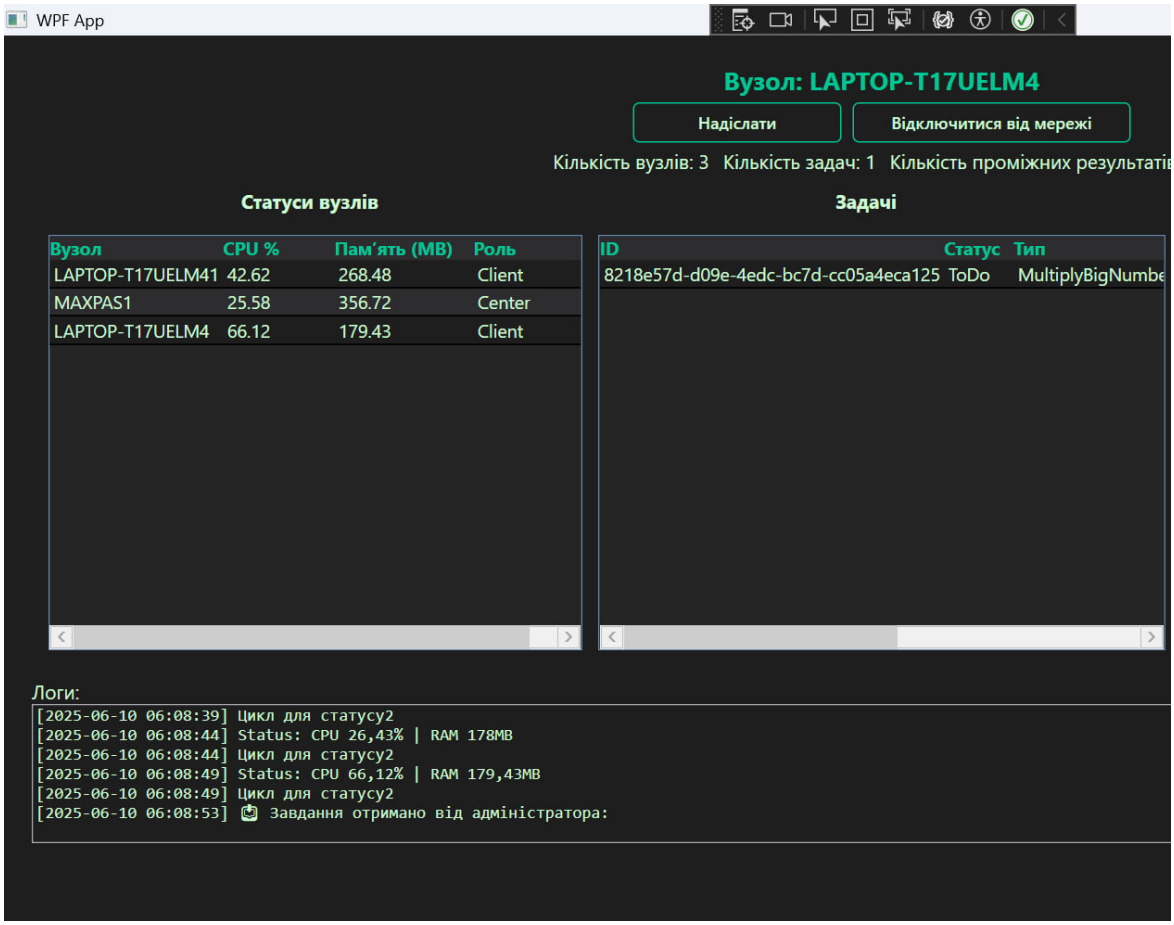


Рисунок 3.16 – Запуск задачі в систему. Перегляд з вузла

Сайт відправляє задачу вузлу, як це можна побачити на рисунку 3.16, той записує собі її з статусом ToDo та ділиться зі всіма іншими вузлами, які записують задачу в базу даних. Після того, як центр її отримав, то він починає процес розподілення задачі на підзадачі та відправку їх до інших вузлів, щоб вони почали виконувати завдання. В цей час центр оновлює статус на InProgress. Вузли отримують відправлені підзадачі і обраховують їх, виводять у таблицю результати та відправляють іншим вузлам. Центр перевіряє чи всі підзадачі були обраховані і якщо так, то обраховує кінцевий результат та перемикає статус у Success, як можна побачити на рисунку 3.17.

