

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«__» _____ 2025 р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення
робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Інтелектуальна система розподілу завдань для роботизованих
систем»**

Виконав:

студент IV курсу, групи ІК-11

Рудяков Ілля Олексійович _____

Керівник:

доцент, к.т.н., доц.,

Батрак Євгеній Олександрович _____

Рецензент:

доцент кафедри ОТ, к.т.н., доц.,

Волокита Артем Миколайович _____

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«__» _____ 2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Рудякову Іллі Олексійовичу

1. Тема проєкту «Інтелектуальна система розподілу завдань для роботизованих систем», керівник проєкту Батрак Євгеній Олександрович, доцент кафедри ІСТ, затверджені наказом по університету від «23» травня 2025 р. № 1705-с
2. Термін подання студентом проєкту: 9 червня 2025 р
3. Вихідні дані по проєкту: система повинна забезпечувати автоматизоване управління розподілом завдань між роботизованими агентами з урахуванням їхнього поточного стану, завантаженості та пріоритетності завдань. Передбачається підтримка декількох алгоритмів призначення завдань (наприклад, випадковий, за навантаженням, за доступністю), з можливістю гнучкого налаштування на рівні організації. Система має інтегруватись з інтерфейсом візуалізації для моніторингу в реальному часі поточного статусу робіт, завдань.
4. Зміст пояснювальної записки: перелік умовних скорочень та термінів, вступ, аналіз існуючих рішень, визначення вимог до системи розподілу задач, розроблення серверної та клієнтської частини системи, тестування розробленої системи, висновки, перелік використаних джерел.

5. Перелік графічного матеріалу: діаграма архітектури, блок-схема процесу розподілу, діаграма класів, діаграма прецедентів.

6. Дата видачі завдання: «1» лютого 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Аналіз предметної області та постановка задачі	1.02 – 10.02	Виконано
2.	Розробка загальної архітектури системи	10.02 – 20.02	Виконано
3.	Проектування бази даних та структури завдань	20.02 – 5.03	Виконано
4.	Реалізація моделі зберігання та обробки задач	5.03 – 15.03	Виконано
5.	Розробка алгоритмів розподілу задач	15.03 – 29.03	Виконано
6.	Побудова API для взаємодії з роботами та призначенням задач	29.03 – 10.04	Виконано
7.	Реалізація модуля моніторингу та аналітики навантаження	10.04 – 20.04	Виконано
8.	Розробка інтерфейсу користувача	20.04 – 10.05	Виконано
9.	Тестування системи, перевірка розподілу і балансування	10.05 – 18.05	Виконано
10.	Оптимізація, внесення змін	18.05 – 20.05	Виконано
11.	Написання пояснювальної записки	20.05 – 25.05	Виконано
12.	Підготовка до захисту	25.05 – 31.05	Виконано

Студент

Ілля РУДЯКОВ

Керівник

Євгеній БАТРАК

АНОТАЦІЯ

Інтелектуальна система розподілу завдань у роботизованих системах.

Проект містить 104 с. тексту, 54 рисунки, 21 таблицю, посилання на 22 літературні джерела, додатки та 4 конструкторських документи.

РОЗПОДІЛ ЗАВДАНЬ, БАГАТОАГЕНТНІ СИСТЕМИ, АЛГОРИТМИ ПРИЗНАЧЕННЯ, ОПТИМІЗАЦІЯ, РОБОТОТЕХНІКА.

У проекті розроблено інтелектуальну систему автоматизованого розподілу завдань для багатокомпонентного середовища з роботизованими агентами, яка забезпечує гнучке та обґрунтоване призначення задач залежно від характеристик роботів, параметрів задач і заданої логіки розподілу.

Об'єктом дослідження виступає процес управління задачами в системах з автономними виконавцями. Предметом є алгоритми динамічного розподілу завдань у багатозадачному середовищі з урахуванням змінних станів агентів та вимог до задач.

Метою проекту є побудова гнучкої системи, яка дозволяє конфігурувати різні стратегії призначення задач у роботизованих системах та адаптувати логіку розподілу під специфіку середовища. У системі реалізовано набір алгоритмів: від простих евристик (Random, Least Connections) до більш складних підходів на основі методів штучного інтелекту (Ant Colony, Genetic Algorithm, Simulated Annealing, Fuzzy Logic, Linear Optimization). Задачі проходять попередню фільтрацію за категоріями, обов'язковими та бажаними вимогами, після чого система здійснює вибір оптимального виконавця.

Система має централізовану серверну архітектуру з REST API, адміністративну клієнтську частину, а також модуль збору й віртуалізації аналітичних даних. Проведене тестування підтвердило здатність системи адаптивно та стабільно розподіляти задачі в умовах різних навантажень і варіативних вимог до виконання.

SUMMARY

The Intelligent task distribution system for robotic systems.

The project contains 104 pages. text, 54 figures, 21 tables, links to 22 literary sources, annexes and 5 design documents.

Keywords: task distribution, multi-agent systems, assignment algorithms, optimization, robotics.

This project presents the development of an intelligent task distribution system designed for multi-component environments with robotic agents. The system enabled flexible and justified task assignment based on robot characteristics, task parameters, and configurable allocation logic. The object of the study is the task management process in systems with many autonomous executors. The subject is the algorithms for dynamic task distribution in multitasking environments, considering the changing states of agents and task requirements.

The aim of the project is to create a flexible system that allows configuring various task assignment strategies for robotic systems and adapting the allocation logic to different environmental conditions. The system includes a set of algorithms, ranging from basic heuristics (Random, Least Connections) to advanced artificial intelligence-based approaches (Ant Colony, Genetic Algorithm, Simulated Annealing, Fuzzy Logic, Linear Optimization). Tasks undergo preprocessing through filtering by category and by mandatory and optional requirements, followed by the selection of the most suitable executor.

The architecture features a centralized backend with REST API, an administrative client interface, and a module for collecting and visualizing analytical data. Testing confirmed the system's ability to distribute tasks adaptively and reliably under different conditions and variable execution constraints.

№ рядка	Формат	Позначення	Найменування	Кільк.	№ екзем.	Примітка
1			<u>Документація загальна</u>			
2						
3			Знову розроблена			
4						
5	A4	ІК11.270БАК.006 ПЗ	Інтелектуальна система розподілу	97		
6			завдань для роботизованих систем.			
7			Пояснювальна записка			
8						
9	A3	ІК11.270БАК.006 Д1	Інтелектуальна система розподілу	1		
10			завдань для роботизованих систем.			
11			Діаграма архітектури			
12						
13	A3	ІК11.270БАК.006 Д2	Інтелектуальна система розподілу	1		
14			завдань для роботизованих систем.			
15			Блок схема процесу розподілу			
16						
17	A3	ІК11.270БАК.006 Д3	Інтелектуальна система розподілу	1		
18			завдань для роботизованих систем.			
19			Діаграма класів			
20						
21	A3	ІК11.270БАК.006 Д4	Інтелектуальна система розподілу	1		
22			завдань для роботизованих систем.			
23			Діаграма прецедентів			
24						
25						
26						
27						
28						
Зм.	Лист	№ докум.	Підпис	Дата	ІК11.270БАК.006 ТП	
Розроб.		Рудяков І.О.			Літ.	Аркуш
Керівн.		Батрак Є.О.			Т	Аркушів
					1	1
					КПІ ім. Ігоря Сікорського	
Затв.		Ролік О. І.			Група ІК-11	
					Відомість дипломного проєкту	

**Пояснювальна записка
до дипломного проєкту
на тему: «Інтелектуальна система розподілу завдань
для роботизованих систем»**

Київ – 2025 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	5
ВСТУП.....	6
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ	8
1.1 Предметна область та актуальність проблеми	8
1.1.1 Сучасні виклики в управлінні багатоагентними системами	8
1.1.2 Проблеми розподілу задач між агентами	9
1.2 Класифікація підходів до розподілу завдань	10
1.2.1 Централізовані та децентралізовані системи	10
1.2.2 Роль алгоритмів призначення в багатокритеріальному середовищі	12
1.2.3 Алгоритми оптимального та евристичного розподілу	14
1.3 Огляд схожих систем.....	15
1.3.1 Система розподілу завдань для прибирання морського пластику	15
1.3.2 Система Task Allocation in Autonomous Warehouse Robots (Amazon Robotics)	17
1.3.3 Децентралізований жадібний алгоритм розподілу завдань	19
Висновки до розділу 1.....	20
2 ВИМОГИ ДО СИСТЕМИ	22
2.1 Бізнес-вимоги.....	22
2.1.1 Основні функціональні вимоги.....	22
2.1.2 Вимоги до взаємодії користувача з системою	23
2.1.3 Вимоги до масштабованості та адаптивності	24
2.2 Архітектурні вимоги	25
2.3 Вимоги до безпеки	26
2.3.1 Аутентифікація та авторизація користувачів.....	26
2.3.2 Захист API	27

					ІК11.270БАК.006 ПЗ							
Зм.	Лист	№ докум.	Підпис		Інтелектуальна система розподілу завдань для роботизованих систем. Пояснювальна записка	Літ.		Арк.		Аркушів		
Розробив	Рудяков І.О.			Т				2	97			
Перевірив	Батрак Є.О.											
Затв.	Ролік О. І.								КПІ ім. Ігоря Сікорського Група ІК-11			

2.3.3 Шифрування конфіденційних даних	27
2.3.4 Захист від атак	27
Висновки до розділу 2.....	28
3 РОЗРОБЛЕННЯ СИСТЕМИ	30
3.1 Проектування архітектури системи.....	30
3.1.1 Загальні принципи.....	30
3.1.2 Архітектура системи	31
3.1.3 Архітектурна реалізація серверної та клієнтської частини	32
3.2 Серверна частина.....	35
3.2.1 Реєстрація та аутентифікація користувачів.....	35
3.2.2 Керування користувачами, ролями та доступами	36
3.2.3 Конфігурація системи	37
3.2.4 Створення категорій робітв	38
3.2.5 Створення можливостей.....	39
3.2.6 Створення робітв	40
3.2.8 Розподілення задач (алгоритмічна частина)	42
3.3 Клієнтська частина.....	44
3.3.1 Реєстрація та аутентифікація.....	44
3.3.2 Керування користувачами, ролями та правами доступу	45
3.3.3 Конфігурація системи	47
3.3.4 Створення категорій.....	48
3.3.5 Створення можливостей.....	49
3.3.6 Створення робітв	50
3.3.7 Створення задач.....	52
3.3.8 Розподіл задач	54
3.4 Алгоритми	55
3.4.1 Latest Task Finished	55
3.4.2 Least Connections	56
3.4.3 Load Balancing.....	57
3.4.4 Random	57

3.4.5 Round Robin.....	58
3.4.6 Ant Colony	59
3.4.8 Genetic Task	60
3.4.9 Fuzzy Logic.....	61
3.4.10 Simulated Annealing	63
3.4.11 Linear Optimization	64
3.4.12 Порівняльна характеристика алгоритмів.....	65
Висновки до розділу 3.....	66
4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ.....	68
4.1 Методика тестування	68
4.1.1 Цілі тестування	68
4.1.2 Вхідні дані для моделювання	68
4.1.3 Метрики оцінки ефективності	69
4.1.4 Процедура тестування	70
4.2 Тестування алгоритмів.....	71
4.2.1 Latest Task Finished	72
4.2.2 Least Connections	74
4.2.3 Load Balancing.....	76
4.2.4 Random	78
4.2.5 Round Robin.....	80
4.2.6 Ant Colony	82
4.2.7 Genetic Task	84
4.2.8 Fuzzy Logic.....	86
4.2.9 Simulated Annealing	88
4.2.10 Linear Optimization	90
Висновки до розділу 4.....	92
ВИСНОВКИ.....	94
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	96

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІС – інформаційна система.

АНР (англ. Analytic hierarchy process) – метод аналізу ієрархій

API (англ. Application Programming Interface) – інтерфейс прикладного програмування.

AUV (англ. Autonomous underwater vehicle) – автономний підводний апарат.

ERP (англ. Enterprise resource planning) – планування ресурсів підприємства.

ETA (англ. Estimated Time of Arrival) – очікуваний час завершення.

ID (англ. Identifier) – ідентифікатор.

ILP (англ. Integer Linear Programming) – лінійне цілочисельне програмування.

IoT (англ. Internet of Things) – інтернет речей.

JWT (англ. Json Web Token) – токен аутентифікації.

MFA (англ. Multi-Factor Authentication) – багатофакторна аутентифікація.

TOPSIS (англ. Multi-Criteria decision analysis method) – багатокритеріальний метод аналізу рішень.

ВСТУП

У сучасних умовах стрімкого розвитку автоматизації та робототехніки дедалі більше компаній, дослідницьких інститутів і виробничих впроваджують багатокомпонентні роботизовані системи для виконання різноманітних завдань. Такі системи активно використовуються в логістиці, промисловому виробництві, сільському господарстві, побуті та оборонній сфері. Зі зростанням кількості взаємодіючих роботів з'являється новий виклик: як ефективно організувати процес автоматизованого розподілу завдань між виконавцями так, щоб забезпечити стійкість, гнучкість і стабільну продуктивність системи загалом.

Об'єктом дослідження у цьому проєкті є процес розподілу завдань між агентами у багатокомпонентному середовищі з автономними роботами. Предметом дослідження виступають алгоритмічні підходи до автоматизованого призначення завдань із урахуванням поточного стану роботів, їхніх характеристики, специфіки завдань і обмежень середовища.

Більшість існуючих систем управління базуються на простих правилах або централізованій логіці, що не враховує динаміку змін – таких як завантаження роботів, черги задач, або зміна доступності ресурсів. Відсутність адаптивності призводить до зниження ефективності, появи простоїв, надмірного навантаження окремих агентів, а іноді – до конфліктів у системі. Практика впровадження систем такого типу в Amazon Robotics, на автономних складах або серед дронів-постачальників показує, що лише гнучке адаптивне управління розподілом навантаження дозволяє масштабувати систему без втрат продуктивності.

Наукові дослідження останніх років доводять актуальність задач оптимального і адаптивного розподілу. Популярності набувають підходи на основі машинного навчання, метаевристичних алгоритмів (генетичні, колонії мурах, імітаційне загартування) та багатокритеріальних моделей. Проте широке використання цих методів у реальних умовах часто обмежується через складність реалізації, нестабільність у непередбачуваних ситуаціях, потребу в потужних обчислювальних ресурсах і відсутність гнучких архітектур.

					ІК11.270БАК.006 ПЗ	Арк.
						6
Зм.	Лист	№ докум.	Підпис	Дата		

Метою проекту є побудова інтелектуальної адаптивної системи автоматизованого розподілу завдань для групи роботизованих агентів з підтримкою різних стратегій призначення, урахуванням стану виконавців, характеристик задач та конфігурацій середовища.

Для досягнення цієї мети необхідно вирішити наступні завдання:

- дослідити існуючі методи розподілу завдань для багатоагентних систем;
- визначити архітектуру нової системи, обрати відповідні технології та протоколи взаємодії;
- запропонувати спосіб отримання інформації про стан роботів;
- розробити ядро системи, яке пропонує підтримку різних алгоритмів розподілу завдань;
- застосувати REST API для взаємодії з агентами (роботами);
- застосувати модуль аналітики та подання для моніторингу активних завдань та навантаження;
- протестувати систему в змодельованому середовищі та в різних умовах.

Запропоноване рішення базується на розширюваній архітектурі, яка дозволяє конфігурувати алгоритми залежно від цілей користувача. Особливістю підходу є активна участь агентів у процесі отримання завдань через API, що забезпечує децентралізований режим взаємодії та високу стійкість до збоїв. Завдання проходять попередню фільтрацію за категоріями, обов'язковими та необов'язковими вимогами, після чого на основі обраного алгоритму виконується пошук найкращого виконавця. Завдяки підтримці широкого спектра стратегій, система може адаптуватися до різних типів роботів, сценаріїв і задач та під різні сценарії використання.

Отримані результати можуть бути використані:

- у сільському господарстві для управління парком сільськогосподарських роботів;
- в логістичних терміналах та доставці;
- для керування дронами та безпілотними автомобілями;
- в інших галузях, де важлива правильна організація завдань.

					ІК11.270БАК.006 ПЗ	Арк.
						7
Зм.	Лист	№ докум.	Підпис	Дата		

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предметна область та актуальність проблеми

1.1.1 Сучасні виклики в управлінні багатоагентними системами

Системи з множинними роботизованими агентами стають дедалі популярнішими у різних сферах діяльності: промисловості, логістиці, аграрному секторі, обороні, медицині та рятувальних операціях. Завдяки здатності до паралельної роботи, такі системи дозволяють значно підвищити продуктивність, зменшити час виконання завдань та забезпечити більш гнучке реагування на зміни в середовищі. Однак, зростання кількості роботів у системі призводить до ускладнення процесів керування, координації та розподілу завдань, через що погіршується рівень координації між агентами..

Одним із ключових викликів є динамічність середовища виконання. Роботи можуть мати різні технічні характеристики, рівень заряду батареї, навантаження або статус (наприклад, зайнятий, очікує, несправний). Крім того, завдання можуть мати різну складність, пріоритетність або часові обмеження, що вимагає гнучкого та інтелектуального підходу до їхнього розподілу.

Іншим важливим аспектом є обмеженість ресурсів. Часто системи працюють у реальному часі, де необхідно приймати рішення швидко, без надмірних затримок. Це накладає обмеження на обчислювальні потужності та пропускну здатність мережевих каналів, особливо у випадках з децентралізованими системами, де роботи обмінюються інформацією між собою.

Крім того, зростає проблема у масштабованості. Система має ефективно працювати як з декількома, так і з багатьма роботами без значного зниження продуктивності. Це потребує побудови архітектурних рішень, які здатні адаптуватися до змін у кількості агентів без потреби в повному перегляді логіки розподілу завдань.

Не менш важливою є надійність. В умовах реального виробництва чи складського середовища збої в роботі одного агента не повинні призводити до зупинки всієї системи. Це вимагає використання відмовостійких механізмів та

					ІК11.270БАК.006 ПЗ	Арк.
						8
Зм.	Лист	№ докум.	Підпис	Дата		

алгоритмів, здатних динамічно перерозподіляти завдання у разі виявлення помилки або втрати з'єднання з одним із роботів. Таким чином, сучасні багатоагентні системи стикаються з рядом викликів, які зумовлюють необхідність розробки нових, більш ефективних підходів до розподілу завдань, що враховують поточний стан агентів, наколишнє середовище та характеристики самих завдань. Розв'язання цих проблем є основою для побудови високопродуктивних, гнучких і надійних роботизованих систем нового покоління.

1.1.2 Проблеми розподілу задач між агентами

Розподіл задач між агентами є однією з центральних проблем в управлінні багатоагентними системами. Ефективність цього процесу напряду впливає на загальну продуктивність, швидкість виконання завдань, ступінь простою роботів та збалансованість навантаження в системі. Нерівномірне або невдале призначення задач може призвести до перевантаження окремих агентів, зниження ефективності або навіть до втрати задачі через недотримання часових обмежень.

Однією з головних проблем є вибір критерію оптимальності. У різних випадках це можуть бути мінімізація загального часу виконання (makespan), зменшення простою, балансування навантаження між роботами або мінімізація втрат енергії. У багатьох випадках ці критерії суперечать один одному, що робить задачу багатокритеріальною. Наприклад, мінімізація часу виконання може призводити до нераціонального споживання енергії або до перевантаження найбільш потужних роботів [3].

Ще однією складністю є гетерогенність (варіативність) агентів та їх властивостей. Роботи в системі можуть мати різні технічні характеристики, апаратні та функціональні можливості, обмеження в пересуванні або здатність виконувати лише певні типи задач. Це означає, що не кожна задача може бути виконана кожним агентом. Тому при розподілі необхідно враховувати відповідність між вимогами задачі (наприклад, необхідні сенсори, інструменти, швидкість, вантажопідйомність) і характеристики агента.

					ІК11.270БАК.006 ПЗ	Арк.
						9
Зм.	Лист	№ докум.	Підпис	Дата		

Суттєвим чинником є й динаміка середовища. У реальних умовах завдання можуть надходити до системи в режимі реального часу, а стан агентів постійно змінюється (наприклад, через завершення задачі, зміну місцезнаходження або технічний збій). Це ускладнює застосування статичних алгоритмів розподілу, які розраховані на наперед відомі умови. Натомість виникає потреба в реактивному плануванні, де система реагує на зміну стану у реальному часі [13].