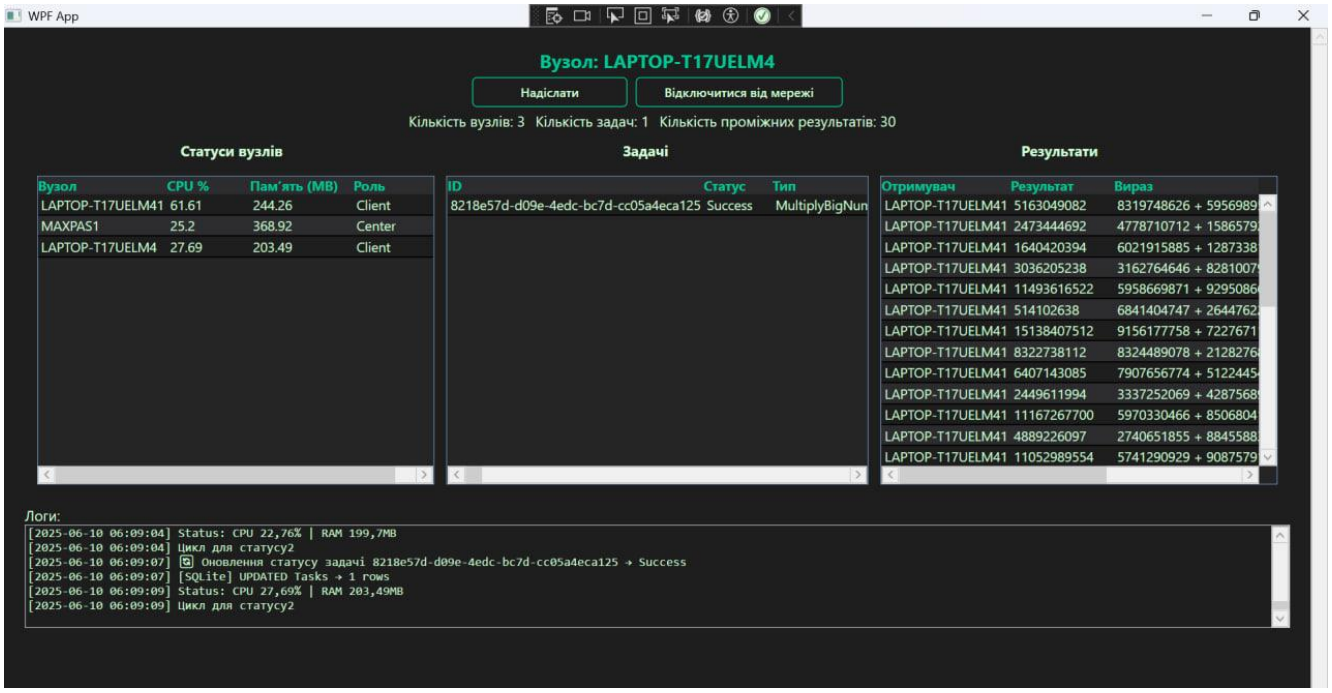


Рисунок 3.17 – Запуск задачі в систему

Адміністратор може переглядати як виконуються задачі на сторінці “Задачі системи”, що зображено на рисунку 3.18.

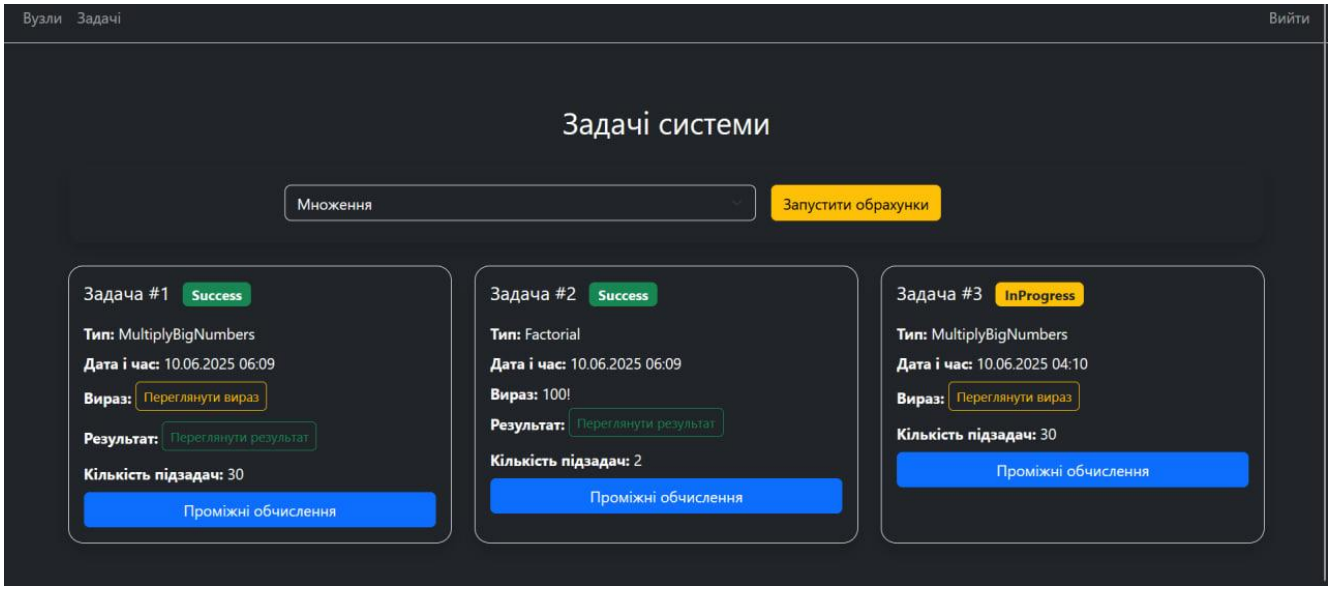


Рисунок 3.18 – Сторінка задач системи

Кожен статус задачі відображається різним кольором. Якщо результат або вираз занадто великий, то виводиться кнопка, яка дозволяє його переглянути у модальному вікні, що можна побачити на рисунку 3.19.

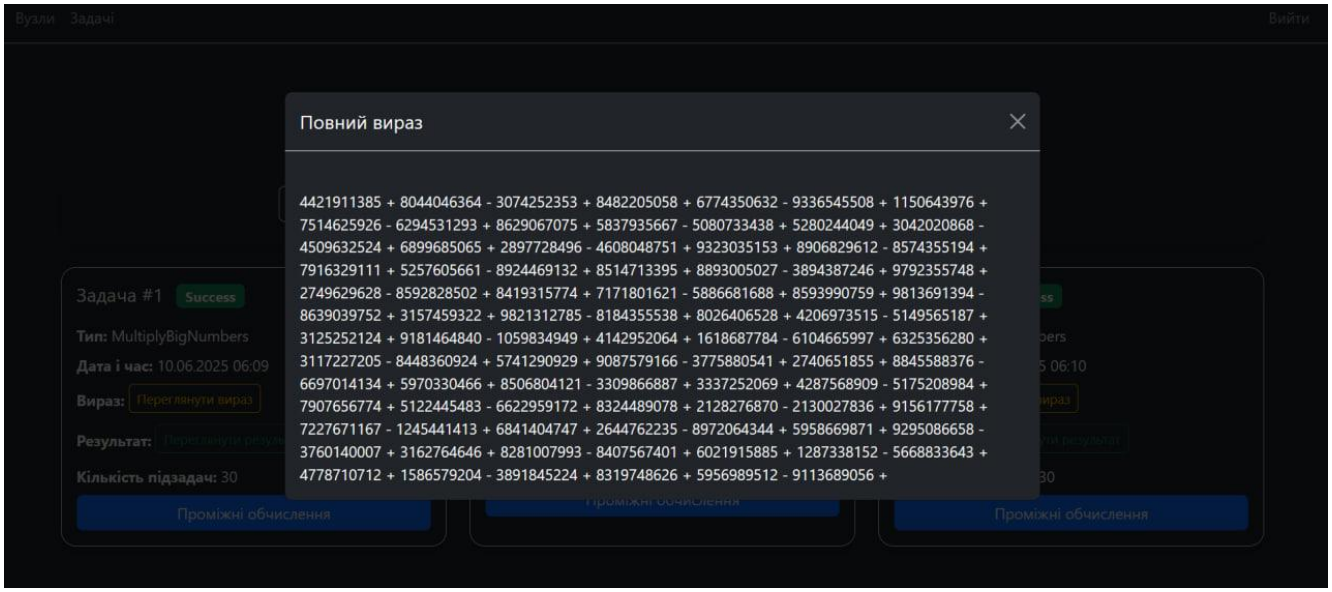


Рисунок 3.19 – Модальне вікно для великого виразу

Для задач, що в прогресі або виконані є кнопка переглянути проміжні обчислення для конкретної задачі. На сторінці, що зображена на рисунку 3.20, можна побачити проміжні обчислення у вигляді таблиці. У ній можна побачити, хто який вираз виконував, який вираз виконував і які результати отримав. Якщо результати або вираз занадто великі, то вони також стають кнопкою, яка веде до модального вікна, як на рисунку 3.19.

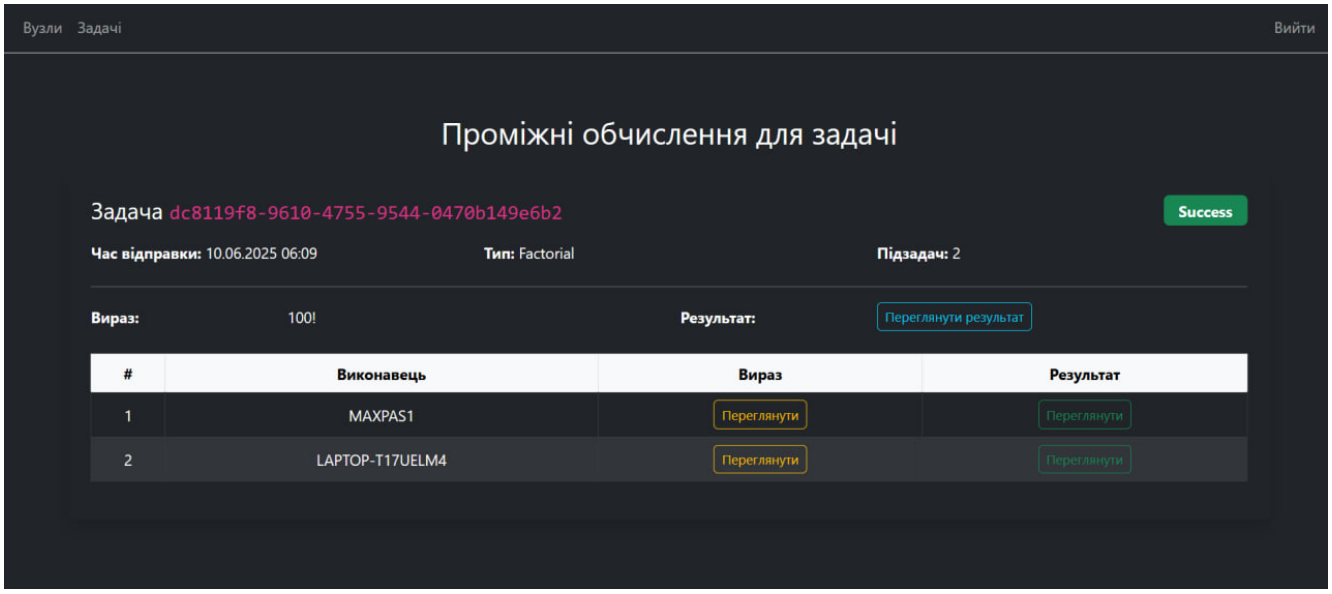


Рис 3.20 – Перегляд проміжних обчислень

3.5. Висновки до третього розділу

У третьому розділі було проведено детальний аналіз алгоритмічного та програмного забезпечення розробленої децентралізованої системи для розподілу обчислювальних задач. Було описано ключові алгоритми взаємодії між компонентами, логіку визначення і передачі ролі центру, обробку обчислювальних задач, а також дії у разі зміни центру або виключення вузла з системи.