Важливим аспектом є також обмеженість інформації. У деяких архітектурах, особливо децентралізованих, кожен агент володіє лише частковою інформацією про стан інших роботів або загальну чергу задач. Це накладає обмеження на точність прийнятих рішень і вимагає розробки алгоритмів, здатних ефективно працювати за умов неповної інформації.

Таким чином, проблема розподілу у багатоагентних системах є комплексною та багатofакторною. Вона вимагає побудови адаптивних, масштабованих і інтелектуальних алгоритмів, які здатні забезпечити ефективне функціонування системи в умовах реального середовища, динамічних змін і непередбачуваних ситуацій.

1.2 Класифікація підходів до розподілу завдань

1.2.1 Централізовані та децентралізовані системи

Системи розподілу задач у багатоагентних середовищах поділяються на два основні типи за архітектурним принципом: централізовані та децентралізовані [14]. Кожен із підходів має свої переваги, недоліки та області застосування.

Централізовані системи базуються на наявності одного або декількох центральних контролерів (диспетчерів, серверів), які володіють повною або майже повною (частковою) інформацією про стан усіх існуючих агентів, чергу задач і поточну ситуацію в середовищі. Саме ці контролери приймають рішення про те, як і кому призначити задачі відповідно до заданих вимог. Агенти в такій архітектурі виконують команди без самостійного прийняття рішень, тобто сервер повністю відповідає за збалансоване розподілення задач.

Перевагами централізованого підходу є:

- повна інформованість – система має доступ до глобального стану, що дозволяє приймати зважені та оптимальні рішення;
- простота реалізації логіки – всі обчислення і логіка зосереджені в одному місці;
- легкість у відлагодженні та модифікації – внесення змін до алгоритмів розподілу здійснюється централізовано.

Недоліки централізованих систем:

- низька масштабованість – зі збільшенням кількості робіт зростає навантаження на центральний вузол;
- вразливість до відмов – вихід з ладу диспетчера може призвести до повної зупинки системи;
- затримки в комунікації – централізований обмін повідомленнями створює додаткову латентність.

Централізовані підходи доцільні в умовах обмеженої кількості робіт, стабільного середовища або коли надзвичайно важлива оптимальність рішення.

Децентралізовані системи працюють за принципом автономності. Кожен робот самостійно приймає рішення про прийняття або відхилення задачі, часто базуючись на частковій інформації або в результаті комунікації. В таких системах може не існувати єдиного диспетчера, що координує агентів.

Переваги децентралізації:

- висока надійність – відмова одного або кількох робіт (агентів) не зупиняє систему;
- масштабованість – додавання нових агентів не створює критичного навантаження;
- гнучкість – система адаптивна до змін у реальному часі;
- зниження затримок у прийнятті рішень.

Основні недоліки:

- обмежена ефективність – через неповну інформацію рішення можуть бути субоптимальними;

– складність координації – особливо в задачах, що потребують взаємодії між роботами;

– вища складність реалізації – автономна логіка прийняття рішень у кожного агента потребує додаткової обробки.

У сучасних системах часто застосовуються гібридні підходи, що поєднують централізоване планування з автономністю роботів. Наприклад, центральний модуль може формувати загальний план роботи, а роботи – самостійно виконувати його, адаптуючись до локальних умов.

Обраний підхід значною мірою визначає характеристики всієї системи: її стабільність, продуктивність, швидкість реалізації та простоту підтримки. У рамках даного проєкту реалізованого частково централізовану архітектуру, де сервер відповідає за розподіл задач, але агенти ініціюють комунікацію самостійно, що дозволяє поєднати переваги обох підходів.

1.2.2 Роль алгоритмів призначення в багатокритеріальному середовищі

У системах автоматизованого керування завданнями, особливо в умовах динамічного середовища з великою кількістю агентів, ключову роль відіграє алгоритм призначення задач. Його завдання полягає у виборі оптимального виконавця для кожної задачі на основі множини факторів. У простих випадках це може бути один критерій, наприклад, доступність виконавця. Проте в реальних системах рішення приймаються з урахуванням багатьох параметрів – таких як пріоритет задачі, очікуваний час виконання (ETA), поточне завантаження, та інші. Набір параметрів може зростати з розвитком галузі, що робить задачу розподілу призначення все складнішою і складнішою.

Багатокритеріальне середовище характеризується наявністю декількох цілей, які можуть бути суперечливими. Наприклад, вибір найближчого робота може зменшити час прибуття, але не гарантує, що цей робот має необхідні технічні характеристики. Або ж призначення задачі найменш завантаженому роботу може призвести до збільшення простоїв у інших [6].

У таких умовах прості евристики, як-от випадковий вибір або фільтрація лише за одним параметром, втрачають ефективність. Натомість застосовуються алгоритми, що враховують множину критеріїв одночасно.

Основні підходи включають:

- зважене ранжування (Weighted Scoring) – кожному параметру призначається ваговий коефіцієнт, після чого обчислюється загальна «оцінка» кожного агента;

- фільтрація за обов’язковими та необов’язковими критеріями – задачі фільтрують список можливих виконавців за критичними параметрами, після чого застосовується додаткова логіка для вибору найкращого;

- методи мультикритеріальної оптимізації – наприклад, метод аналізу ієрархій (АНР), метод компромісного ранжування (TOPSIS), тощо;

- машинне навчання – використовується для прогнозування ефективності призначення на основі історичних даних, особливо в складних динамічних середовищах.

Важливою особливістю багатокритеріального підходу є адаптивність – здатність системи змінювати ваги критеріїв або сам алгоритм залежно від ситуації. Наприклад, у пікові години система може надавати перевагу швидкості виконання, а в періоди низького навантаження – енергоефективності або рівномірності використання ресурсів [8].

Крім того, для забезпечення надійності та відмовостійкості системи, алгоритми повинні враховувати поточні та прогнозовані стани роботів, можливі збої у виконанні задач, а також механізми повторного розподілу у разі недоступності виконавця.

Вони мають бути здатні адаптуватися до змін у середовищі та вміти оперативно перебудовувати план дій. Таким чином, правильний вибір та реалізація алгоритмів призначення є критичним чинником для забезпечення ефективності, стабільності й масштабованості багатоагентних систем у багатокритеріальному середовищі та є надзвичайно важливим для забезпечення безперебійної роботи всіх компонентів.

1.2.3 Алгоритми оптимального та евристичного розподілу

Задача ефективного призначення виконавців у системах розподілу завдань належить до класу комбінаційних оптимізаційних задач. Вона може бути вирішена як за допомогою оптимальних алгоритмів, так і шляхом застосування евристик, які забезпечують наближені, але обчислювально менш затратні рішення [12].

Оптимальні алгоритми, такі як метод гілок і меж, динамічне програмування або моделі на основі цілочисельного лінійного програмування (ILP), гарантують знаходження найкращого рішення відповідно до заданих критеріїв.

Однак, у випадку великої кількості робіт і задач, вони стають непридатними для застосування в режимі реального часу через високу обчислювальну складність. Натомість евристичні алгоритми пропонують певний компроміс між якістю отриманого рішення та швидкістю обчислення. Вони не гарантують повного глобального оптимуму, але дозволяють отримати прийнятне рішення за обмежений час.

Прикладами таких алгоритмів є:

- жадібні алгоритми (greedy), які обирають найкращий варіант на кожному кроці;
- правила пріоритету (наприклад, спершу найбільш пріоритетні задачі – найменш завантаженим роботам);
- генетичні алгоритми [1];
- алгоритми мурашиних колоній.

Для реальних систем, де важлива швидка реакція на зміни середовища, евристики часто є єдиним практичним вибором. Вони можуть бути розширені додатковими обмеженнями або адаптивними механізмами, що покращують ефективність у конкретному застосуванні.

Таким чином, вибір між оптимальними та евристичними підходами залежить від вимог до точності, швидкості, обсягу даних і доступних обчислювальних ресурсів, а також від окремих потреб сфери застосування, які явно задають яким чином функціонуватиме система.

1.3 Огляд схожих систем

У сфері розподілу завдань між роботами існує низка систем, які вирішують схожі задачі, що дозволяє провести порівняльний аналіз та визначити переваги й недоліки різних підходів.

1.3.1 Система розподілу завдань для прибирання морського пластику

Одним із прикладів сучасної системи інтелектуального розподілу завдань у багатоагентних середовищах є рішення, запропоноване у науковій публікації «*A Novel Multi-Robot Task Allocation Model in Marine Plastics Cleaning Based on Replicated Dynamics*». Система розроблена для групи автономних підводних роботів (AUVs), що виконують задачу збирання пластикових відходів у морському середовищі [2].

Проблема накопичення морського пластику набуває все більшої екологічної значущості. Використання людських ресурсів для прибирання таких відходів є вкрай неефективним через складність доступу до певних районів, небезпечні умови праці та масштаб забруднення. Саме тому залучення групи автономних роботів є доцільним і перспективним рішенням. Однак для ефективної роботи такої групи потрібен механізм, який дозволить розподіляти завдання так, щоб уникати дублювання дій, зменшувати простой, враховувати динамічну зміну умов середовища та особливості кожного робота.

У системі використовується підхід, заснований на динаміці реплікаторів (replicator dynamics) – математичній моделі з еволюційної теорії ігор. Основна ідея полягає в тому, що кожна стратегія (тобто вибір завдання певним роботом) змінює свою ймовірність застосування від «успіху» цієї стратегії в минулому.

У контексті роботів, це дозволяє кожному з них на основі зворотного зв'язку адаптувати свої дії, поступово надаючи перевагу більш ефективним стратегіям вибору завдань. Тобто система умовно «навчається» на минулому досвіді задля прийняття кращих рішень та уникнення помилок у майбутньому.

Модель підтримує гетерогенність роботів – тобто враховується, що різні роботи можуть мати різні параметри та характеристики, наприклад: швидкість, ємність акумулятора, маневровість тощо. Це значно підвищує точність та практичну ефективність системи.

Характеристики системи:

- централізована архітектура – координація дій здійснюється централізовано, через загальну систему розподілу;

- адаптивність – алгоритм самостійно коригує поведінку роботів залежно від динамічних змін середовища або продуктивності виконання.

Недоліки системи:

- централізована архітектура створює єдину точку відмови. У випадку втрати зв'язку з центральним координатором або його виходу з ладу – робота всієї системи буде паралізована;

- висока залежність від якості зворотного зв'язку. Модель на базі динаміки реплікаторів ефективно працює лише при стабільному і своєчасному надходженні інформації про виконання задач. Затримки або втрати даних можуть призводити до некоректного навчання системи;

- високі обчислювальні витрати при великій кількості агентів. Зростання кількості роботів та задач збільшує складність розрахунків, що може уповільнити процес прийняття рішень;

- складність у застосування до завдань з різною природою. Система добре працює у вузько спеціалізованому середовищі (прибирання пластику), однак її перенесення на інші домени потребує адаптації моделей, що ускладнює масштабування.

Розглянута система демонструє ефективне застосування інтелектуального підходу до розподілу завдань у багатокомпонентному середовищі з мобільними роботами. Її сильними сторонами є використання математичних моделей для самонавчання, координація дій між агентами та орієнтація та сталий результат. Водночас централізована архітектура, обмежена адаптивність та специфічність під конкретну задачу (очищення океану) знижують її універсальність і

масштабованість у ширшому контексті. Цей приклад ілюструє потенціал подібних систем, але також підкреслює необхідність розробки гнучких рішень та розширюваних архітектур.

1.3.2 Система Task Allocation in Autonomous Warehouse Robots (Amazon Robotics)

Однією з найвідоміших реалізацій автоматизованого розподілу завдань у багатороботизованому середовищі є система автоматизованого управління автономними складськими роботами, що використовується компанією Amazon (колишня Kiva Systems). Дана система застосовується у десятках логістичних центрів і є еталонним прикладом масштабованої, високопродуктивної платформи управління мобільними роботами [5, 7].

Система складається з сотень до тисяч автономних мобільних роботів, які переміщують стелажі з товарами до операторів або до інших автоматизованих зон. Розподіл завдань відбувається централізовано: головний сервер аналізує вхідні замовлення, стан складу та його інвентаризації, положення кожного окремого робота та завантаженість транспортних усіх маршрутів, за якими переміщуються роботи.

Основні фактори, які враховуються при визначенні задач:

- відстань від робота до цілі;
- поточний статус (виконує чи не виконує завдання);
- залишковий заряд батареї;
- пріоритет обробки замовлення;
- ситуація на маршрутах (конфлікти, затори).

Рух роботів координується у реальному часі за допомогою складного алгоритму уникнення зіткнень, що також є частиною алгоритмічної розподільчої логіки. Роботи отримують нові завдання одразу після завершення попереднього, що знижує час простою до мінімуму. Саме так роботи постійно отримують завдання, і витрати ресурсів зменшуються.

Переваги системи:

- масштабованість – система довела свою здатність працювати із тисячами пристроїв одночасно;
- оптимізація часу виконання – завдяки продуманій маршрутизації та призначенню задач;
- низький рівень простоїв – завдання призначаються із врахуванням майбутньої доступності робітв;
- інтеграція з ERP/WMS – дозволяє працювати у великій корпоративній екосистемі.

Недоліки системи:

- замкнута архітектура – система розроблена для внутрішнього використання і недоступна для широкого впровадження в інших сферах;
- централізована координація – високий ступінь залежності від центрального сервера створює потенційну точку відмови;
- складна адаптація до інших доменів – специфіка складської логістики ускладнює використання у виробництві, сервісі чи мобільних зовнішніх середовищах;
- вартість впровадження – надзвичайно високі інвестиції, необхідні для повного розгортання інфраструктури.

Система Amazon Robotics є прикладом надзвичайно ефективного централізованого розподілу завдань у контрольованому середовищі. Вона забезпечує високу продуктивність і мінімізацію простоїв у логістичних процесах. Водночас замкнутість архітектури, складність адаптації до інших предметних областей та висока вартість впровадження обмежують її застосування поза межами великих комерційних складів. Ефективність системи значною мірою зумовлена суворою регламентацією середовища, де усі змінні жорстко контрольовані. Такий підхід дозволяє досягати високої швидкості виконання операцій, але значно ускладнює адаптацію до нестандартизованих або відкритих середовищ [9]. Але досвід системи, попри свою вузьку спеціалізацію залишається цінним джерелом інженерних рішень та оптимізаційних стратегій.

1.3.3 Децентралізований жадібний алгоритм розподілу завдань

Децентралізовані жадібні алгоритми (Greedy Task Allocation) є досить популярним підходом у багатороботних (багатоагентних) системах, особливо коли неможливо забезпечити повноцінне централізоване керування через обмеження зв'язку з виконавцями, необхідність масштабованості або автономності роботів. У таких системах кожен робот самостійно приймає зважене рішення про прийняття задачі на основі повної локальної інформації, таких як власне поточне навантаження, стан, відстань до цілі, пріоритет задачі та інших параметрів, які впливають на вибір [4].

Сутність підходу полягає в тому, що:

- кожен робот отримує інформацію про доступні задачі (через ширококомове повідомлення або централізовану публікацію списку);
- виконує оцінку «корисності» кожної задачі з власної точки зору (наприклад, мінімальна відстань чи час виконання);
- надсилає запит на виконання задачі з найвищою корисністю;
- у разі конфлікту між кількома роботами, задача присвоюється найефективнішому претенденту.

Такий підхід використовується, зокрема, у роботизованих системах для складів, доставлення товарів (наприклад, дрони, пересувні платформи і т.д.) та рятувальних операціях, де важливо зменшити затримки у прийнятті рішень і забезпечити автономність дій.

Підхід дозволяє швидко приймати рішення без потреби у централізованій координації.

Переваги такого підходу:

- простота реалізації та висока швидкодія;
- висока масштабованість системи – зростання кількості роботів не призводить до критичного збільшення складності;
- стійкість до відмов – вихід з ладу одного робота не впливає на загальну працездатність системи.

Недоліки:

- відсутність глобального бачення може призводити до неефективного розподілу (наприклад, кілька роботів вибирають близькі або швидкі задачі, ігноруючи більш термінові);
- складність синхронізації в умовах обмеженого або нестабільного зв'язку;
- можливі конфлікти при виборі задач, які потребують додаткових механізмів розв'язання.

Децентралізований жадібний підхід до розподілу завдань демонструє високу ефективність у системах із великою кількістю виконавців та обмеженим централізованим контролем. Його основними перевагами є автономність, масштабованість і швидке прийняття рішень, однак ціною цього є потенційна втрата глобальної оптимізації. Такий підхід добре підходить для динамічних середовищ, але вимагає механізмів для уникнення конфліктів та дублювання дій.

У процесі аналізу наявних рішень для розподілу завдань у роботизованих системах було розглянуто як комерційні продукти, так і академічні підходи. Системи на кшталт RoboFleet демонструють потужну архітектуру із можливостями централізованого керування та візуалізації; однак часто обмежуються фіксованими сценаріями та потребують складної конфігурації. Інші рішення, як-от ROS-based multi-robot task allocation, орієнтовані більше на дослідницькі цілі та вимагають глибоких технічних знань, залишаючи значну частину інтеграції та логіки на відповідальність розробника. У свою чергу, децентралізовані жадібні алгоритми пропонують простоту реалізації та добру масштабованість, але поступаються в плані глобальної оптимальності та передбачуваності поведінки системи [15].

Висновки до розділу 1

Предметна область розподілу завдань у багатоагентних роботизованих системах охоплює широкий спектр технічних викликів, пов'язаних з координацією дій автономних агентів у спільному середовищі. Основна проблема полягає у тому, щоб забезпечити ефективне призначення задач з урахуванням обмежених ресурсів,

					ІК11.270БАК.006 ПЗ	Арк.
						20
Зм.	Лист	№ докум.	Підпис	Дата		

динаміки середовища, а також інших параметрів, таких як пріоритет, складність та тривалість завдання [10].

Сучасні системи розподілу завдань поділяються на централізовані, децентралізовані та гібридні. Централізовані підходи забезпечують високу оптимальність розподілу та глобальне бачення системи, однак мають обмежену відмовостійкість і погано масштабуються при великій кількості агентів. Децентралізовані системи, навпаки, дозволяють кожному агенту діяти автономно, що покращує гнучкість і знижує ризик єдиної точки відмови, але водночас ускладнює досягнення глобальної оптимальності.

Проаналізовані приклади – як централізована система Amazon Robotics, так і децентралізовані рішення, що використовують жадібні алгоритми – демонструють, що вибір архітектури залежить від особливостей предметної області. Наприклад, для контрольованих складських середовищ ефективно працює централізоване управління, тоді як для рятувальних операцій або розподілених дрон-систем більш доцільними є децентралізовані підходи.

Окрему увагу слід приділяти ролі алгоритмів призначення в умовах багатокритеріальності, де доводиться балансувати між пріоритетами, енерговитратами, часом виконання та іншими факторами. У таких умовах використовуються як оптимізаційні, так і евристичні алгоритми, зокрема жадібні та аукціонні методи.

Крім того, аналіз реальних прикладів показав, що сучасні системи дедалі частіше поєднують елементи як централізованої, так і децентралізованої архітектур, утворюючи гібридні рішення. Такий підхід дозволяє зберегти контроль над критично важливими процесами, залишаючи при цьому достатню гнучкість для адаптації на рівні окремих агентів.

Таким чином, аналіз предметної області показав, що не існує абсолютно універсального та ідеального рішення для всіх типів задач – кожна система потребує адаптації під конкретні вимоги середовища, через що зростає складність реалізації таких систем.

2 ВИМОГИ ДО СИСТЕМИ

2.1 Бізнес-вимоги

2.1.1 Основні функціональні вимоги

Інтелектуальна система розподілу завдань у роботизованих системах створюється з метою підвищення ефективності виконання технічних засобів, зменшення простоїв та оптимізації процесу виконання завдань роботами (агентами, виконавцями). Основна функція полягає в автоматичному призначенні завдань відповідним виконавцям на основі їхнього поточного стану та стану системи, функціональних можливостей, залишкового навантаження, апаратних та технічних характеристик.

Система повинна забезпечувати такі функціональні можливості:

- збір інформації про стан кожного робота (зайнятість, ETA, поточне завдання, технічні параметри);
- прийом нових задач і збереження їх у черзі з відповідними метаданими (пріоритет, складність, обов'язкові та необов'язкові вимоги);
- призначення задач роботам відповідно до обраного алгоритму розподілу;
- можливість гнучко модифікувати алгоритми та їх параметрів відповідно до особливостей середовища;
- надання агентам (роботам) інтерфейсу API для ініціалізації запиту нової задачі та оновлення статусу виконання;
- інтерфейс для користувачів та адміністраторів з метою перегляду активних задач, стану роботів, журналу розподілу та ручного втручання;
- логування усіх дій, пов'язаних із призначенням задач, для подальшого аналізу;
- масштабованість для підтримки зростаючої кількості роботів і задач без втрати продуктивності.

Таким чином, система має бути достатньо гнучкою, щоб адаптуватися до різних умов експлуатації, та водночас стабільною і надійною для забезпечення безперервної роботи в реальному часі [11].

					ІК11.270БАК.006 ПЗ	Арк.
						22
Зм.	Лист	№ докум.	Підпис	Дата		

2.1.2 Вимоги до взаємодії користувача з системою

Для ефективного використання системи розподілу завдань у роботизованому середовищі передбачено низку вимог до взаємодії кінцевого користувача з інтерфейсом. Інтерфейс має бути інтуїтивно зрозумілим, адаптивним до різних пристроїв та забезпечувати швидкий доступ до ключової інформації про стан системи, роботів і задач.

Основні вимоги:

- огляд статусу системи в реальному часі. Користувач повинен мати можливість переглядати поточний стан кожного робота (зайнятість, статус задачі, ETA, тощо), історію виконаних задач та загальну аналітику;
- візуальне відображення черг задач. Необхідно реалізувати табличний або графічний інтерфейс для перегляду черги задач кожного робота з можливістю редагування та перенесення задач;
- інструменти ручного втручання; Користувач повинен мати змогу вручну перерозподіляти задачі між роботами, та мати можливість назначати конкретних виконавців для завдання на етапі створення;
- система сповіщень. Користувач повинен отримувати повідомлення про критичні події, збої у виконанні задач, неактивність роботів та інші типи сповіщень, необхідні для коректного контролю системи;
- багаторівневий доступ. Система повинна підтримувати ролі користувачів (наприклад: адміністратор, оператор, аналітик) з відповідними обмеженнями доступу до функціоналу. Це забезпечить надійний рівень безпеки від несанкціонованого доступу до критично важливого функціоналу системи;
- журнал подій та звітність. Для відстеження рішень, які приймаються системою або користувачем, необхідно реалізувати журнал подій з можливістю перегляду.

Ці вимоги спрямовані на забезпечення повного контролю над процесами розподілу задач, зручності управління та підвищення надійності роботи всієї системи.

					ІК11.270БАК.006 ПЗ	Арк.
						23
Зм.	Лист	№ докум.	Підпис	Дата		

2.1.3 Вимоги до масштабованості та адаптивності

Оскільки система призначена для керування динамічним середовищем із потенційно великою кількістю роботизованих агентів (виконавців), вона повинна бути здатною масштабуватись як горизонтально, так і вертикально. Крім того, важливо передбачити адаптивність до змін у середовищі, параметрах задач або конфігураціях роботів. Саме масштабованість задає подальше існування та покращення роботи системи.

Основні вимоги:

- підтримка збільшення кількості роботів без значної втрати продуктивності. Система має стабільно працювати при підключенні сотень або тисяч роботів, або навіть більше, забезпечуючи своєчасне призначення задачі і обробку запитів;
- гнучкість у структурі задач. Архітектура має дозволяти додавання різних задач, атрибутів і логік без потреби значних змін у коді;
- модульність системи. Компоненти (API, планувальник, аналітика, база даних, інтерфейс) повинні бути відокремленими і мати чіткі точки взаємодії, що дозволить легко змінювати або розширювати функціональність в майбутньому;
- адаптація до конфігурацій роботів. Система повинна враховувати різні технічні характеристики роботів (наприклад вантажопідйомність, автономність, швидкість) та динамічно оновлювати ці дані при зміні параметрів;
- сумісність із зовнішніми сервісами. Можливість інтеграції з іншими системами управління, логістики, аналітики, IoT-платформами, або іншими платформами, які можуть бути потенційно важливими в майбутньому;
- стійкість до збоїв. Повинна бути реалізована обробка збоїв в обміні повідомленнями, відсутності зв'язку з роботами або непередбачених зупинок системи, з механізмами автоматичного відновлення.

Ці вимоги забезпечують довготривалу актуальність, надійність і зручність розвитку даної інтелектуальної системи розподілу задач в умовах швидко зростаючих обсягів даних, гетерогенності агентів та їх властивостей і динамічних змін середовища.

2.2 Архітектурні вимоги

Проектована система розподілу завдань для роботизованих систем повинна бути побудована за принципами модульності, масштабованості та гнучкості. Основою архітектури є класична клієнт-серверна модель, яка передбачає чітке розділення обов'язків між різними компонентами.

Система повинна складатися з трьох основних частин:

- клієнтський інтерфейс – реалізується як веб-додаток на Angular, або можливе використання мобільних версій додатку. Призначено для операторів системи, які мають доступ до моніторингу, перегляду завдань, ручного керування тощо;
- серверна логіка (API) – реалізована на платформі .NET, відповідає за логіку розподілу завдань, взаємодію з роботами, обробку даних, валідацію та аналітику;
- база даних – PostgreSQL. Зберігає інформацію про роботів, задачі, історію виконання, метрики навантаження та інші службові дані.

Для забезпечення взаємодії між агентами (роботами) та сервером використовуються REST API. Основні особливості:

- підтримка HTTP методів: GET, POST, PUT, DELETE;
- JSON як стандартний формат обміну даними;
- документовані маршрути з чіткими правилами валідації.

Серверна частина повинна підтримувати асинхронну обробку запитів для забезпечення високої продуктивності при великій кількості одночасних звернень (роботи можуть паралельно ініціювати запити на отримання задачі або оновлення статусу).

Роботи виступають як незалежні агенти, які самостійно ініціюють взаємодію з сервером. Це забезпечує більшу гнучкість і надійність у випадку відмов зв'язку або часткових збоїв.

Алгоритми розподілу реалізовані у вигляді окремих компонентів (класів) з можливістю легкої заміни або додавання нових стратегій. Це забезпечує надзвичайну можливість до розширюваності системи відповідно до різних

сценаріїв та алгоритмів розподілу задач (випадковий вибір, навантаження, ЕТА тощо).

Архітектура має передбачати горизонтальне масштабування, тобто можливість запуску декількох екземплярів API або клієнта без зміни логіки системи.

2.3 Вимоги до безпеки

Безпека є критично важливим аспектом для будь-якої розподіленої системи, особливо коли йдеться про автоматизоване керування роботами, обробку важливих даних і взаємодію з кількома користувачами чи системами. У межах системи необхідно забезпечити належний рівень для захисту від несанкціонованого доступу, витоку даних та інших загроз.

2.3.1 Аутентифікація та авторизація користувачів

Аутентифікація та авторизація користувачів є основними механізмами для забезпечення доступу до системи лише для авторизованих осіб. Всі користувачі повинні пройти процедуру ідентифікації та аутентифікації перед отриманням доступу до бажаного функціоналу. Для підвищення рівня безпеки рекомендовано використовувати різні сучасні підходи, наприклад багатофакторна аутентифікація (MFA). Це дозволяє значно знизити ризик несанкціонованого доступу через крадіжку пароля, або інших типів атак, спрямованих на отримання даних користувачів.

Система має підтримувати можливість встановлення ролей та доступів, де кожен користувач має доступ лише до тих функцій, які відповідають його ролі (наприклад, адміністратор, оператор чи користувач).

Такий підхід дозволяє зменшити ризик людських помилок і забезпечує дотримання коректних доступів до різних частин системи. Детальна діаграма ролей та доступів користувачів зображена на кресленику ІК11.270БАК.006 Д4.

					ІК11.270БАК.006 ПЗ	Арк.
						26
Зм.	Лист	№ докум.	Підпис	Дата		

2.3.2 Захист API

API-система є серцем архітектури, оскільки робить систему гнучкою та дозволяє взаємодіяти з сервером. Однак саме API часто є вразливим до атак, таких як SQL-injections, DDoS-атаки або зламування сесій.

Для належного захисту серверу рекомендуються такі практики:

- API-ключі. Для кожного запиту що надходить до системи від робота (або групи роботів), має бути використаний унікальний API-ключ для аутентифікації;
- bearer-токени. Для кожного запиту що надходить від клієнтської частини, має бути використаний правильний токен з метою аутентифікації та ідентифікації користувача;
- шифрування. Запити повинні передаватись через зашифроване з'єднання (протокол HTTPS з SSL-сертифікатами).

2.3.3 Шифрування конфіденційних даних

Зберігання чутливих даних (наприклад, особистих даних користувачів, ідентифікаторів роботів та інших критичних параметрів) має бути здійснене у зашифрованому вигляді.

Усі персональні дані та важливі параметри мають бути зашифровані перед збереженням у базі даних за допомогою сучасних алгоритмів, таких як AES-256. Усі дані, що передаються між клієнтом і сервером, повинні бути зашифровані для запобігання їх перехопленню.

2.3.4 Захист від атак

Захист системи від різноманітних атак є необхідною, важливою та невід'ємною умовою для забезпечення її стабільності, безпечної роботи та відмовостійкості. Особливу увагу слід приділити різним прогресуючим в наш час атакам типу DoS, CSRF, SQL-injections, так як в сучасному світі вони становлять

реальну загрозу для критично важливих систем, адже можуть бути спрямовані на знищення чи пошкодження дних.

Рекомендуються такі практики:

– захист від DDoS-атак: Використання спеціалізованих технологій для виявлення і блокування масових атак на сервери системи, зокрема через CDN-сервіси;

– захист проти SQL-ін'єкцій та XSS-атак: Система має бути захищена від ін'єкцій в запити до баз даних та атак через міжсайтове використання скриптів, використовуючи параметричні запити та безпечне зберігання даних у формах.

Отже, система повинна відповідати ряду бізнесових та технічних вимог, щоб забезпечити ефективне та безпечне розподілення завдань між роботами. Основні бізнес-вимоги включають автоматичне призначення завдань відповідно до технічних характеристик роботів, їхнього навантаження та пріоритетів завдань, а також інтеграцію з іншими системами для моніторингу та управління.

Технічні вимоги охоплюють необхідність використання масштабованих архітектурних рішень, підтримку різних задач, а також забезпечення високої надійності та ефективності роботи системи. Важливою частиною є вимоги до апаратного забезпечення, зокрема підтримка сумісності з різними типами роботів та платформ.

Вимоги до безпеки передбачають захист від несанкціонованого доступу, аутентифікацію користувачів, шифрування даних та регулярний моніторинг безпеки. Всі ці вимоги забезпечують безперебійну та безпечну роботу системи, а також високу надійність при виконанні завдань у реальному часі.

Висновки до розділу 2

Описаний спектр вимог до системи розподілу завдань у багатоагентному роботизованому середовищі охоплює ключові аспекти її проектування та впровадження. Ретельна формалізація бізнес-вимог дозволяє забезпечити ефективну та надійну роботу системи відповідно до потреб області застосування і

					ІК11.270БАК.006 ПЗ	Арк.
						28
Зм.	Лист	№ докум.	Підпис	Дата		

специфіки цільових сценаріїв. Зокрема, серед основних вимог визначено здатність системи керувати чергами задач, контролювати виконання завдань роботами, підтримувати їхню пріоритезацію, що доволі важливо для реальних сценаріїв.

Суттєвим компонентом функціональності є зручна взаємодія користувача з системою. Інтерфейс повинен не лише відображати актуальну інформацію у реальному часі, але й давати змогу оперативно втручатися в розподіл задач, переназначати їх, переглядати аналітику та отримувати сповіщення. Підтримка ролей з різними рівнями доступу створює передумови для чіткого розмежування відповідальності та контролю за діями користувачів.

Вимоги до масштабованості та адаптивності системи відображають її здатність функціонувати в умовах змінного навантаження, змін конфігурації роботів, збільшення кількості задач і тп.

Технічні вимоги задають архітектурну основу системи, орієнтовану на модульний підхід, який полегшує супровід та масштабування.

Вимог до безпеки мають критичне значення у контексті взаємодії із зовнішніми мережами, доступу до конфіденційних даних і забезпечення цілісності системи. Передбачено механізми аутентифікації, авторизації з урахуванням ролей, шифрування даних, захисту API та протидії типових атак. Система має бути готова до реагування на інциденти та забезпечувати високу стійкість до зовнішніх загроз.

Таким чином, сформульовані вимог забезпечують повний опис системи та формують фундамент для її подальшого проєктування, розробки та тестування. Вони охоплюють як бізнес-потреби користувачі, так і технічні та безпекові аспекти, які необхідні для стабільної, масштабованої та захищеної роботи системи у складному багатоагентному середовищі.

3 РОЗРОБЛЕННЯ СИСТЕМИ

3.1 Проектування архітектури системи

3.1.1 Загальні принципи

Для розробки ефективної системи автоматизованого розподілу завдань у багатоагентних роботизованих системах, було прийнято певні архітектурні рішення, що забезпечують стабільність, гнучкість, відмовостійкість та ефективність роботи. Ось основні принципи, на яких базується архітектура реалізації системи:

– модульність та розширюваність: система побудована на модульному, де кожен компонент виконує чітко визначену функцію. Це дозволяє, наприклад, безперешкодно оновлювати алгоритми розподілу завдань або змінювати логіку роботи з базою даних, не порушуючи функціональність інших частин системи. Кожен модуль реалізується як окремий сервіс або клас, який можна легко модифікувати чи замінити, без необхідності зміни решти системи;

– масштабованість: система спроектована з урахуванням можливості масштабування для роботи з великими обсягами даних і збільшення кількості робіт. Для цього система підтримує горизонтальне масштабування серверів, що дає змогу додавати нові вузли в систему при збільшенні навантаження. Для забезпечення високої продуктивності використовується розподілене зберігання даних на основі PostgreSQL, що дозволяє ефективно працювати з великими обсягами інформації, навіть якщо система зростає в кілька разів;

– безпека: оскільки система працює з конфіденційними даними, включаючи завдання, їхній стан і пріоритети, було впроваджено багаторівневу систему захисту. Для взаємодії між роботами і сервером застосовуються механізми автентифікації за токенами, ключами, а також шифрування даних на всіх етапах їхнього пересування по мережі. Це гарантує захист від несанкціонованого доступу і зловмисників;

– гнучкість алгоритмів: однією з ключових особливостей архітектури є можливість інтеграції різних алгоритмів для розподілу завдань. Алгоритми можуть бути динамічно змінені або додані, що дає змогу адаптувати систему до різних типів

					ІК11.270БАК.006 ПЗ	Арк.
						30
Зм.	Лист	№ докум.	Підпис	Дата		

задач і характеристик робіт. Також алгоритми можуть бути гнучко налаштовані адміністратором системи, що впливає на їх подальшу роботу.

Таким чином, архітектура поточної системи є гнучкою, масштабованою та надійною, що дозволяє адаптувати її до змінних умов з роботами, їхнім навантаженням та різними типами завдань, забезпечуючи при цьому високу продуктивність і безперебійну роботу на великих обсягах даних.

3.1.2 Архітектура системи

Архітектура системи автоматизованого розподілу завдань для роботизованих систем побудована на основі принципів модульності, масштабованості. Усі компоненти системи мають чітке розмежування відповідальностей, що дозволяє забезпечити зручну адаптацію та розширення функціоналу за потреби.

Архітектура системи включає в себе три основні шари:

– шар збору даних та обробки завдань: відповідає за отримання даних про завдання, поточний стан робіт, їхню завантаженість і доступність для виконання нових завдань. Цей шар забезпечує обробку запитів від робіт через API та фільтрацію завдань за критеріями, які визначають, який робот найкраще підходить для їх виконання;

– шар розподілу завдань та керування: центральний компонент системи, що здійснює вибір робота для виконання завдання на основі алгоритмів розподілу. Цей шар відповідає за обчислення оптимальних шляхів виконання завдань, а також за контроль та моніторинг виконання завдань роботами;

– шар аналітики та візуалізації: цей компонент дозволяє користувачам і адміністраторам переглядати статус завдань, оцінювати навантаження робіт у реальному часі та здійснювати прогнозування щодо потреби в додаткових роботах чи коригуванні алгоритмів розподілу завдань. Візуалізація забезпечує зручний інтерфейс для моніторингу процесів.

Архітектура розробленої системи дозволяє досягти високої гнучкості та адаптивності в обробці завдань роботами, а також забезпечує надійність і

масштабованість у разі збільшення кількості робіт або складності завдань. Всі компоненти системи взаємодіють через стандартні інтерфейси та легко інтегруються з іншими бізнес-системами. Детальна схема реалізації архітектури зображена на кресленику ІК11.270БАК.006 Д1.

3.1.3 Архітектурна реалізація серверної та клієнтської частини

У процесі розробки системи розподілу завдань для роботизованих систем було прийнято рішення дотримуватись принципів Clean Architecture для серверної частини та чіткої модульної структури для клієнтської. Це дозволяє забезпечити масштабованість, підтримку та незалежність модулів один від одного, спрощуючи тестування та майбутній розвиток системи.

Серверна частина реалізована з використанням технологій сучасних та прогресивних технологій: .NET 8, Entity Framework для роботи з базою даних, та MediatR для організації внутрішньої взаємодії між компонентами. Загальна структура відповідає принципам Clean Architecture, де код поділений на кілька шарів із чіткими зонами відповідальності.

Основні шари:

- Domain: містить сутності (Entities), Value Objects, перерахування (Enums), абстракції для роботи з базою даних, константи (Consts), та деякі абстракції сервісів системи (рисунки 3.1);

- Application: містить логіку взаємодії із зовнішніми системами у вигляді команд (Commands) та запитів (Queries), реалізованих через бібліотеку MediatR. Також тут розміщені DTO, валідатори (FluentValidation), інтерфейси сервісів (рисунки 3.2);

- Infrastructure: реалізує зовнішні залежності – роботу з базою даних, API, файловими системами. Тут міститься реалізація абстракцій із шарів Domain та Application (рисунки 3.3);

- Web: містить контролери (Controllers), конфігурацію додатку, DI-контейнер, middlewares, OpenAPI документацію, тощо (рисунки 3.4).

					ІК11.270БАК.006 ПЗ	Арк.
						32
Зм.	Лист	№ докум.	Підпис	Дата		

Abstractions	09.04.2025 20:49	Папка файлів	
Algorithms	09.04.2025 20:49	Папка файлів	
bin	09.04.2025 20:47	Папка файлів	
Capabilities	09.04.2025 20:49	Папка файлів	
Chatting	09.04.2025 20:49	Папка файлів	
Consts	09.04.2025 20:50	Папка файлів	
Logging	19.04.2025 21:43	Папка файлів	
obj	27.04.2025 15:54	Папка файлів	
Repositories	26.04.2025 18:28	Папка файлів	
Robots	19.04.2025 21:36	Папка файлів	
Services	09.04.2025 20:52	Папка файлів	
Tasks	27.04.2025 15:48	Папка файлів	
Tenants	26.04.2025 17:18	Папка файлів	
Nerbotix.Domain.csproj	19.04.2025 17:05	C# Project File	1 КБ

Рисунок 3.1 – Архітектура шару Domain

Algorithms	09.04.2025 20:57	Папка файлів	
Analytics	27.04.2025 17:01	Папка файлів	
ApiKeys	21.04.2025 14:48	Папка файлів	
Auth	13.04.2025 20:28	Папка файлів	
BackgroundJobs	19.04.2025 20:02	Папка файлів	
bin	17.02.2025 14:48	Папка файлів	
Chatting	09.04.2025 20:58	Папка файлів	
Common	16.03.2025 17:10	Папка файлів	
obj	27.04.2025 15:54	Папка файлів	
Robots	19.04.2025 18:20	Папка файлів	
Roles	15.03.2025 14:43	Папка файлів	
Services	21.04.2025 15:15	Папка файлів	
SuperAdmin	26.04.2025 17:09	Папка файлів	
User	16.04.2025 21:25	Папка файлів	
Users	09.04.2025 21:13	Папка файлів	
Nerbotix.Application.csproj	28.04.2025 21:12	C# Project File	2 КБ
RegisterDependencies.cs	09.04.2025 20:57	Файл CS	1 КБ

Рисунок 3.2 – Архітектура шару Application

Algorithms	30.03.2025 20:14	Папка файлів	
Authentication	19.04.2025 20:52	Папка файлів	
bin	17.02.2025 14:48	Папка файлів	
Chatting	09.04.2025 21:19	Папка файлів	
Data	21.04.2025 14:21	Папка файлів	
Emailing	13.04.2025 21:44	Папка файлів	
Hangfire	19.04.2025 20:08	Папка файлів	
Migrations	27.04.2025 15:54	Папка файлів	
obj	27.04.2025 15:54	Папка файлів	
Robots	27.04.2025 14:40	Папка файлів	
Services	21.04.2025 14:43	Папка файлів	
Storage	09.04.2025 21:29	Папка файлів	
Nerbotix.Infrastructure.csproj	21.04.2025 14:37	C# Project File	2 КБ
RegisterDependencies.cs	21.04.2025 14:43	Файл CS	10 КБ

Рисунок 3.3 – Архітектура шару Infrastructure

Attributes	09.04.2025 21:34	Папка файлів	
bin	09.04.2025 20:47	Папка файлів	
Controllers	28.04.2025 21:14	Папка файлів	
Extensions	13.04.2025 19:30	Папка файлів	
Infrastructure	09.04.2025 21:34	Папка файлів	
Models	09.04.2025 20:47	Папка файлів	
obj	27.04.2025 15:54	Папка файлів	
Properties	09.04.2025 20:47	Папка файлів	
Services	13.04.2025 23:08	Папка файлів	
SuperAdmin	26.04.2025 18:27	Папка файлів	
appsettings.Development.json	13.04.2025 22:32	JSON File	1 КБ
appsettings.json	27.04.2025 13:58	JSON File	2 КБ
appsettings.Production.json	27.04.2025 13:59	JSON File	1 КБ
Dockerfile	21.04.2025 20:44	Файл	1 КБ
MapsterConfig.cs	09.04.2025 21:35	Файл CS	1 КБ
Nerbotix.Api.csproj	13.04.2025 19:25	C# Project File	2 КБ
Program.cs	27.04.2025 13:59	Файл CS	1 КБ
RegisterDependencies.cs	27.04.2025 14:00	Файл CS	3 КБ

Рисунок 3.4 – Архітектура шару Web

Саме завдяки даній архітектурі було створене чітке та зрозуміле розділення відповідальностей на шари, що значно пришвидшило розробку повноцінної системи.