Особливу увагу приділено реалізації механізмів самоорганізації, що забезпечують здатність системи автоматично призначати новий центр у разі перевантаження чи збоїв. Система підтримує постійний моніторинг навантаження, використовує локальну базу даних для збереження поточного стану та здатна відновлювати задачі, навіть якщо відбулася зміна центру.

У підрозділі 3.2 описано структуру програмного забезпечення, яка включає WPF-додатки на кожному вузлі, локальні сховища та централізований веб-інтерфейс адміністратора. Всі вузли функціонують як рівноправні одиниці, що забезпечує симетричну, гнучку і стійку архітектуру. Представлена діаграма розгортання демонструє фізичне розміщення компонентів системи у мережі.

Окремий підрозділ присвячено реалізації веб-базованого інтерфейсу, який дозволяє адміністратору запускати задачі, переглядати завантаженість вузлів, керувати центром та отримувати результати обчислень. Інтерфейс побудований на базі ASP.NET Core MVC з використанням WebSocket (SignalR), що забезпечує швидке оновлення інформації в реальному часі. Було наведено макети основних сторінок, зокрема – сторінки вузлів, сторінки задач, сторінки перегляду проміжних обчислень та сторінки авторизації.

Розділ завершується демонстрацією реального застосування системи, де описано повний життєвий цикл задачі: від ініціації адміністратором – до її розподілення, виконання та збереження результатів усіма вузлами. Особливо підкреслено механізм взаємного дублювання результатів, що підвищує надійність.

Таким чином, розділ показує, що розроблена система не лише функціонує в умовах розподілення обчислень, але й реалізує повноцінну платформу для адміністрування, масштабування та забезпечення стійкої роботи в умовах змін або збоїв.

ВИСНОВКИ

У даній дипломній роботі було розроблено, спроектовано та реалізовано програмно-апаратну універсальну розподілену систему для виконання паралельних обчислень з елементами децентралізації та самоорганізації. Було проведені теоретичні дослідження, архітектурне проєктування, вибір технологій. Це дозволило створити надійну, масштабовану платформу, що функціонує у динамічному середовищі та забезпечує автоматичне розподілення обчислювальних задач між вузлами.

У першому розділі було здійснено глибокий аналіз сучасних архітектурних підходів до побудови розподілених систем, таких як багаторівневий, клієнт-серверний, подієво-керований та децентралізований (P2P). На основі цього аналізу було обґрунтовано вибір саме децентралізованої архітектури з елементами централізованого адміністрування. Також було здійснено вибір технічних інструментів: використано мову програмування C# з фреймворками ASP.NET Core MVC для веб-інтерфейсу та WPF для компонентів системи. Комунікація реалізована через WebSocket з використанням JSON як формату обміну даними. Це дозволило закласти гнучкий, безпечний та сучасний технічний фундамент проєкту.

У другому розділі було спроєктовано архітектуру системи, що включає компоненти-учасники (вузли), які рівноправно взаємодіють у P2P-мережі, автоматично обирають центр за критерієм навантаженості та виконують задачі поставлені адміністратором. Розглянуто механізми взаємодії, обміну статусами, перемикання ролей та децентралізованого зберігання даних. Побудовано діаграми прецедентів і класів, що наочно демонструють логіку функціонування. Було сформовано перелік задач, які легко розпаралелюються (сума масиву, перевірка парності, факторіал тощо) та складніших (щільне множення матриць, скалярний добуток), що демонструє практичну ефективність системи.

У третьому розділі було описано програмну реалізацію системи: створено WPF-додатки, які встановлюються на вузли, а також веб-інтерфейс адміністратора

для візуалізації та управління. Реалізовано повний цикл задач: постановка, розподіл, виконання, збір результатів і відновлення після збоїв. Описані алгоритми самоорганізації, балансування навантаження, зміни центра, автоматичної реєстрації вузлів, а також ручне втручання адміністратора. Наведено макети інтерфейсу, діаграми активності, приклади запуску, що підтверджують життєздатність рішення.

Завдяки застосуванню WebSocket-технологій було реалізовано постійний, малозатратний за ресурсами канал двостороннього обміну повідомленнями між усіма учасниками системи. Це дозволяє досягнути низьких затримок у передаванні статусів та задач, оперативно реагувати на зміни в мережі, підтримувати стабільну синхронізацію між усіма вузлами незалежно від їхньої кількості. Структура повідомлень у форматі JSON забезпечила гнучкість, простоту розбору даних та розширюваність протоколу взаємодії.

Розроблений адміністративний інтерфейс, дозволяє здійснювати повноцінне керування системою з будь-якого пристрою у мережі. Поєднання автоматичної та ручної моделі управління дозволяє адаптувати систему до різних сценаріїв використання – як у демонстраційному середовищі, так і в продуктивному середовищі з більшим навантаженням.