Особливості реалізації:

- для обробки запитів використовується MediatR – це дозволяє розділити логіку контролерів від бізнес-логіки та зробити код менш зв’язним;
- для валідації – FluentValidation. Валідація команд та запитів відбувається до потрапляння логіки в обробники;
- впроваджено логування;
- застосовано патерн Repository.

Клієнтська частина на Angular 19, з використанням Signal API, RxJS, PrimeNg. Вся структура організована за функціональним принципом (feature-based structure), що дозволяє легко та швидко масштабувати проєкт.

Особливості реалізації:

- State Management – локально через Signals, без глобального стану, завдяки простоті сценаріїв;
- HTTP-комунікація – сервісна модель з окремими сервісами, які інкапсулюють запити до серверу;

– адаптивність інтерфейсу – використання бібліотеки компонентів PrimeNg забезпечує чисту та адаптивну розмітку;

– компоненти реального часу – через механізм WebSockets реалізовано деякі компоненти, дані яких оновлюються в режимі реального часу.

Така архітектурна організація дозволила розділити логіку системи на незалежні модулі, забезпечити її тестованість та масштабованість. Завдяки використанню MediatR, Clean Architecture та модульного підходу у клієнті, система легко адаптується до нових вимог і може бути доповнена без втрати стабільності. Детальна діаграма класів зображена на кресленику ІК11.270БАК.006 ДЗ.

3.2 Серверна частина

3.2.1 Реєстрація та аутентифікація користувачів

Реєстрація та аутентифікація є фундаментальними компонентами системи, які забезпечують захищений доступ до її функціоналу. У реалізації використовуються два підходи аутентифікації: JWT токени для користувачів клієнтської частини та API Keys для роботів.

Процес реєстрації реалізовано через окремий API – ендпоінт, який викликається адміністратором компанії через клієнтську частину. Адміністратор має змогу запрошувати нового користувача доєднатись до компанії.

Для входу в систему реалізовано окремий ендпоінт /auth/login (рисунок 3.5), який перевіряє відповідність облікових даних. У випадку успішної аутентифікації система генерує JWT-токен, що містить закодовану інформацію про користувача (Id, роль, пошта тощо) і видає його клієнту. Токен прикріплюється до кожного запиту користувача як заголовок запиту (Authorization: Bearer <token>), що дозволяє системі розпізнати користувача без необхідності повторної авторизації.

Крім access-токену, генерується також refresh-токен, який зберігається у базі даних та використовується для автоматичного оновлення токену доступу при завершенні його строку дії. Якщо строк дії токену оновлення вийшов, користувач має знову ввести свої дані для входу.

					ІК11.270БАК.006 ПЗ	Арк.
						35
Зм.	Лист	№ докум.	Підпис	Дата		

Таким чином, механізм реєстрації та аутентифікації забезпечує надійний контроль доступу до ресурсів системи.

Auth		^
POST	/Auth/new-company	🔒 ⌵
POST	/Auth/login	🔒 ⌵
POST	/Auth/forgot-password	🔒 ⌵
POST	/Auth/reset-password	🔒 ⌵
POST	/Auth/change-password	🔒 ⌵
POST	/Auth/refresh-token	🔒 ⌵
POST	/Auth/register	🔒 ⌵

Рисунок 3.5 – Ендпоїнти механізму аутентифікації та реєстрації

3.2.2 Керування користувачами, ролями та доступами

Система реалізує багаторівневу модель керування правами доступу на основі ролей (RBAC – Role-Based Access Control). Це дозволяє забезпечити гнучке обмеження функціональності залежно від призначеної ролі користувача.

На поточному етапі реалізовано декілька системних ролей:

- адміністратор – має повний доступ до всіх функцій системи;
- користувач – має обмежений доступ, визначений роллю.

Кожному користувачу при створенні або редагуванні облікового запису призначається відповідна роль (рисунок 3.6). Зміна ролі можлива тільки адміністраторами через окремі ендпоїнти (рисунок 3.7).

У серверній частині доступ до функціоналу контролюється за допомогою атрибутів авторизації.

Users		^
GET	/Users	🔒 ⌵
POST	/Users	🔒 ⌵
GET	/Users/{id}	🔒 ⌵
PUT	/Users/{id}	🔒 ⌵
DELETE	/Users/{id}	🔒 ⌵

Рисунок 3.6 – Керування користувачами

Roles		^
POST	/Roles	🔒 ▼
GET	/Roles	🔒 ▼
GET	/Roles/{id}	🔒 ▼
PUT	/Roles/{id}	🔒 ▼
DELETE	/Roles/{id}	🔒 ▼

Рисунок 3.7 – Керування ролями

3.2.3 Конфігурація системи

Підсистема конфігурації дозволяє адміністраторам налаштовувати параметри, які впливають на роботу системи розподілу завдань та взаємодію з роботами через API. Для забезпечення безпечної взаємодії між роботами та сервером кожному агенту призначається унікальний API-ключ (рисунок 3.8). Під час кожного запиту до серверу ключ верифікується, і тільки після цього робот отримує відповідь.

Система підтримує декілька алгоритмів розподілу задач, і майже кожен з них може мати власний набір конфігураційних параметрів (рисунок 3.9). Наприклад для мурашиного алгоритму доступні такі параметри:

- Iterations – кількість ітерацій симуляції;
- Evaporation Rate – коефіцієнт зменшення феромонів;
- Alpha / Beta – параметри, що визначають вагу феромонів і евристики;
- Colony Size – кількість віртуальних агентів (мурашок).

Завдяки цій підсистемі, система залишається гнучкою і масштабованою: адміністратори можуть адаптувати поведінку розподілу без необхідності вносити зміни у код або повторно запускати сервер.

ApiKeys		^
POST	/ApiKeys	🔒 ▼
GET	/ApiKeys	🔒 ▼
DELETE	/ApiKeys/{id}	🔒 ▼

Рисунок 3.8 – Керування Арі ключами



Рисунок 3.9 – Керування конфігурацією алгоритмів

3.2.4 Створення категорій робіт

Система підтримує класифікацію робіт за категоріями, що дозволяє групувати їх за призначенням, функціональністю або іншими логічними ознаками. Категорії є базовим елементом при розподілі завдань, оскільки кожна задача належить до певної категорії і має бути призначена лише тим роботам, які до неї належать. Це гарантує призначення задач лише тим виконавцям, які можуть виконати дану задачу.

Основні можливості модуля:

- створення та редагування категорій – кожна категорія має назву, опис а також набір параметрів, які будуть притаманні кожному роботу з категорії;
- прив’язка робіт до категорії – кожен робот може належати до однієї або кількох категорій. Це гнучкий підхід, що дозволяє враховувати багатофункціональність деяких пристроїв;
- обмеження доступності задачі – при створенні обирається категорія, і лише роботи з відповідною приналежністю можуть претендувати на її виконання. Це виключає випадкове призначення задач невідповідним агентам.

На рівні серверної частини категорії реалізовані як окрема сутність у базі даних з відношенням багато-до-багатьох робіт. Це забезпечує швидкий пошук та фільтрацію. Для роботи з категоріями реалізовано окремий контролер (рисунок 3.10) з CRUD-операціями, захищений через механізм ролей. Завдяки модулю категорій система отримує можливість гнучко адаптуватися під різні типи задач і сценарії виконання, підвищуючи ефективність і точність розподілу.

Categories		^
POST	/Categories	🔒 ▼
GET	/Categories	🔒 ▼
DELETE	/Categories/{id}	🔒 ▼
GET	/Categories/{id}	🔒 ▼
PUT	/Categories/{id}	🔒 ▼

Рисунок 3.10 – Ендпоінти для керування категоріями

3.2.5 Створення можливостей

У системі важливим аспектом ефективного розподілу задач є врахування індивідуальних можливостей кожного робота. Можливості (capabilities) – це перелік дій, функцій або технічних характеристик, які конкретний робот може виконувати. Наприклад, здатність переміщати вантаж, працювати на відкритому повітрі, обробляти делікатні об’єкти тощо.

Основні функції модуля:

- створення та редагування можливостей – кожна можливість має назву, опис та може бути використана як обов’язкова чи додаткова умова при розподілі задач;
- прив’язка можливостей до роботів – користувач може вказати, які саме можливості має кожен робот. Один робот може мати довільну кількість можливостей;
- Фільтрація задач за можливостями – під час пошуку релевантного робота для задачі, система виконує перевірку, чи відповідає набір можливостей вимогам задачі. Це дозволяє уникнути неправильного призначення задач.

Усі алгоритми, що реалізовані в системі, перед призначенням задачі виконують фільтрацію по можливостях. Якщо робот не має обов’язкової можливості, він одразу виключається з переліку кандидатів. Якщо ж мова про додаткові (необов’язкові) можливості, то вони можуть впливати на рейтинг або оцінку відповідності.

На сервері можливості реалізовані як дві окремі таблиці в базі даних (capabilities та capabilities groups), які поєднані з роботами зв’язком багато-до-багатьох. Задачі також мають перелік як і обов’язкових і додаткових

(опціональних) capabilities, що перевіряється при обробці запиту на призначення задачі.

API надає стандартний набір операцій (рисунок 3.11) для створення, редагування, перегляду та видалення можливостей, а також для управління прив'язкою до роботів (рисунок 3.12). Цей підхід забезпечує масштабованість та розширюваність: у майбутньому можна додати нові можливості або змінити логіку обробки без суттєвих змін у структурі системи.

Capabilities			^
POST	/Capabilities		🔒 ▼
GET	/Capabilities		🔒 ▼
DELETE	/Capabilities/{id}		🔒 ▼
GET	/Capabilities/{id}		🔒 ▼
PUT	/Capabilities/{id}		🔒 ▼
GET	/Capabilities/groups		🔒 ▼

Рисунок 3.11 – Операції керування можливостями

Robots			^
POST	/Robots		🔒 ▼
GET	/Robots		🔒 ▼
DELETE	/Robots/{id}		🔒 ▼
GET	/Robots/{id}		🔒 ▼
PUT	/Robots/{id}		🔒 ▼
PUT	/Robots/{id}/status		🔒 ▼
GET	/Robots/{id}/next-task		🔒 ▼
POST	/Robots/import		🔒 ▼

Рисунок 3.12 – Операції прив'язки можливостей до роботів

3.2.6 Створення роботів

Ключовим елементом системи є роботи – виконавці задач. Для конкретної роботи алгоритмів розподілу завдань, кожен робот має бути зареєстрований у системі з детальними характеристиками та параметрами, що впливають на призначення задач.

При створенні нового робота вказуються такі дані:

- назва робота – унікальне ім'я, що дозволяє легко ідентифікувати його в системі;
- категорії – типи або групи, до яких належить робот (наприклад, дрон, наземна платформа, маніпулятор);
- можливості (capabilities) – перелік дій, які здатен виконувати робот (обираються із попередньо створеного списку);
- інші параметри – наприклад, координати останнього місця роботи, час завершення попереднього завдання, тощо.

З боку API передбачені ендпоінти (рисунок 3.13) для:

- створення, оновлення, видалення робота;
- отримання інформації про поточний стан робота.

Robots			^
POST	/Robots		🔒 ▼
GET	/Robots		🔒 ▼
DELETE	/Robots/{id}		🔒 ▼
GET	/Robots/{id}		🔒 ▼
PUT	/Robots/{id}		🔒 ▼
PUT	/Robots/{id}/status		🔒 ▼
GET	/Robots/{id}/next-task		🔒 ▼
POST	/Robots/import		🔒 ▼

Рисунок 3.13 – Операції керування роботами

3.2.7 Створення задач

Завдання є центральним елементом у системі розподілу. Кожне завдання описує дію або набір дій, які мають бути виконані певним роботом, та містять усю необхідну для цього інформацію.

При створенні задачі вказуються такі поля (рисунок 3.14):

- назва задачі – короткий опис або унікальний ідентифікатор завдання;

- категорія – тип задачі, що використовується для попереднього фільтрування відповідних робіт;
- обов’язкові можливості – перелік capabilities, які обов’язково має підтримувати робот для виконання задачі;
- необов’язкові можливості – бажані, але не критичні можливості;
- пріоритет – числовий або рівневий показник важливості задачі, який враховується під час конкуренції між задачами;
- оцінка тривалості (ETA) – очікуваний час виконання, який використовується при балансуванні навантаження;
- складність – умовна метрика, яка дозволяє алгоритмам враховувати не лише час, а й ресурсну вартість виконання.

Також передбачено журнал задач, у якому відображаються всі створені завдання з фільтрами та пошуком.

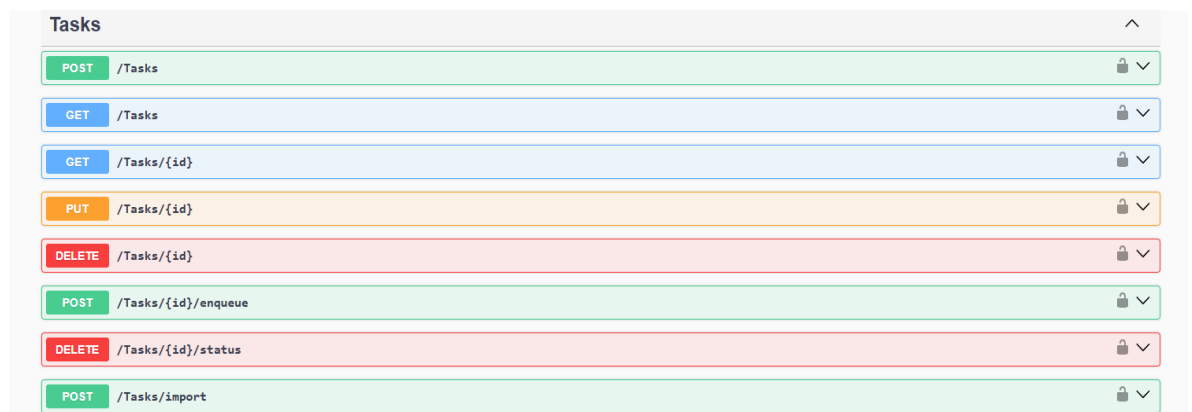


Рисунок 3.14 – Операції керування задачами

3.2.8 Розподілення задач (алгоритмічна частина)

Модуль розподілення задач є ядром та ключовим елементом інтелектуальної системи, адже саме завдяки ньому приймається зважене рішення про те, який робот повинен виконати кожну окрему задачу. Його безпосередня робота базується на заздалегідь визначених фільтрах для окремих задач, конфігураціях робіт, бізнес-правилах та алгоритмах оптимального призначення.

Етапи розподілення задачі:

а) фільтрація за категорією;

1) кожна задача належить до певної категорії робіт;

2) на цьому етапі відсіюються всі, хто не підходить за категорією;

б) перевірка обов'язкових можливостей;

1) система перевіряє, чи робот має всі capabilities, зазначені в задачі як обов'язкові;

2) якщо хоча б одна можливість відсутня – робот виключається з розгляду;

в) оцінка відповідності необов'язковим можливостям;

1) цей параметр не є критичним, але надає перевагу роботам, які мають більше таких можливостей;

г) оцінка навантаження;

1) використовується залишкове навантаження (розраховується як ЕТА поточних задач);

2) роботи з меншим навантаженням отримують більший шанс на призначення задачі;

д) алгоритмічна логіка;

1) у системі реалізовано набори алгоритмів: класичні, евристичні, математичні;

2) користувач може обрати алгоритм для задачі або залишити поле порожнім, що значить, що система використає системний алгоритм, зазначений в конфігурації компанії.

Архітектура дозволяє легко додавати нові алгоритми. Кожен алгоритм реалізовано як окремий сервіс, що реалізує спільний інтерфейс. Визначено певний набір методів, які можуть бути використані в роботі конкретних алгоритмів розподілу.

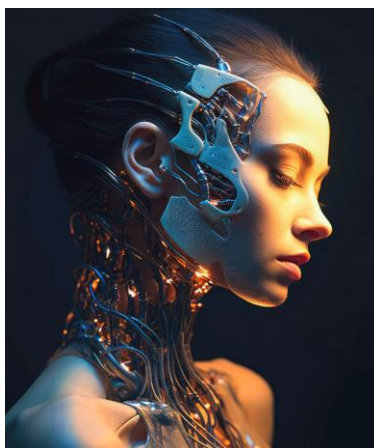
Це дозволяє в майбутньому масштабувати підхід до більш складних та комплексних алгоритмів. Детальна блок-схема розподілу зображена на кресленіку ІК11.270БАК.006 Д2.

3.3 Клієнтська частина

3.3.1 Реєстрація та аутентифікація

У клієнтській частині реалізовано окремий незалежний модуль для реєстрації нових компаній, запрошення користувачів та авторизації існуючих. Процес взаємодії з серверною частиною виконується через серверний API, де всі запити до ендпоінтів здійснюється з використанням HTTPS для безпечної та надійної передачі даних.

При відкритті застосунку користувач потрапляє на сторінку входу (рисунок 3.15), де має змогу ввести логін (email) та пароль. Після успішної аутентифікації система отримує JWT-токен, який зберігається у localStorage браузера й використовується для авторизованих запитів до захищених ресурсів.



Log in
Not a member yet? [Register your company](#)

Email

Password

[Forgot your password?](#)

Рисунок 3.15 – Сторінка авторизації

Інтерфейс розроблено з урахуванням простоти та зручності користування: є валідація форм, повідомлення про помилки введення, а також обробка неправильних даних, що забезпечує інтуїтивний механізм взаємодії з системою.

На окремій сторінці (рисунок 3.16) реалізована також реєстрація нової компанії. Вона вимагає вказання пошти та назви. При успішній реєстрації, користувач отримує лист на вказану пошту з привітальним повідомлення та посиланням на сторінку, де він може завершити реєстрацію, вказавши новий пароль для входу (рисунок 3.17).

					ІК11.270БАК.006 ПЗ	Арк.
						44
Зм.	Лист	№ докум.	Підпис	Дата		



Create new company

Enter details about your new company

Create company

Рисунок 3.16 – Реєстрація нової компанії

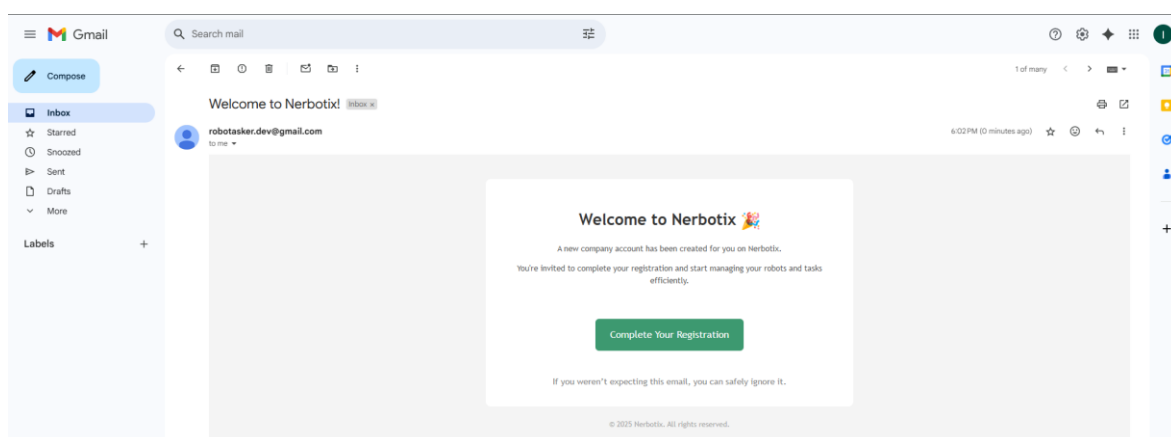


Рисунок 3.17 – Привітальний лист після створення компанії

3.3.2 Керування користувачами, ролями та правами доступу

Клієнтська частина системи передбачає зручний інтерфейс для адміністраторів, які мають змогу переглядати список зареєстрованих користувачів, запрошувати нових, призначати їм ролі, редагувати та керувати правами доступу (рисунок 3.18).

На головній сторінці модуля керування користувачами відображається таблиця з повною інформацією: імена, email-адресам, ролі та кнопки керування користувачами. Кожен користувач може мати одну або кілька ролей, які визначають його можливості та доступи в системі. Адміністратор може змінювати дані користувача, та за потреби повністю видалити користувача із системи. Один користувач може належати лише до однієї компанії.

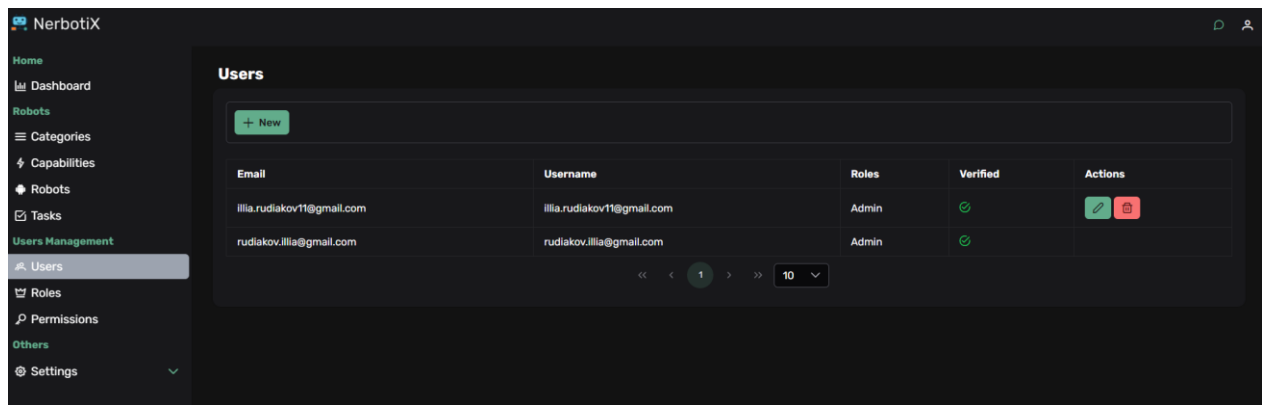


Рисунок 3.18 – Сторінка керування користувачами

При натисканні кнопки редагування відкривається форма, де можна змінити роль та ім'я користувача (рисунок 3.19). Список можливих ролей формується динамічно з серверної частини, яка отримує список з бази даних.

Основні ролі в системі:

- адміністратор – має повний доступ до всіх функцій, включно з керуванням користувачами, конфігурацією алгоритмів, конфігурацією системи та іншими критичними налаштуваннями;
- користувач (обмежений доступ) - може переглядати статуси робіт і задач, але не має можливості вносити зміни.

Кожна дія в інтерфейсі перевіряється на основі ролі користувача. Наприклад, кнопки редагування доступні лише адміністраторам. Це забезпечується через механізм захищених маршрутів на рівні клієнта. Це гарантує, що жоден користувач не зможе отримати доступ до ресурсу, який не передбачений наданою йому роллю. Завдяки цьому гарантується безпека та цілісність даних.

Рисунок 3.19 – Форма керування користувачем

3.3.3 Конфігурація системи

Клієнтська частина містить окремий розділ конфігурацій, призначений для адміністраторів. Цей розділ дозволяє налаштовувати системні параметри, необхідні для взаємодії з роботами, а також конфігурувати параметри алгоритмів розподілу завдань. Розділ складається з двох основних підпунктів: керування API ключами та налаштування алгоритмів.

API ключі використовуються для автентифікації роботів, які взаємодіють із серверною частиною системи. У клієнтському інтерфейсі реалізовано функціонал генерації та видалення API ключів (рисунок 3.20). Інтерфейс дозволяє:

- створити новий ключ з описом (наприклад, «Робот 4»);
- переглянути активні ключі;
- видалити ключ.

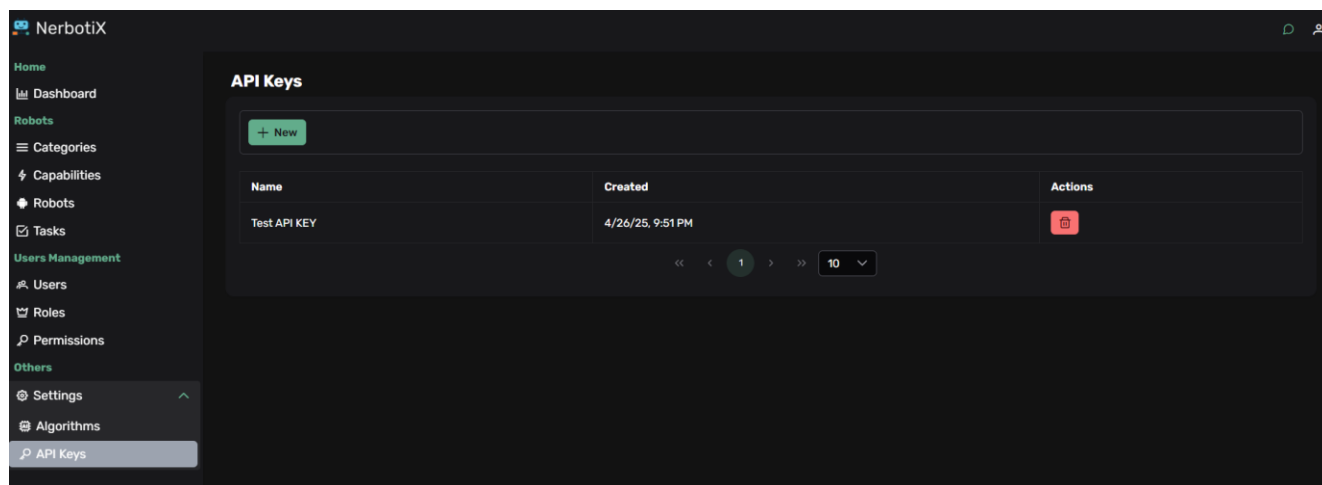


Рисунок 3.20 – Керування API ключами

Для алгоритмів розподілу завдань (наприклад, Ant Colony) передбачено окремий інтерфейс налаштування параметрів (рисунок 3.21). Залежно від обраного алгоритму, користувач бачить перелік доступних параметрів (наприклад, IterationCount, EvaporationRate, Alpha, Beta тощо). Інтерфейс дозволяє:

- вибрати алгоритм;
- задати значення параметрів вручну;

– зберегти конфігурацію, яка буде застосована для наступного сеансу розподілення задач.

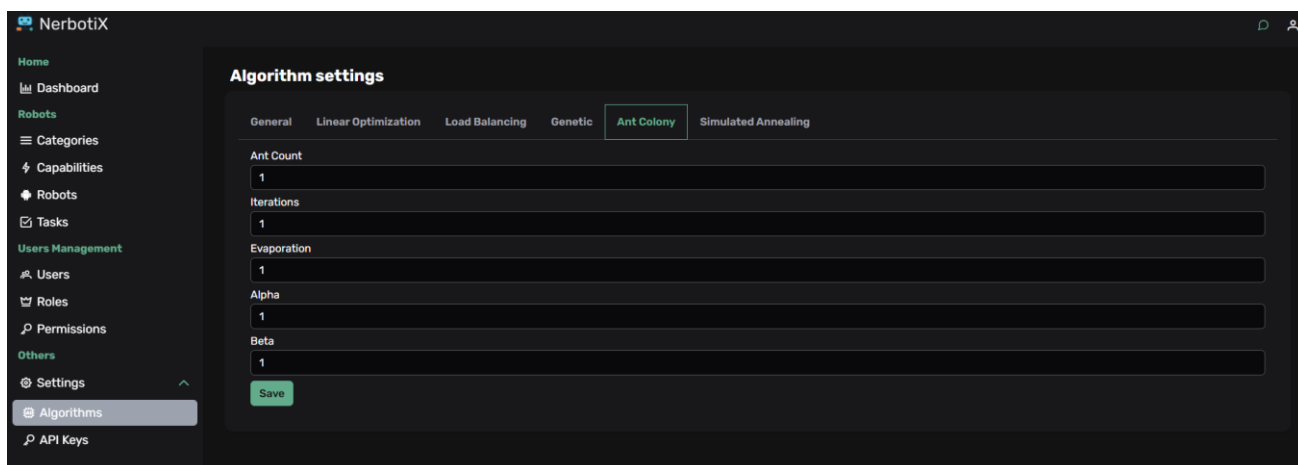


Рисунок 3.21 – Конфігурація параметрів алгоритмів

Завдяки цьому модулю користувачі можуть динамічно експериментувати з параметрами алгоритмів, не вдаючись до змін у коді, що значно полегшує процес тестування та оптимізації.

3.3.4 Створення категорій

Для зручної класифікації роботів у системі передбачено можливість створення категорій. Категорії дозволяють групувати роботів за типами, функціональними можливостями або призначенням, що суттєво полегшує управління та подальше призначення завдань.

У клієнтській частині реалізовано інтерфейс, де адміністратор може:

- переглянути наявні категорії у вигляді таблиці (рисунок 3.22);
- створити нову категорію, вказавши її назву, опис, властивості (рисунок 3.23);
- редагувати або видалити існуючі категорії.

Під час створення задач або роботів користувач може обрати одну або декілька категорій, що дозволяє фільтрувати або обмежувати доступ до завдань певного типу.

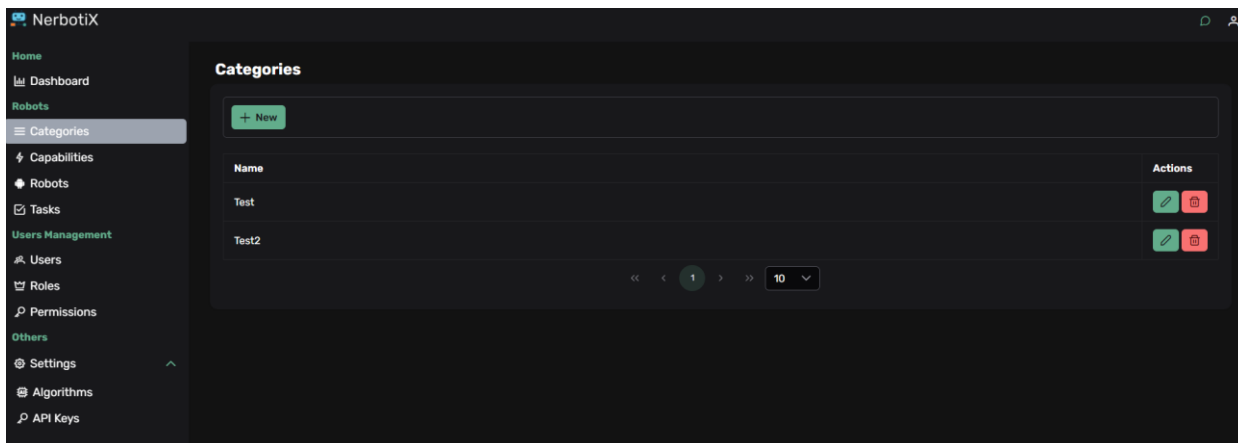


Рисунок 3.22 – Сторінка перегляду категорій

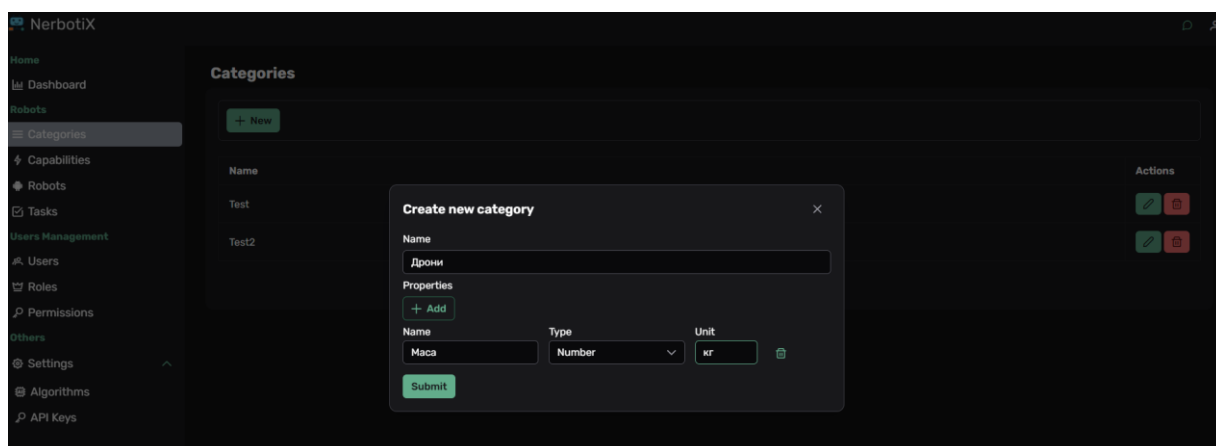


Рисунок 3.23 – Вікно створення категорій

3.3.5 Створення можливостей

У системі можливості (capabilities) відіграють доволі важливу роль, та визначають, які саме дії або функції здатен виконувати конкретний виконавець (робот). Це дозволяє більш точно та ефективно призначати задачі, адже деякі завдання можуть виконуватись лише з роботами з відповідними можливостями (рисунок 3.24), наприклад деякі роботи мають кращу потужність та можуть виконувати важкі завдання.

Інтерфейс клієнтської частини дозволяє:

- переглядати перелік уже наявних можливостей;
- додавати нові можливості з описом;
- редагувати або видаляти існуючі.

Можливості можуть бути довільно налаштовані адміністратором. Під час створення або редагування робота можна вказати, якими можливостями він володіє, шляхом мультивибору з доступного списку (рисунок 3.25).

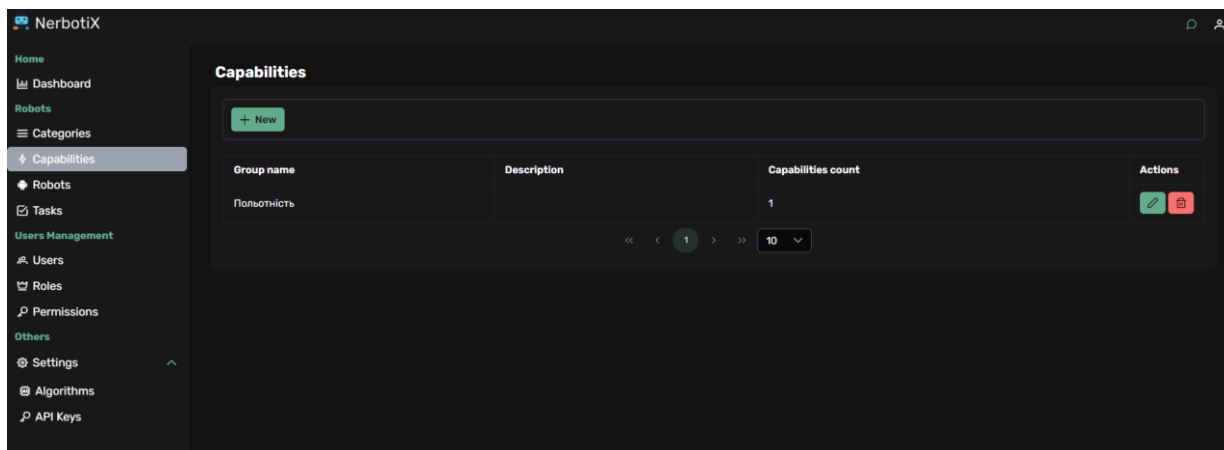


Рисунок 3.24 – Керування можливостями

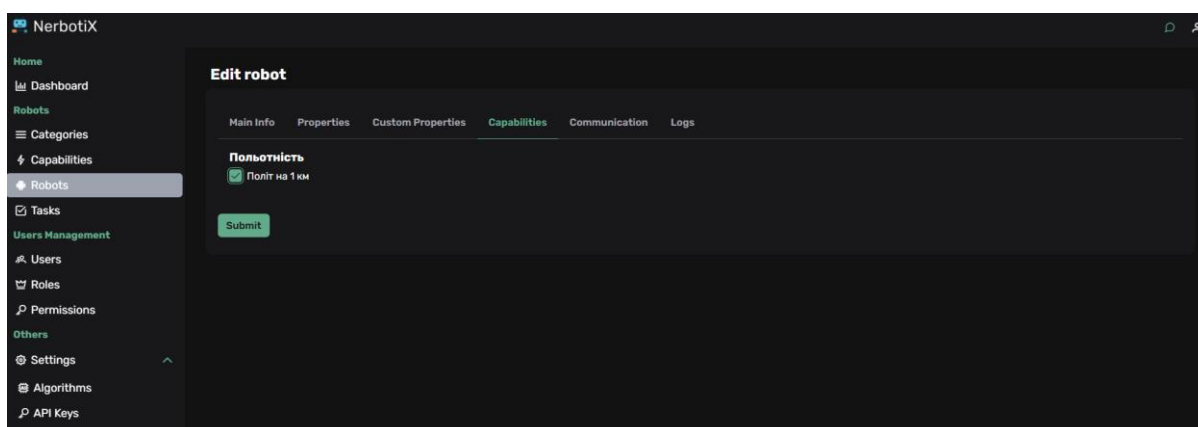


Рисунок 3.25 – Керування можливостями робота

3.3.6 Створення роботів

Створення роботів є одним із ключових етапів налаштування системи. Кожен робот в системі представляє окремий виконавчий пристрій (агент), що має власні характеристики, функції, можливості та параметри (рисунок 3.26). У процесі створення робота адміністратор обирає для нього відповідні можливості (capabilities), що відповідають його функціональним навичкам. Також задаються базові параметри, такі як назва, опис, конфігурації та інші.

Форма створення робота включає в себе наступні поля (рисунок 3.27):

- назва – унікальне ім'я робота для ідентифікації;
- категорії – вибирається одна або декілька створених категорій, що описує тип пристрою;
- властивості – роботу назначаються значення властивостей, які прив'язані до категорій;
- тип комунікації та параметри комунікації (рисунок 3.28).