Отримані результати можуть бути використані, як основа для побудови більш складних обчислювальних кластерів, а також як навчальний приклад для проєктування розподілених систем нового покоління.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Architecture Styles in Distributed Systems. URL: https://www.geeksforgeeks.org/architecture-styles-in-distributed-systems (дата звернення: 15.04.25)
2. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design. Pearson Education. 2021, 1068 p.
3. What is a Distributed System? URL: https://www.geeksforgeeks.org/what-is-a-distributed-system (дата звернення: 15.04.25)
4. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 614 p.
5. Client-Server Architecture - System Design. URL: https://www.geeksforgeeks.org/client-server-architecture-system-design/ (дата звернення: 16.04.25)
6. Peer-to-Peer (P2P) Architecture. URL: https://www.geeksforgeeks.org/peer-to-peer-p2p-architecture/ (дата звернення: 16.04.25)
7. Event-Driven Architecture - System Design. URL: https://www.geeksforgeeks.org/event-driven-architecture-system-design/ (дата звернення: 19.04.25)
8. Burns B. Designing Distributed Systems. O'Reilly Media, 2018.166 p.
9. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference. O'Reilly Media, 2022. 1058 p.
10. Troelsen A. Pro C# 10 with .NET 6: foundational principles and practices in programming. New York: Apress, 2022. 1705 p.
11. Skeet J. C# in Depth, Fourth Edition. Manning, 2019. 526 p.
12. Asynchronous programming with async and await. URL: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/ (дата звернення: 20.04.25)
13. Vermeit N. Introducing .NET 6. Apress, 2022. 319 p.

14. Nathan, A. Windows Presentation Foundation Unleashed. Pearson Education, 2018. 191 p.

15. Arnaud Weil. Learn WPF MVVM - XAML, C# and the MVVM pattern. Lulu.com, 2016, 174 p.

16. Alex Khang. Professional WPF and C# Programming: Practical Software Development Using WPF and C#. XOKR Express, 2019. 405 p.

17. Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice (SEI Series in Software Engineering). Addison-Wesley Professional, 2021. 464 p.

18. Wilson Hayes. C# Programming for Graphical User Interface Development: A Beginner's Guide to Building Dynamic and User-Friendly GUI Applications with Windows Forms, WPF, and More. Apress, 2024. 375 p.

19. Dawson E. GUI Development in C#: Modern Interface Design. Kindle Edition, 2025. 315 p.

20. WebSockets in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/websockets (дата звернення: 11.05.25)

21. Jackson B. Hands-On Real-Time Web Applications with WebSockets. Packt Publishing, 2019. 350 p.

22. The WebSocket Protocol. RFC 6455. URL: https://tools.ietf.org/html/rfc6455 (дата звернення: 12.05.25)

23. Stephen Cleary. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. O'Relly, 2019. 150 p.

24. Lock A. ASP.NET Core in Action. Manning, 2023. 984 p.

25. Freeman A. Pro ASP.NET Core MVC 6. Apress, 2021.1286 p.

26. Horsdal C. Microservices in ASP.NET Core. Manning, 2022. 348 p.

27. Introduction to ASP.NET Core MVC. URL: https://learn.microsoft.com/en-us/training/modules/aspnet-core-mvc-introduction/ (дата звернення: 15.05.25)

28. Troelsen A., Japikse P. Pro C# 8 with .NET Core 3: Foundational Principles and Practices in Programming. Apress, 2020. 121 p.

29. Peres R. Modern Web Development with ASP.NET Core 3: An end to end guide covering the latest features of Visual Studio 2019. Packtx, 2020. 416 p.

30. Introduction to Bootstrap. URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення: 16.05.25)

31. Layton J. Learning Bootstrap 5: Create modern websites using the power of Bootstrap 5. Packt Publishing, 2021. 325 p.

32. Foreman D. Responsive Web Design with HTML5 and Bootstrap 5. Foreman Technology LTD, 2022. 694 p.

33. Foreman D. Bootstrap 5 Foundations. Foreman Technology LTD, 2021. 128 p.

34. Hussain C. Mastering Bootstrap 5. Sonar Publishing, 2023. 482 p.

35. Use Bootstrap in ASP.NET Core projects. URL: https://learn.microsoft.com/en-us/aspnet/core/client-side/bootstrap (дата звернення: 16.05.25)

36. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Relly, 2017. 390 p.