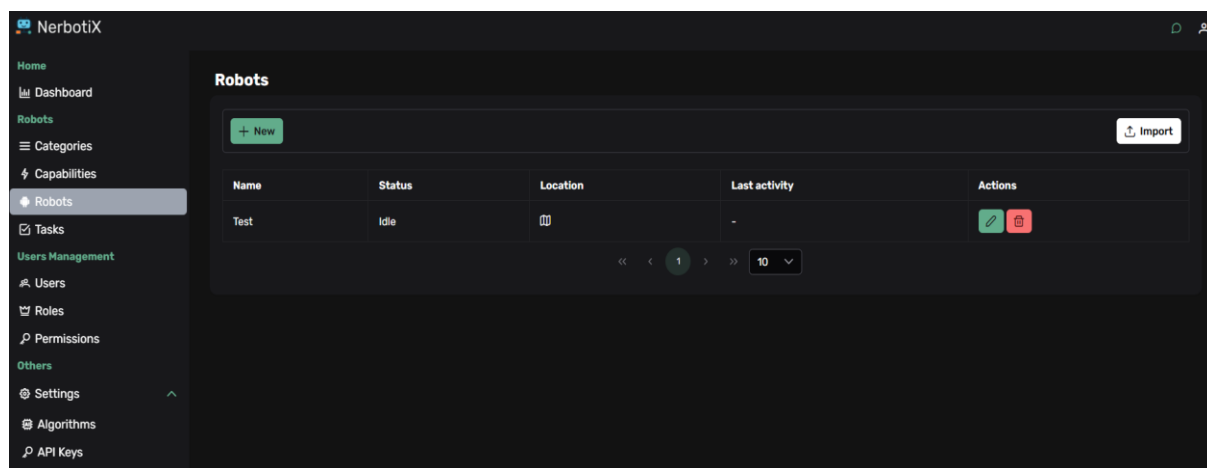


Рисунок 3.26 – Сторінка керування роботами

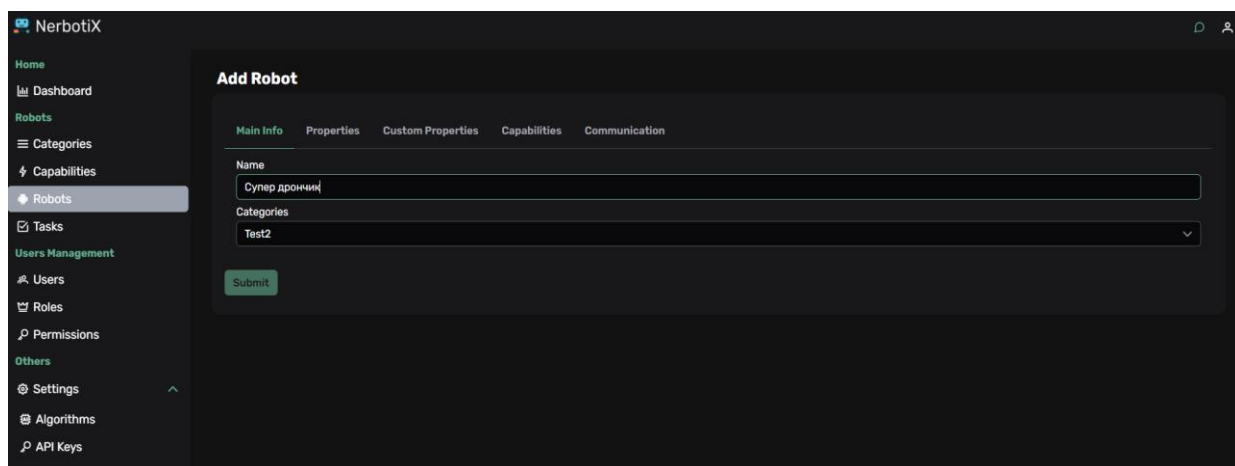


Рисунок 3.27 – Форма створення робота

Також передбачено можливість імпортування роботів з файлу JSON. Користувач може завантажити файл з списком роботів і після цього вони будуть додані до системи.

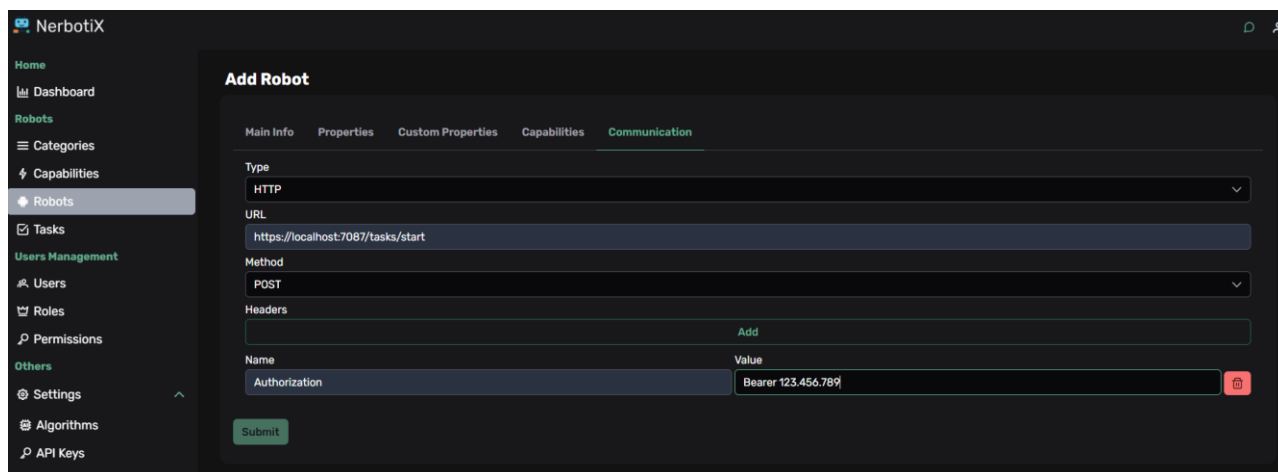


Рисунок 3.28 – Керування взаємодією з роботом

Після створення робот з’являється у загальному переліку, де можна легко та швидко змінювати його параметри, переглядати список активних задач тощо. Завдяки цьому користувач може швидко переглянути список виконавців, та їх актуальний стан.

3.3.7 Створення задач

Створення задач є центральною частиною взаємодії користувача з системою. Задачі формують робоче навантаження для роботів і містять основну інформацію, необхідну для планування та виконання.

Інтерфейс створення задачі включає такі поля (рисунок 3.29):

- назва задачі – короткий опис який ідентифікує завдання;
- опис – додаткове текстове поле для розширеного опису суті задачі;
- категорія роботів – визначає, які роботи можуть бути кандидатами для виконання задачі;
- необхідні можливості (capabilities) – система автоматично фільтрує роботів, які відповідають заданим вимогам;
- пріоритет – значення від 1 (найвищий) до 10 (найнижчий), впливає на черговість розподілу;
- орієнтовний час виконання (ETA) – у форматі: «ГГГ:ММ».

Рисунок 3.29 – Форма створення задачі

Після заповнення та підтвердження задача потрапляє у загальну чергу задач, з якої надалі буде призначена одному з доступних роботів на основі заданого алгоритму розподілу та відповідних заданих параметрів та конфігурацій виконавців. У списку задач користувач бачить поточний статус, призначеного робота, тощо (рисунок 3.30).

Name	Description	Status	Completed	Estimate	Priority ↑↓	Complexity	Robot	Actions
AAA		InProgress	-	00:00:05	1	1	-	

Рисунок 3.30 – Список задач

Аналогічно до роботів, користувач може завантажити (імпортувати) список задач із файлу JSON, що значно полегшує створення великої кількості задач.

3.3.8 Розподіл задач

Хоча основна логіка розподілу задач виконується на серверній стороні, клієнтський інтерфейс забезпечує користувача всіма необхідними інструментами для моніторингу та контролю процесу.

Після створення задачі, система автоматично ініціює процес призначення відповідного робота з урахуванням усіх заданих параметрів: категорій, необхідних можливостей, пріоритету, ЕТА та поточного стану роботів. У випадку, якщо на момент створення задачі немає доступного робота, який відповідає критеріям, система залишає задачу у статусі «очікую призначення» (рисунок 3.29).

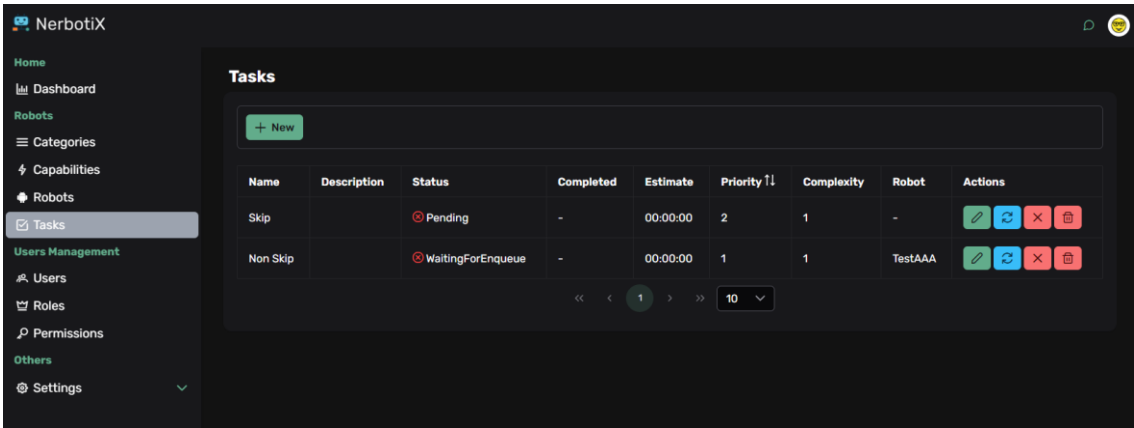


Рисунок 3.31 – Статус задачі, для якої не знайдено виконавця

У такій ситуації користувач має можливість вручну повторно ініціювати процес розподілу, скориставшись відповідною кнопкою у клітинці Actions. Це дозволяє, наприклад, після зміни параметрів задачі або появи нових доступних роботів, повторно запустити спробу призначення без необхідності створювати нову задачу (рисунок 3.30).

Name	Description	Status	Completed	Estimate	Priority ↑↓	Complexity	Robot	Actions
Skip		⊗ Pending	-	00:00:00	2	1	-	   

Рисунок 3.32 – Можливість повторного розподілу задачі

3.4 Алгоритми

У системі розподілу задач між роботизованими агентами ключовим елементом є вибір алгоритму, який визначає, яким чином і на основі яких критеріїв задачі призначаються виконавця. Оскільки характеристик задач, доступних роботів та вимоги до ефективності можуть змінюватися, система передбачає модульну архітектуру, яка підтримує використання різних типів алгоритмів.

Усі алгоритми умовно розділено на такі категорії:

- класичні алгоритми, які базуються на простих правилах та метриках, часто використовуються в системах з обмеженими вимогами до точності або в реальному часі;
- евристичні алгоритми, які застосовують підходи з області штучного інтелекту або оптимізаційного пошуку;
- математичні алгоритми, які використовують формалізовані методи, засновані на математичних моделях і методах оптимізації.

Усі алгоритми мають єдиний інтерфейс реалізації, що дозволяє легко підключати нові методи без зміни логіки решти системи.

3.4.1 Latest Task Finished

Даний алгоритм реалізує просту, але ефективну логіку призначення задач: обирається той робот, який найдавніше завершив попередню задачу або перебуває в режимі очікування найдовше. Такий підхід дозволяє зменшити час простою кожного робота та забезпечити більш рівномірний розподіл навантаження у динамічних середовищах.

Принцип роботи:

- для кожного робота, що доступний до виконання задач, перевіряється час завершення його останньої задачі (або час виходу в режим очікування);
- алгоритм сортує список доступних роботів за цим часом у порядку зростання;

– обирається перший у списку – той, хто найдовше чекає.

Переваги:

- простота реалізації;
- добре працює в системах з відносно рівномірним потоком задач;
- зменшує ймовірність «забутих» робіт, що надто довго залишаються без задач.

Недоліки:

- не враховує потужність, швидкість чи інші характеристики робота;
- не враховує складність задачі – задачі можуть потрапити до технічно слабших агентів.

3.4.2 Least Connections

Даний алгоритм є простим та ефективним підходом до розподілу завдань у динамічному середовищі. Основна ідея цього методу полягає у виборі такого робота, у якого найменше поточних активних завдань у черзі. Такий підхід дозволяє збалансувати навантаження, а саме розмір пулу задач між усіма доступними роботами та уникати ситуацій, коли одні роботи перевантажені великою кількістю задач, а інші простоюють.

Принцип роботи:

- для кожного з робіт визначається кількість задач у його черзі;
- вибирається робот з найменшим значенням цього показника.

Переваги:

- простота реалізації;
- швидка робота навіть при великій кількості робіт;
- зменшує ймовірність простою робіт, рівномірно розподіляючи навантаження;
- вирівнює пули робіт.

Недоліки:

- не враховує тривалість задач чи швидкість виконання кожним роботом;

– може призначити задачу зайнятому роботу, якщо той має коротку чергу, але повільно виконує завдання.

3.4.3 Load Balancing

Даний алгоритм є більш гнучким і намагається враховувати не лише кількість завдань у черзі, а й більш комплексне поняття «навантаження» робота. Це може включати: кількість задач, їхню тривалість, статус робота, його індивідуальні характеристики. У поточній реалізації навантаження обчислюється як сумарна тривалість всіх задач у черзі. Це дозволяє надавати перевагу тим роботам, які хочуть і мають декілька задач, але з коротким часом виконання.

Принцип роботи:

- для кожного робота обчислюється метрика «навантаження», яка може базуватися на сумарній оцінці тривалості задач у черзі (ETA);
- алгоритм вибирає робота з найменшим сумарним навантаженням.

Переваги:

- більш точна оцінка стану кожного робота порівняно з простими лічильниками;
- зменшує ймовірність затримок, пов'язаних із призначенням задач повільним або перевантаженим роботам.

Недоліки:

- потребує додаткової інформації про задачі (наприклад, оцінки тривалості, пріоритетності, складності тощо);
- трохи складніший у реалізації та може мати більшу обчислювальну складність в порівнянні з попередніми алгоритмами.

3.4.4 Random

Даний алгоритм є одним із найпростіших методів розподілу задач між роботами. Його основна ідея полягає у випадковому виборі одного з доступних

					ІК11.270БАК.006 ПЗ	Арк.
						57
Зм.	Лист	№ докум.	Підпис	Дата		

роботів, який задовольняє вимоги задачі. Цей підхід не враховує жодних характеристик задач або роботів (навантаження, пріоритет, тривалість, складність тощо), що робить його максимально простим, але й найменш оптимальним для складних сценаріїв.

Принцип роботи:

- всі доступні (відфільтровані за параметрами) роботи збираються в список;
- з нього обирається один випадковий робот.

Переваги:

- простота реалізації;
- може бути корисним у тестуванні або випадках, коли всі роботи мають приблизно однакове навантаження та характеристики.

Недоліки:

- ігнорує навантаження, стан, попередню активність чи продуктивність роботів;
- може призводити до дисбалансу у випадку великої кількості задач або складного середовища.

3.4.5 Round Robin

Даний алгоритм є одним із класичних методів, який застосовується для циклічного розподілу завдань серед роботів. У цьому випадку завдання надаються роботам по черзі, без урахування їхнього навантаження або попередньої діяльності, тобто кожен робот отримує задачу по черзі від першого до останнього, а потім цикл повторюється. Лічильник, на основі якого працює даний алгоритм, зберігається в налаштуваннях окремо для кожної компанії.

Принцип роботи:

- всі доступні роботи збираються в список;
- завдання передається кожному роботу по черзі у визначеному порядку;
- коли досягається останній робот у списку, наступне завдання отримує перший робот знову, і процес повторюється.

Цей алгоритм простий, передбачуваний і корисний, коли важливо забезпечити рівномірне розподілення задач, однак він не враховує продуктивність чи навантаження кожного робота.

Переваги:

- простота реалізації та передбачуваність;
- чудово працює в ситуаціях, коли роботи рівні за своїми характеристиками і навантаженням;
- забезпечує рівномірний розподіл задач між роботами.

Недоліки:

- ігнорує реальний стан роботів. Наприклад, якщо один робот зайнятий складною задачею, а інший – неактивний, вони будуть отримувати нові задачі по черзі, що може призвести до неефективності;
- не враховує пріоритети задач або інші параметри, що може призвести до неефективності.

3.4.6 Ant Colony

Даний алгоритм базується на принципах пошуку та оптимізації, натхнених поведінкою мурах у природі. Вони здатні знаходити найкоротші шляхи до джерела їжі завдяки використанню феромонів, які залишають на своєму шляху та які інші мурахи можуть виявляти і слідувати за ними.

У випадку розподілу завдань, колонія мурах використовує схожий підхід для вибору робота. Алгоритм намагається знайти оптимального робота для виконання завдань, використовуючи інформацію про поточний стан кожного робота, який оцінюється через фітнес-функцію, а також через кількість феромонів, які вказують на «найкращих» роботів.

Принцип роботи:

- ініціалізація: визначаються параметри колонії мурах, такі як кількість мурах, ітерацій, випаровування феромонів, а також коефіцієнти альфа і бета, які регулюють вплив феромонів і евристичних оцінок;

– пошук робота: мурахи обирають роботів на основі ймовірностей, які визначаються через два фактори: кількість феромонів та його «фітнес», що розраховується за допомогою оцінки властивостей робота;

– оновлення феромонів: Після кожної ітерації мурахи оновлюють феромони на обраних роботах, збільшуючи їх за рахунок вибору кращих роботів для виконання завдань.

Переваги:

– алгоритм ефективно працює в динамічних середовищах, де необхідно враховувати як характеристики роботів, так і їх поточний стан;

– враховує як поточне навантаження, так і «довгострокову» ефективність роботів завдяки феромонам;

– є адаптивним до зміни середовища і може покращувати свої результати.

Недоліки:

– висока складність обчислень, особливо з великою кількістю роботів та ітерацій;

– потрібно налаштовувати параметри для досягнення оптимальних результатів;

– алгоритм може бути повільним у порівнянні з іншими підходами, якщо потрібна швидка реакція.

Даний алгоритм в багатьох випадках може обирати одного і того же виконавця, що може викликати перевантаження окремих агентів.

3.4.8 Genetic Task

Генетичний алгоритм – це метод оптимізації, що імітує еволюційні процеси, такі як селекція, схрещування та мутація, щоб знаходити рішення складних задач.

В контексті розподілу завдань роботам, генетичний алгоритм використовує популяцію роботів для вирішення задачі, оптимізуючи їх характеристики через ітерації (покоління). Це дозволяє знайти робота, який найкраще відповідає вимогам конкретного завдання.

Принцип роботи:

- ініціалізація популяції: вибір початкової популяції роботів із доступних для виконання завдань;
- оцінка фітнесу: оцінка кожного робота на основі його характеристик (к-сть завдань, завантаження, складність поточних завдань тощо);
- селекція: вибір «елітних» роботів (найкращих за фітнесом), які будуть брати участь у створенні наступного покоління;
- схрещування: вибір пар роботів для схрещування, з метою створення нових роботів (нащадків), що успадковують характеристики батьків;
- мутація: випадкове змінювання деяких роботів для збагачення популяції і підтримки генетичного різноманіття;
- повторення: процес повторюється для кількох поколінь, поки не буде знайдений робот, який найкраще підходить для виконання завдання.

Переваги:

- може ефективно вирішувати задачі з великою кількістю роботів і завдань;
- підходить для пошуку складних рішень, де немає чітко визначеного оптимального шляху.

Недоліки:

- вимагає налаштування параметрів (наприклад розмір популяції, ймовірність мутації, тощо);
- зазвичай має високу складність обчислень, що може впливати на продуктивність, особливо з великою кількістю поколінь;
- може не завжди гарантувати глобальний оптимум.

3.4.9 Fuzzy Logic

Алгоритм для розподілу завдань роботам використовує принципи нечіткої логіки для оцінки кожного робота на основі його властивостей та інші фактори, що можуть впливати на ефективність виконання завдання. Він схожий на багатокритеріальну задачу, яка враховує набір параметрів та їх важливість.

Нечітка логіка дозволяє уникнути точних порівнянь і врахувати варіативність (гетерогенність) характеристик роботів, що робить алгоритм більш гнучким у різних ситуаціях.

Принцип роботи:

– оцінка робота: кожен робот оцінюється за допомогою нечіткої логіки, що визначає ступінь належності його властивостей до заданого діапазону (низький, середній, високий);

– функція належності: для кожної властивості робота розраховується коефіцієнт належності, що визначає, наскільки ця характеристика підходить для виконання завдання;

– розподіл завдання: роботи з найвищими оцінками (найбільшою сумою коефіцієнтів належності) обираються для виконання задачі.

Переваги:

– гнучкість: алгоритм може працювати з нечіткими та варіативними характеристиками роботів, що дозволяє ефективно вирішувати завдання навіть у складних умовах;

– швидкість прийняття рішень: порівняно з іншими алгоритмами, що потребують складних розрахунків, цей підхід дозволяє швидше отримувати результат;

– легкість налаштування: параметри нечіткої логіки можна швидко адаптувати до конкретних вимог завдання та властивостей роботів.

Недоліки:

– нечіткість у результатах: якщо властивості роботів сильно відрізняються за значеннями, алгоритм може не дати оптимального вибору, оскільки враховуються лише «близькі» до середнього значення властивості;

– потребує коректних вхідних даних: для досягнення високої ефективності необхідно правильно налаштувати значення факторів та коефіцієнтів належності, що може вимагати додаткової конфігурації;

– залежність від інтерпретації властивостей: оцінка роботів може бути менш точною, якщо неправильно обрані фактори для нечіткої логіки.

					ІК11.270БАК.006 ПЗ	Арк.
						62
Зм.	Лист	№ докум.	Підпис	Дата		

3.4.10 Simulated Annealing

Алгоритм імітації відпалу – це евристичний метод пошуку глобального оптимуму, що базується на аналогії з процесом охолодження металу. У контексті розподілу завдань, алгоритм застосовується для знаходження найкращого робота для конкретного завдання шляхом поступового вивчення сусідніх варіантів із можливістю тимчасового прийняття гірших рішень для уникнення локального максимуму.

Принцип роботи:

- ініціалізація: випадково обирається початковий робот як поточне рішення. Задаються початкова температура, швидкість охолодження, кількість ітерацій та мінімальна температура;

- оцінка рішень (фітнес-функція): для кожного робота розраховується «придатність» на основі його характеристик і навантаження (враховується кількість активних завдань і загальна тривалість черги);

- пошук сусідів: випадково обирається інший робот;

- правило прийняття: якщо новий робот має кращу оцінку – він приймається. Якщо гіршу – може бути прийнятий із певною ймовірністю, що знижується зі зменшенням температури;

- охолодження: температура знижується після кожного циклу, поступово зменшуючи ймовірність прийняття гірших рішень;

- завершення: алгоритм завершується, коли температура досягає мінімального значення. Повертається робот із найкращою оцінкою.

Переваги:

- пошук глобального оптимуму: завдяки можливості приймати гірші рішення, алгоритм може уникнути «застрягання» в локальних максимумах;

- гнучкість: може працювати з будь-якою функцією оцінки, що враховує специфіку роботи роботів і задач;

- простота реалізації: алгоритм має просту структуру та не потребує складних математичних моделей.

Недоліки:

- нестабільність результату: через випадковість на початку, результат може відрізнятись при кожному запуску;
- налаштування параметрів: вибір початкової температури, швидкості охолодження та кількості ітерацій суттєво впливає на якість результату;
- швидкість: при великій кількості роботів або занадто високій кількості ітерацій, час виконання може зростати.

3.4.11 Linear Optimization

Алгоритм лінійної оптимізації – прямолінійний підхід до вибору найкращого робота на основі зваженої суми його числових характеристик. Кожна властивість робота (наприклад, швидкість, точність, вантажопідйомність, потужність, рівень запасу батареї, тощо) має власне числове значення як коефіцієнт значущості (вагу), яке враховується під час обчислення загального балу робота.

Принцип роботи:

- обчислення балу: для кожного робота розраховується загальний бал як сума добутків значень його числових характеристик на відповідні ваги;
- врахування цілі оптимізації: в залежності від налаштувань категорії задач система або максимізує результат (вибирає робота з найвищим балом), або мінімізує (вибирає робота з найнижчим балом), наприклад коли потрібно розвантажити переважених роботів;
- вибір робота: усі роботи сортуються за власною оцінкою, а далі алгоритм обирає першого робота з найвищою загальною оцінкою.

Переваги:

- швидкість: алгоритм працює дуже швидко, адже не потребує складних ітерацій чи випадкових виборів;
- простота налаштування: достатньо всього лиш правильно задати конфігурацію для певної категорії: лише задати ваги властивостей і вибрати напрям оптимізації.

Недоліки:

- лінійність: алгоритм не враховує взаємозв'язки між характеристиками або змінні зовнішні умови;
- не враховує завантаження: якщо не занести це як властивість, то алгоритм може обрати вже перевантаженого робота;
- не гнучкий: не підходить для задач, де потрібна адаптація або врахування складної логіки.

3.4.12 Порівняльна характеристика алгоритмів

У даній системі було реалізовано та досліджено кілька категорій алгоритмів для розподілу задач між роботами, що охоплюють різні підходи. Нижче та в таблиці 3.1 подана порівняльна характеристика усіх алгоритмів, створена на основі різних факторів, особливостях:

- прості евристики: RoundRobin, Random – орієнтовані на простоту;
- нечітка логіка: дозволяє враховувати розмиті критерії та нечіткі правила;
- стохастичні методи оптимізації: Genetic Algorithm, Simulated Annealing – імітують природні або фізичні процеси пошуку рішень;
- класичні математичні методи: Linear Optimization – використовує зважену суму числових параметрів для визначення найкращого кандидата.

Таблиця 3.1 – Порівняльна характеристика алгоритмів

Назва	Тип	Урахування стану	Адаптивність	Складність реалізації	Швидкодія	Оптимальність
Greedy	Евристика	Частково	Низька	Низька	Висока	Локальна
Round-Robin	Ротаційна схема	Ні	Середня	Дуже низька	Дуже висока	Низька
Random	Стохастичний	Ні	Середня	Дуже низька	Дуже висока	Низька
Fuzzy Logic	Нечітка логіка	Так	Висока	Середня	Середня	Середня (висока)
Genetic	Стохастична оптимізація	Так	Висока	Висока	Низька	Висока (не гарантована)

Simulated Annealing	Стохастична оптимізація	Так	Висока	Висока	Середня	Висока (не гарантована)
Linear Optimiz.	Лінійна модель	Ні	Середня	Низька	Висока	Середня (висока)

Висновки та рекомендації щодо застосування:

- Greedy – для невеликих систем, де важлива швидкість, а не точність;
- Round-Robin – для балансування навантаження без урахування специфіки;
- Fuzzy Logic – коли параметри роботів не завжди є числом;
- Genetic – для складних, багатofакторних систем;
- Simulated Annealing – для задач уникання локальних мінімумів;
- Linear Optimization – при чітко визначених числових критеріях.

Висновки до розділу 3

У даному розділі було детально представлено процес розробки інтелектуальної системи розподілу завдань для роботизованих систем. Особлива увага приділялась архітектурним рішенням, які забезпечують масштабованість, модульність, гнучкість та зручність подальшого супроводу системи. Ретельне проектування архітектури стало фундаментом для надійної реалізації всіх компонентів системи – як серверної, так і клієнтської.

На рівні архітектури серверної частини було впроваджено підхід Clean Architecture із використанням бібліотеки MediatR як посередника для взаємодії між компонентами. Такий підхід дозволив розділити бізнес-логіку від інфраструктурних елементів, що значно полегшило тестування всієї системи. Це особливо важливо для складних розподілених систем, де кожен новий функціональний модуль не повинен впливати на існуючу логіку.

Також сервер передбачає ведення журналу подій (логування), підтримку багатокористувацького середовища, адже робить систему гнучкою, бо надає можливість перегляду усіх критичних записів, що важливо під час непередбачуваних ситуацій, коли виконавці пошкоджуються чи стають

					ІК11.270БАК.006 ПЗ	Арк.
						66
Зм.	Лист	№ докум.	Підпис	Дата		

недоступними для виконання задач через поломки або інші критичні ситуації які відбуваються під час виконання завдань.

Клієнтська частина реалізована з використанням Angular – сучасного фреймворку, який забезпечує високу продуктивність та зручність створення модульного інтерфейсу. Реалізований функціонал дозволяє легко керувати користувачами, роботами, задачами, конфігураціями та параметрами алгоритмів. Розмежування прав доступу на основі ролей дозволяє ефективно організовувати роботу системи та забезпечує належний рівень безпеки.

З клієнтського боку забезпечено дружній до користувача інтерфейс, який дозволяє не лише створювати та керувати сутностями системи, а й в реальному часі спостерігати за її станом. Важливою складовою стала інтеграція модуля аналітики, який реалізує візуалізацію поточного навантаження робіт, ETA та структури черг, що дозволяє оперативно реагувати на неефективні сценарії розподілу.

Окремим важливим етапом стала реалізація різних алгоритмів розподілу. Було розроблено низку стратегій – від базових (Round Robin, Random і тп) до складних (Ant Colony, Genetic Algorithm, Simulated Annealing, Fuzzy Logic). Це дає змогу адаптувати систему до різних сценаріїв та навантажень, забезпечуючи оптимальне використання ресурсів робіт. Проведено порівняльний аналіз усіх підходів, що дозволяє обґрунтовано обирати той чи інший метод.

Крім того, усі основні процеси супроводжуються валідацією та обробкою винятків, що забезпечує стабільну роботу навіть при некоректних даних або частковій відмові компонентів. Усі критичні точки взаємодії протестовані, а реалізація у вигляді REST API забезпечує незалежність між клієнтом і сервером, що дозволяє масштабувати або замінювати будь-який компонент системи без порушення її загальної функціональності.

					ІК11.270БАК.006 ПЗ	Арк.
						67
Зм.	Лист	№ докум.	Підпис	Дата		

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методика тестування

4.1.1 Цілі тестування

Основним методом тестування є перевірка ефективності та надійності усіх реалізованих алгоритмів розподілу задач між роботами в умовах, наближених до реальних сценаріїв експлуатації в підприємствах, або інших сферах діяльності. Тестування покликане оцінити, наскільки розумно та збалансовано кожен з алгоритмів призначає задачі, мінімізуючи час простою, перевантаження окремих агентів і загальний ЕТА.

До конкретних цілей тестування належать:

- перевірка правильності розподілу задач для кожного з алгоритмів;
- оцінка ефективності алгоритмів за основними метриками: ЕТА, рівномірність навантаження, кількість задач у черзі, відсоток простоїв роботів;
- виявлення меж продуктивності (через стрес-тестування) при зростанні кількості задач та агентів;
- порівняння продуктивності алгоритмів між собою в ідентичних умовах.

У підсумку, тестування має забезпечити ґрунтовану аналітику, яка дозволить визначити доцільність використання кожного алгоритму залежно від цільового середовища, типу задач та обмежень реального застосування.

4.1.2 Вхідні дані для моделювання

Для проведення коректного та об'єктивного тестування алгоритмів розподілу задач були змодельовані набір вхідних даних, які відображають типові або найпопулярніші сценарії функціонування системи розподілу задач в реальному середовищі.

Вхідні дані умовно поділяють на три основні групи:

а) набір роботів;

1) кількість агентів: від 5 до 100, залежно від сценарію;

					ІК11.270БАК.006 ПЗ	Арк.
						68
Зм.	Лист	№ докум.	Підпис	Дата		

2) категорії робіт: кожен робот належить до певних категорій, що визначають набір властивостей;

3) властивості та можливості (capabilities): набори параметрів, що характеризують ефективність виконання задач;

б) набір задач;

1) кількість задач: варіюється від десятків до сотень залежно від тесту;

2) категорія задачі: визначає, які властивості робіт є критичними;

3) ETA: орієнтовний час виконання задачі, в секундах або хвилинах;

4) пріоритет задачі: деякі тести включають пріоритети, що впливають на порядок призначення;

в) конфігурація середовища;

1) режим моделювання: симуляція в один момент часу (batch) або в режимі надходження задач в реальному часі;

2) навантаження: три рівня – низьке, середнє, високе;

3) алгоритм та його тип.

Всі тести проводились в контрольованому середовищі з фіксованими параметрами, що дозволило порівняти поведінку та результати кожного алгоритму. Дані зберігались для подальшого аналізу й візуалізації ключових метрик.

4.1.3 Метрики оцінки ефективності

Для порівняння алгоритмів розподілу задач було визначено набір ключових метрик, які дозволяють оцінити як якість прийнятих рішень, так і загальну продуктивність системи.

Метрики умовно поділяються на індивідуальні (для окремих агентів) так і глобальні:

а) середній ETA;

1) визначає середній очікуваний час завершення усіх задач агентом;

2) чим менше значення, тим краще збалансоване навантаження між роботами;

					ІК11.270БАК.006 ПЗ	Арк.
						69
Зм.	Лист	№ докум.	Підпис	Дата		

б) стандартне відхилення ETA;

- 1) відображає нерівномірність навантаження між роботами;
- 2) мале відхилення свідчить про ефективний розподіл задач;

в) загальний час завершення всіх задач;

- 1) час від початку до завершення останньої задачі;
- 2) менше значення означає більшу загальну продуктивність системи;

г) кількість призначених задач;

- 1) визначає, скільки задач було успішно призначено алгоритмом;
- 2) важливо при перевантаженні або обмеженнях;

д) рівень завантаження роботів;

- 1) розподіл задач між агентами: коефіцієнт використання кожного робота;
- 2) використовується для оцінки балансу між роботами;

е) час виконання алгоритму;

- 1) середній час, який потрібен на обробку однієї задачі алгоритмом;
- 2) важливо для масштабованості та реального часу.

4.1.4 Процедура тестування

Процедура тестування була спрямована на об'єктивне порівняння алгоритмів розподілу задач як у стандартних, так і нестандартних сценаріях, максимально наближених до реального середовища експлуатації. Нижче наведено основні етапи тестування:

а) підготовка даних;

- 1) створення контрольного набору задач різної складності (ETA від 2 хвилин до 2 годин);
- 2) створення пулу роботів з різними можливостями, включаючи;
 - поточне навантаження;
 - кількість активних задач;
 - значення числових властивостей;

- 3) задачі автоматично призначались за допомогою кожного з алгоритмів на одному й тому самому вхідному наборі;
- б) ініціалізація тесту;
 - 1) запуск алгоритму розподілу задач для кожного тестового кейсу;
 - 2) реєстрація часу прийняття рішення;
 - 3) фіксація призначеного робота та ETA;
- в) реєстрація результатів;
 - 1) для кожного алгоритму збиралися метрики;
 - 2) результати зберігались у базі даних для подальшого аналізу;
- г) візуалізація;
 - 1) створення графіків завантаження, ETA, розподілу задач між роботами;
 - 2) побудова гістограм і часових ліній ефективності алгоритмів.

4.2 Тестування алгоритмів

У цьому розділі наведено результати тестування кожного з реалізованих алгоритмів. Тестування проводилось у середовищі із параметрами: було створено пул із 20 роботів і згенеровано набір із 150-200 задач. Усі задачі мали випадкові характеристики, що включали приблизний час виконання (ETA), вимог до можливостей роботів, рівень пріоритету, складність. Такий підхід дозволив наблизити тестування до реальних сценаріїв роботи розподіленої системи з високим рівнем варіативності та непередбачуваності вхідних даних. Для кожного з алгоритмів були проведені однакові серії симуляції на одному наборі даних. Основна мета – оцінити здатність кожного алгоритму забезпечити рівномірне навантаження між роботами, мінімізувати простой та виконати задачі вчасно. Оцінювання відбувалося на основі зібраних логів, які фіксували події призначення, виконання, а також миттєвий стан кожного з роботів. Додатково аналізувалась стабільність роботи алгоритмів при повторному запуску на ідентичному наборі вхідних даних.

4.2.1 Latest Task Finished

Алгоритм демонструє доволі помітну нерівномірність у завантаженні роботів (рисунок 4.1): між мінімальним і максимальним значеннями параметру ЕТА спостерігається різниця у приблизно 30 разів. Високі значення стандартного відхилення та дисперсії спричинені неоднаковими (варіативними) значеннями ЕТА для кожної задачі, що є нормальним для динамічних систем. Але це все одно свідчить про перевантаження певних роботів, в той час коли інші можуть бути взагалі не задіяні (рисунок 4.2).

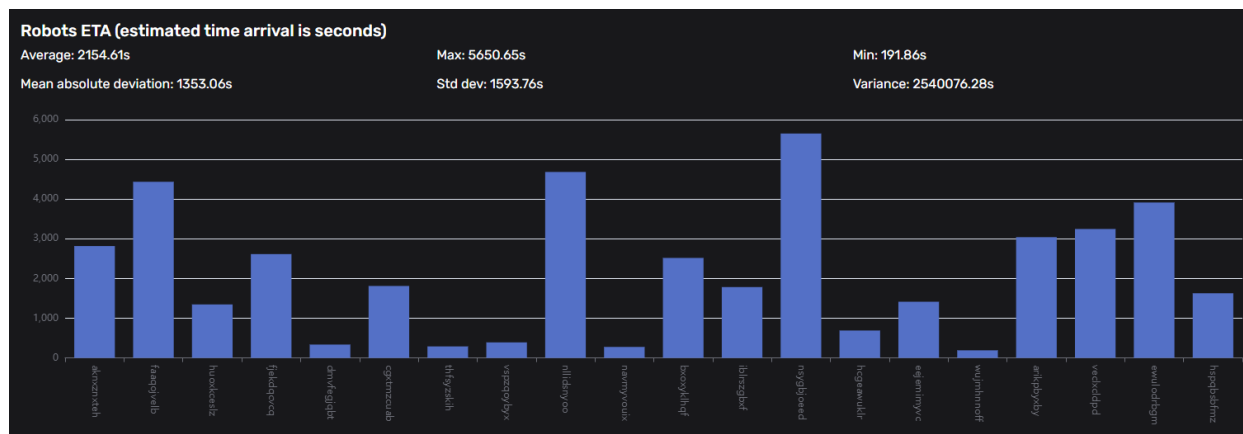


Рисунок 4.1 – ЕТА роботів за алгоритмом Latest Task Finished

Статистичні показники зазначені в таблиці 4.1:

Таблиця 4.1 – Метрики ЕТА алгоритму Latest Task Distribution

Метрика	Значення
Середній ЕТА	2154.6 с.
Максимальний ЕТА	5650 с
Мінімальний ЕТА	191.86 с
Середнє абсолютне відхилення ЕТА	1353 с.
Стандартне відхилення ЕТА	1593 с.
Дисперсія ЕТА	2540076 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота (рисунок 4.2). Рівномірний розподіл є важливою умовою для стабільного функціонування системи. Дані наведено в таблиці 4.2.

Статистичні показники зазначені в таблиці 4.2:

Таблиця 4.2 – Метрики пулу задач алгоритму Latest Task Distribution

Метрика	Значення
Середня к-сть задач	1.45 задач
Максимальна к-сть задач	3 задачі
Мінімальна к-сть задач	1 задача
Середнє абсолютне відхилення к-сті задач	0.54 задач
Стандартне відхилення к-сті задач	0.59 задач
Дисперсія к-сті задач	0.35 задач ²

Середній час розподілу задач алгоритмом: 650 мс.

Попри те, що середнє навантаження коливається в межах 1-3 задач, коефіцієнт варіації все одно свідчить про нерівномірність (рисунок 4.2). Така поведінка характерна для алгоритмів, які пріоритезують завершення задачі окремим агентам.

Час призначення задачі є доволі помірним і дозволяє використовувати алгоритм в умовах помірної динаміки.

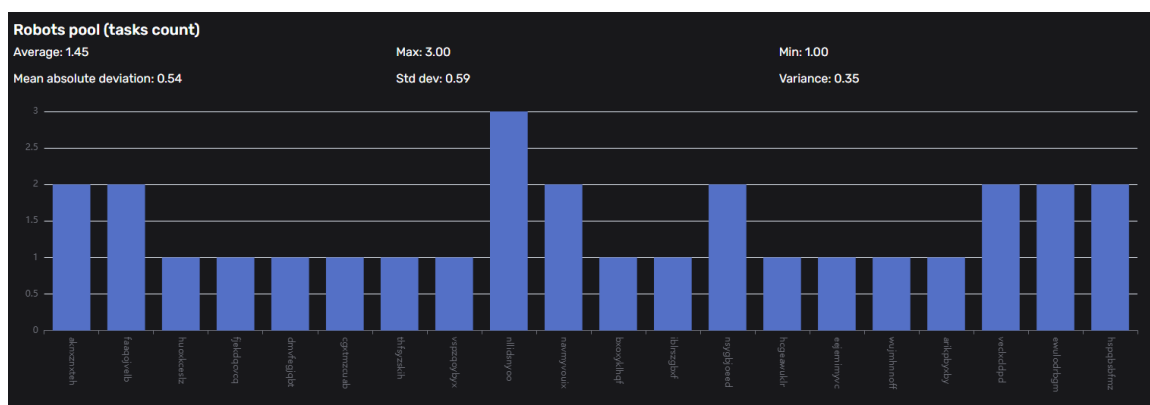


Рисунок 4.2 – Пул (к-сть задач) роботів за алгоритмом Latest Task Finished

4.2.2 Least Connections

Незважаючи на те, що алгоритм покликаний рівномірно розподіляти задачі на основі кількості поточних підключень (активних задач), спостерігається суттєве коливання ETA між роботам – понад 8000 секунд (рисунок 4.3). Високі значення дисперсії і стандартного відхилення свідчать про те, що навіть про схожій кількості задач, їхня тривалість може сильно відрізнятись, що й спричиняє дисбаланс у фактичному навантаженні.