37. Ford N., Richards M., Sadalage P., Dehghani Z. Software Architecture: The Hard Parts. O'Relly, 2021. 132 p.

38. WPF Documentation - Microsoft Docs. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 16.05.25)

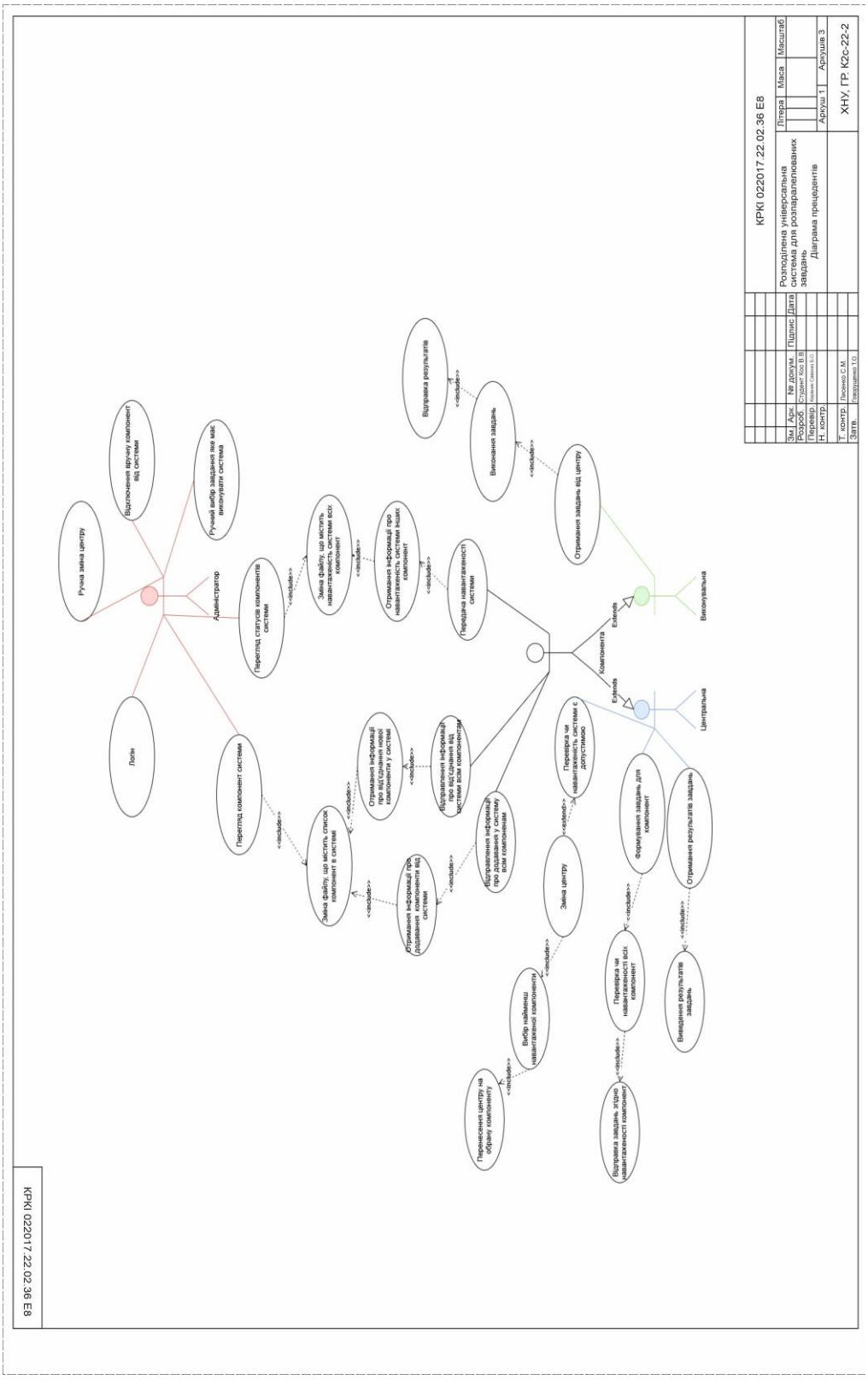
39. Freeman A. Essential ASP.NET Core MVC 3. Apress, 2020.1018 p.

40. Olivera J. ASP.NET Core MVC Cookbook. Packt Publishing, 2018. 668 p.

Додаток А

(обов’язковий)