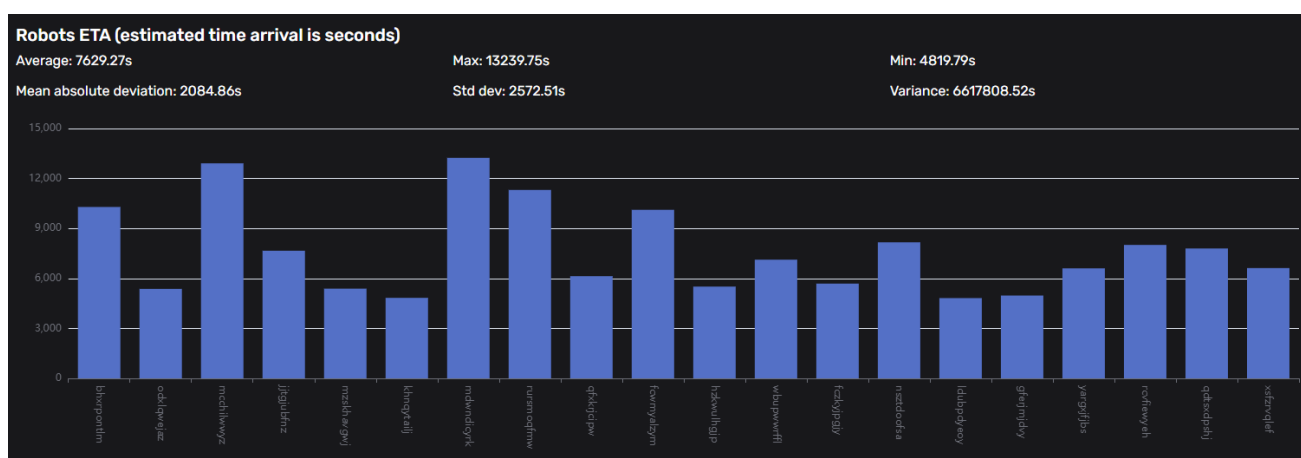


Рисунок 4.3 – ETA роботів за алгоритмом Least Connections

Статистичні показники зазначені в таблиці 4.4:

Таблиця 4.3 – Метрики ETA алгоритму Least Connections

Метрика	Значення
Середній ETA	7269.6 с.
Максимальний ETA	13239.75 с
Мінімальний ETA	4819.79 с
Середнє абсолютне відхилення ETA	2084.86 с.
Стандартне відхилення ETA	2572.51 с.
Дисперсія ETA	6617808 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи (рисунок 4.4).

Статистичні показники зазначені в таблиці 4.4:

Таблиця 4.4 – Метрики пулу задач алгоритму Least Connections

Метрика	Значення
Середня к-сть задач	5 задач
Максимальна к-сть задач	6 задач
Мінімальна к-сть задач	4 задача
Середнє абсолютне відхилення к-сті задач	0.2 задачі
Стандартне відхилення к-сті задач	0.45 задач
Дисперсія к-сті задач	0.2 задач ²

Середній час розподілу алгоритмом: 430 мс.

Алгоритм відмінно справляється з балансуванням задач у термінах їхньої кількості. Відхилення незначні (лише 1 задача), що свідчить про дотримання принципу найменшого навантаження. Це підтверджує, що алгоритм рівномірно розподіляє задачі за кількістю, але не враховує їхню складність чи тривалість (рисунок 4.4). Розподіл задач відбувається швидко, що робить алгоритм придатним для систем з високою динамікою змін.

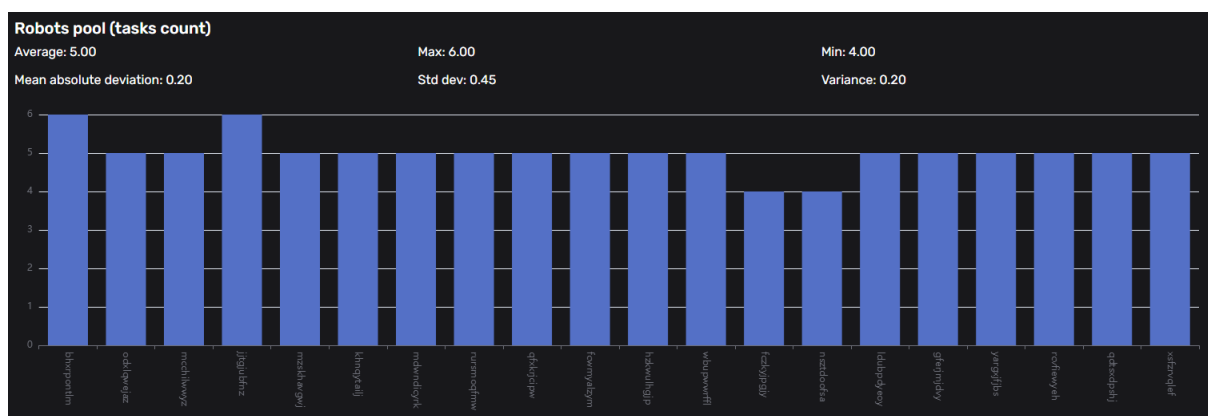


Рисунок 4.4 – Пул (к-сть задач) роботів за алгоритмом Least Connections

4.2.3 Load Balancing

Алгоритм показав достатньо збалансований розподіл по ЕТА – різниця між мінімальним та максимальним значенням не перевищує 3000 с., при середньому значенні приблизно 13700с. Враховуючи варіативність у тривалості задач, це свідчить про ефективне балансування по реальному навантаженню, а не по кількості задач. Стандартне відхилення та дисперсія значно менші, ніж у попередніх алгоритмів, що свідчить про стабільність (рисунок 4.5).

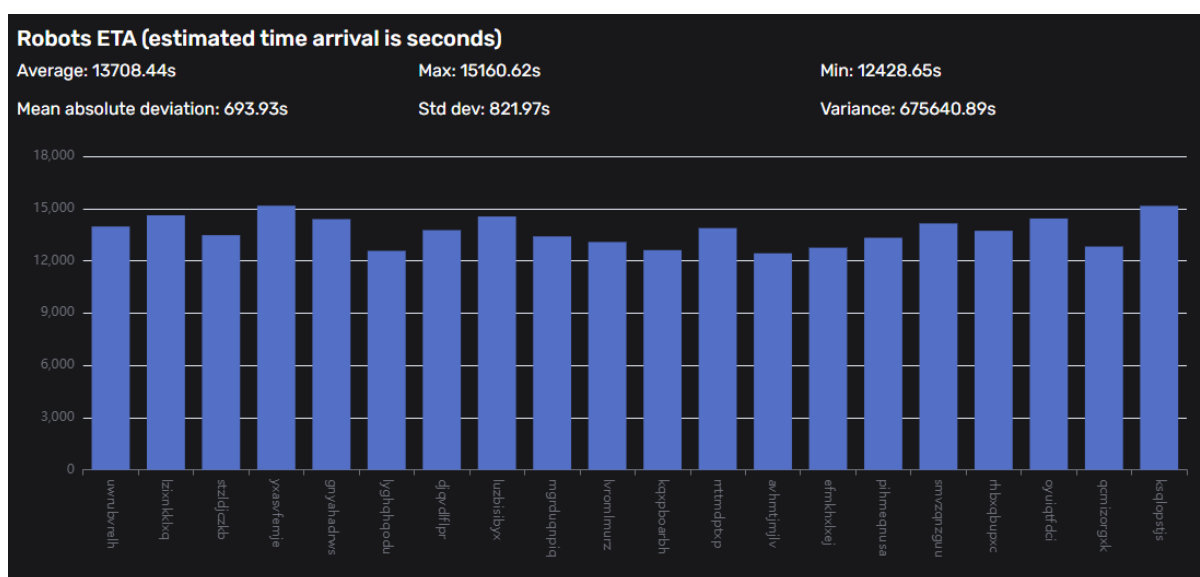


Рисунок 4.5 – ЕТА роботів за алгоритмом Load Balancing

Статистичні показники зазначені в таблиці 4.5:

Таблиця 4.5 – Метрики ЕТА алгоритму Load Balancing

Метрика	Значення
Середній ЕТА	13708.44 с.
Максимальний ЕТА	15160.62 с
Мінімальний ЕТА	12428.65 с
Середнє абсолютне відхилення ЕТА	693.93 с.
Стандартне відхилення ЕТА	821.97 с.
Дисперсія ЕТА	675640с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи (рисунок 4.6).

Статистичні показники зазначені в таблиці 4.6:

Таблиця 4.6 – Метрики пулу задач алгоритму Load Balancing

Метрика	Значення
Середня к-сть задач	5 задач
Максимальна к-сть задач	6 задач
Мінімальна к-сть задач	3 задач
Середнє абсолютне відхилення к-сті задач	0.7 задач
Стандартне відхилення к-сті задач	0.89 задач
Дисперсія к-сті задач	0.8 задач ²

Середній час розподілу алгоритмом: 420 мс.

Незважаючи на відносно більшу дисперсію кількості задач у порівнянні з іншими алгоритмами, даний очевидно не намагається вирівняти саме кількість задач (рисунок 4.6). Натомість, він орієнтується на баланс ETA.

Алгоритм демонструє хорошу продуктивність і є придатним для систем реального часу або систем з великим потоком задач. Швидкість прийняття рішень менша за пів секунди.

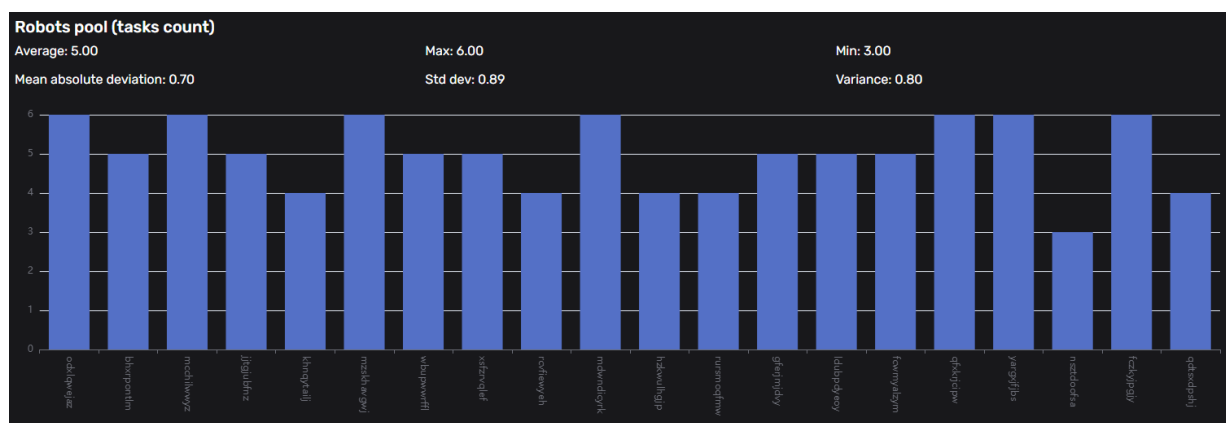


Рисунок 4.6 – Пул (к-сть задач) роботів за алгоритмом Load Balancing

4.2.4 Random

Результати показують значну нерівномірність навантаження між роботами. Один або кілька роботів мають ЕТА близьке до нуля, в той час як інші – перевантажені (до 9.6 тис. секунд). Стандартне відхилення та дисперсія найвищі серед усіх алгоритмів, що вказує на повну відсутність стратегії у розподілі задач. Випадкове призначення не враховує ні поточне навантаження, ні тривалість задач (рисунок 4.7).

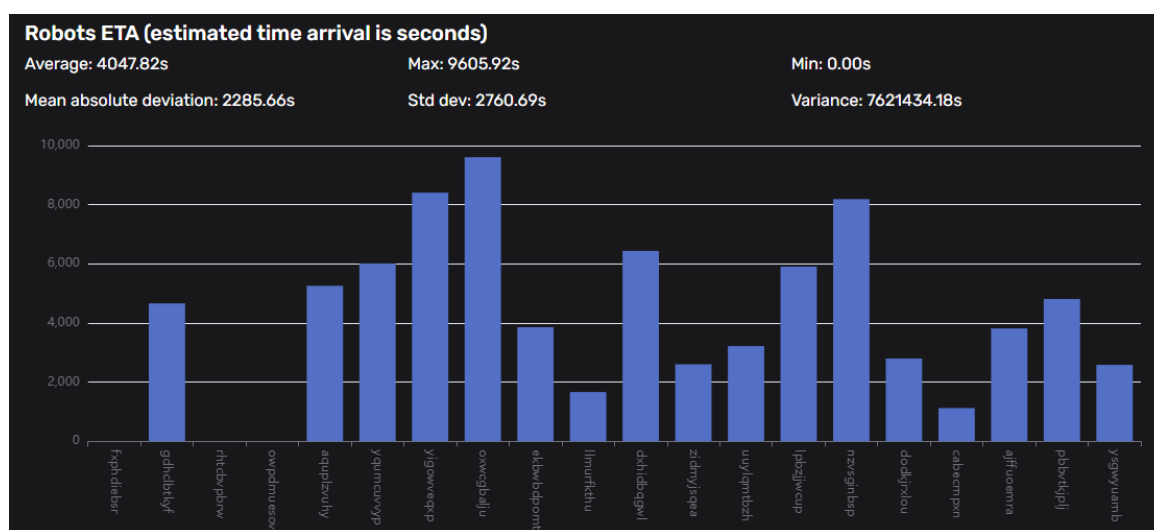


Рисунок 4.7 – ЕТА роботів за алгоритмом Random

Статистичні показники зазначені в таблиці 4.7 :

Таблиця 4.7 – Метрики ЕТА алгоритму Random

Метрика	Значення
Середній ЕТА	4047.82 с.
Максимальний ЕТА	9605.92 с.
Мінімальний ЕТА	0 с.
Середнє абсолютне відхилення ЕТА	2285.53 с.
Стандартне відхилення ЕТА	2760.69 с.
Дисперсія ЕТА	7621434 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи (рисунк 4.8).

Статистичні показники зазначені в таблиці 4.8:

Таблиця 4.8 – Метрики пулу задач алгоритму Random

Метрика	Значення
Середня к-сть задач	2.8 задач
Максимальна к-сть задач	7 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	1.82 задач
Стандартне відхилення к-сті задач	2.14 задач
Дисперсія к-сті задач	4.56 задач ²

Середній час розподілу алгоритмом: 500 мс.

Розподіл задач за кількістю та ЕТА показує, що деякі роботи залишились без задач, тоді як інші мають до 7 задач у черзі. Це є очікуваним результатом для випадкового підходу (рисунк 4.8).

Незважаючи на посередню швидкість прийняття рішень, результати свідчать про низьку ефективність алгоритму. Переваги у швидкості не компенсують суттєвих втрат у балансуванні навантаження.

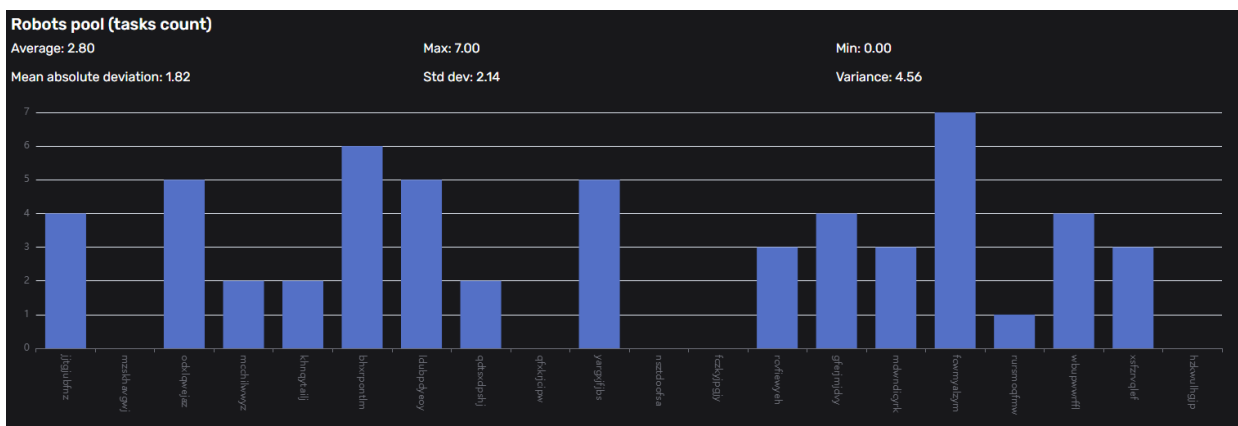


Рисунок 4.8 – Пул (к-сть задач) роботів за алгоритмом Random

4.2.5 Round Robin

Попри ідею рівномірного розподілу задач по черзі, алгоритм демонструє високий рівень дисбалансу по ЕТА. Один або кілька роботів залишились без задач (ЕТА = 0), тоді як інші перевантажені (до 12.9 тис. сек). Це вказує на слабе врахування реального навантаження – при черговому розподілі не береться ні поточна тривалість, ні їх об'єм (рисунок 4.9).

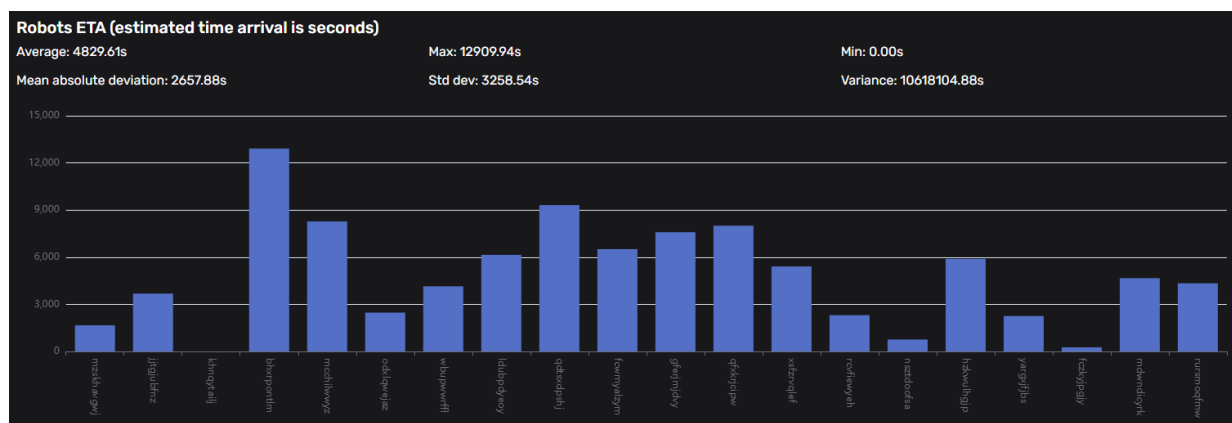


Рисунок 4.9 – ЕТА роботів за алгоритмом Round Robin

Статистичні показники зазначені в таблиці 4.9:

Таблиця 4.9 – Метрики ЕТА алгоритму Round Robin

Метрика	Значення
Середній ЕТА	4829.61 с.
Максимальний ЕТА	12909.94 с.
Мінімальний ЕТА	0 с.
Середнє абсолютне відхилення ЕТА	2657.88 с.
Стандартне відхилення ЕТА	3258.54 с.
Дисперсія ЕТА	10617104 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи (рисунок 4.10).

Статистичні показники зазначені в таблиці 4.10:

Таблиця 4.10 – Метрики пулу задач алгоритму Round Robin

Метрика	Значення
Середня к-сть задач	2.95 задач
Максимальна к-сть задач	7 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	1.55 задач
Стандартне відхилення к-сті задач	1.83 задач
Дисперсія к-сті задач	3.35 задач ²

Середній час розподілу алгоритмом: 500 мс.

Кількість задач на одного робота також сильно варіюється – від повної відсутності до 7 задач. Це свідчить, що при черговому розподілі задач не враховується поточне навантаження (рисунок 4.10).

Алгоритм забезпечує стабільну швидкість прийняття рішень, що можна вважати перевагою. Але швидкість реалізації не компенсує недоліків у балансуванні ресурсів.



Рисунок 4.10 – Пул (к-сть задач) роботів за алгоритмом Round Robin

4.2.6 Ant Colony

Алгоритм демонструє дуже високий розкид у значеннях ЕТА: деякі роботи залишилися повністю без задач, тоді як інші отримали задачі загальною тривалістю понад 13 годин. Це свідчить про нерівномірне навантаження, яке може виникати через локальні оптимуми або нестійкість до випадковості в фазі пошуку (рисунок 4.11).