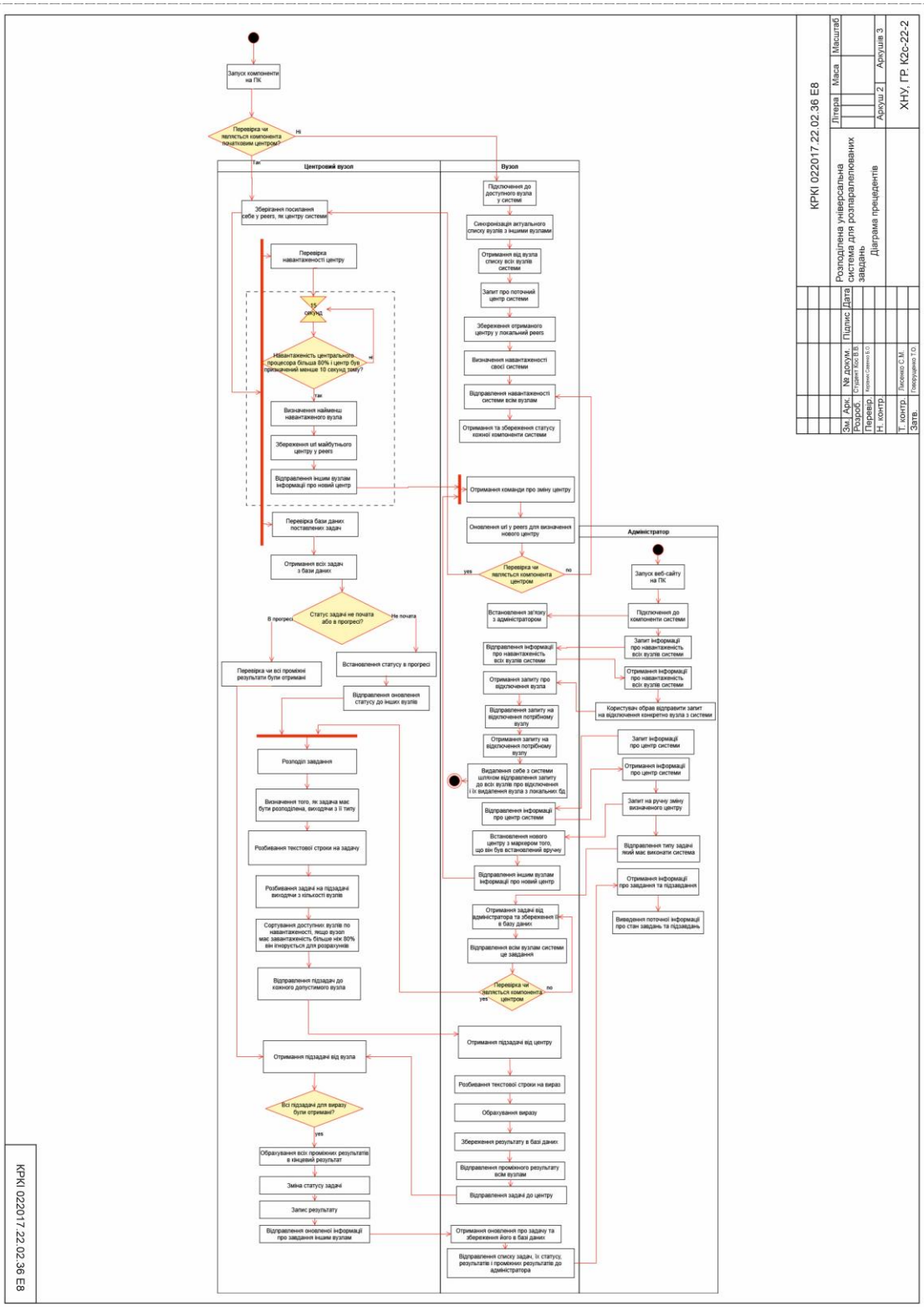
КОПІЯ КРЕСЛЕННЯ «ДІАГРАМА ПРЕЦЕДЕНТІВ»



Додаток Б

(обов’язковий)

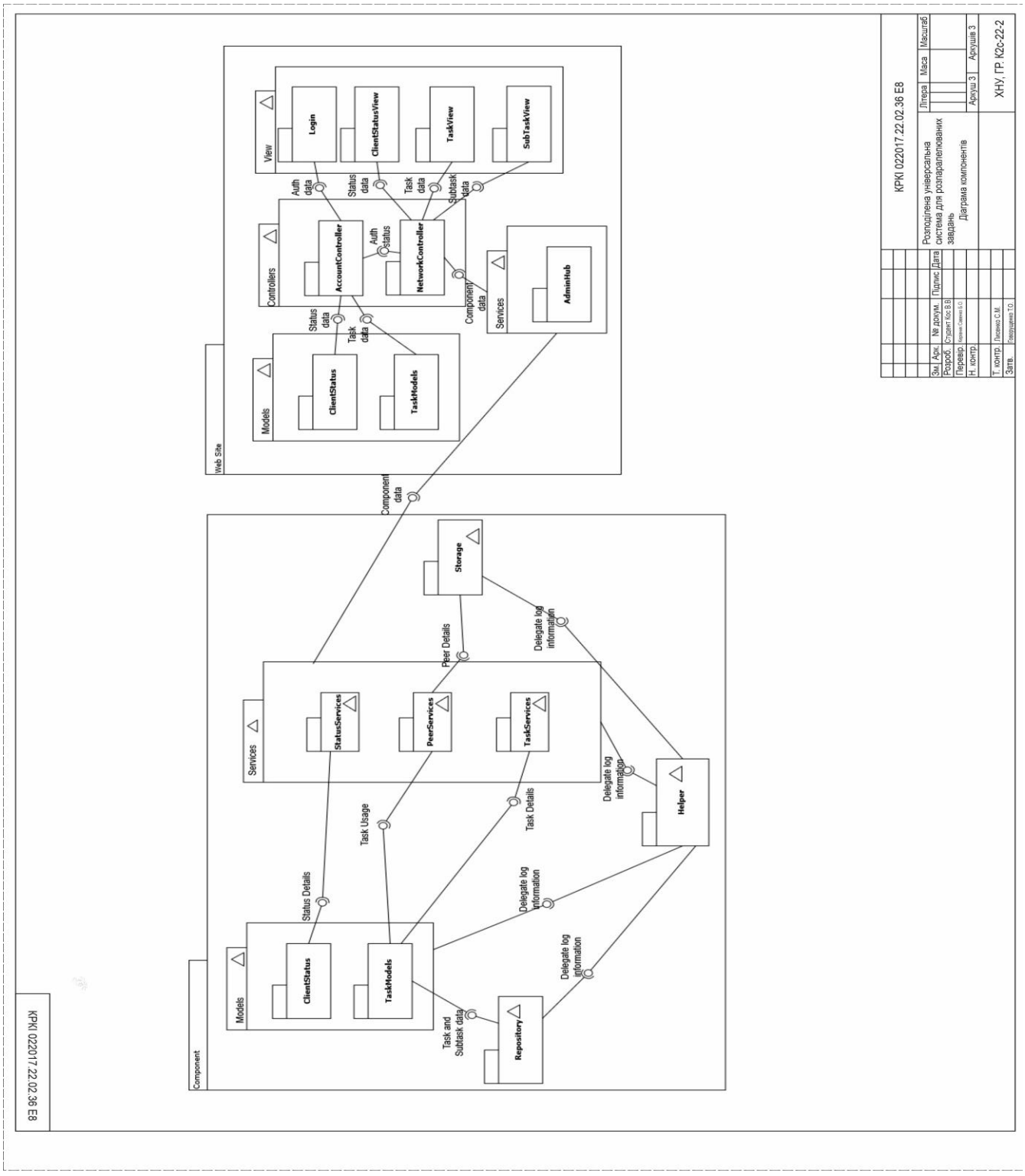
КОПІЯ КРЕСЛЕННЯ «ДІАГРАМА АКТИВНОСТІ»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «ДІАГРАМА КОМПОНЕНТ»



Додаток Г

(обов’язковий)

КОД ПРОГРАМИ

Вихідний програмний код, що було створено під час розробки програмно-апаратної розподіленої універсальної системи для розпаралелювання завдань, є відкритим для загального доступу і доступним у відкритому репозиторії, за посиланням – https://github.com/MrGold6/DSys.

Даний репозиторій містить програмні модулі компоненти та адміністративного веб-сайту.

Адміністративний веб-сайт здійснює ручне керування системою, та запуск задач на виконання. Компоненти розподіляють та виконують завдання, що їм відправляє адміністративний веб-сайт, обмінюються завантаженістю. Усі компоненти мають рівноправні можливості, але на деякий час один з цих вузлів виконує роль центру – координатор розподілу задач та той хто збирає їх результати.

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Валерія КОС

Співавтор:

Назва: Кос_Розподілена універсальна система для розпаралелювання завдань

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.9%

Коефіцієнт подібності 2:0.2%

Мікропробіли: 16

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-12 10:19:10.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-12 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 13.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 9%**

	Document		Sum coincidence on the DB	
ID: 245280 Title: БКР Розподілена універсальна система для розпаралелювання завдань Added in a DB: 2025-06-12 Authors: Валерія КОС Heads: Богдан САВЕНКО Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	81264	757	11247 (14%)	104 (14%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
240802	Title: Звіт з ПДП Розподільна універсальна система для розпаралелюваних завдань Added in a DB: 2025-05-04 Authors: Кос В.В. Heads: Савенко О.С. Consultants: Opponents:	10198 (13.0%)	88 (12.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Кос Валерія Вікторівна

Тема: Розподілена універсальна система для розпаралелювання завдань

Спеціальність**:** 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___70___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є реалізація розподіленої універсальної системи для розпаралелювання завдань.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: у першому розділі було здійснено глибокий аналіз сучасних архітектурних підходів до побудови розподілених систем. На основі цього аналізу було обґрунтовано вибір саме децентралізованої архітектури з елементами централізованого адміністрування. Також було здійснено вибір технічних інструментів: використано мову програмування C# з фреймворками ASP.NET Core MVC для веб-інтерфейсу та WPF для компонентів системи. Комунікація реалізована через WebSocket з використанням JSON як формату обміну даними. Це дозволило закласти гнучкий, безпечний та сучасний технічний фундамент проєкту.

У другому розділі було спроєктовано архітектуру системи, що включає компоненти-учасники, які рівноправно взаємодіють у P2P-мережі, автоматично обирають центр за критерієм навантаженості та виконують задачі поставлені адміністратором. Розглянуто механізми взаємодії, обміну статусами, перемикання ролей та децентралізованого зберігання даних. Побудовано діаграми прецедентів і

класів, що наочно демонструють логіку функціонування. Було сформовано перелік задач, які легко розпаралелюються, та демонструють практичну ефективність системи.

У третьому розділі було описано програмну реалізацію системи: створено WPF-додатки, які встановлюються на вузли, а також веб-інтерфейс адміністратора для візуалізації та управління. Реалізовано повний цикл задач: постановка, розподіл, виконання, збір результатів і відновлення після збоїв. Описані алгоритми самоорганізації, балансування навантаження, зміни центру, автоматичної реєстрації вузлів, а також передбачено ручне втручання адміністратора. Наведено макети інтерфейсу, діаграми активності, компонент та розгортання, приклади запуску, що підтверджують життєздатність рішення.

4. Позитивні сторони роботи: успішно реалізована програмно-апаратна розподілена система, яка ефективно справляється з задачами різних типів. Використані нові технології, добре продуманий інтерфейс для користувача.

5. Негативні сторони роботи: недостатньо приділена увага кібербезпеці системи. Центр системи не має достатньої гнучкості, не передбачено вибір вузького кола вузлів, що можуть бути центрами.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Бодянон І.П. зав. каф. ІПЗ, ХНУ

"13" 06 2025 р. _____(підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Валерії КОС
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-22-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

11.06 2025 року

Кос В.В

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Розподілена універсальна система для розпаралелювання завдань

Автор: Валерія КОС

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Богдан САВЕНКО, доктор філософії

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами: "Вступ, "Висновки";
4) в якості запозичень в окремих місцях системою зафіксовано збіги в термінах, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) технічні модифікації тексту, такі як мікропробіли та парафрази є наслідком форматування, а не цілеспрямованого уникнення перевірки.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1.93% і адресується до 22 першоджерела; та системою Anti-Plagiarism складає 13%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Богдан САВЕНКО

Гарант ОП _____ Андрій НЕЧІПОРУК

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА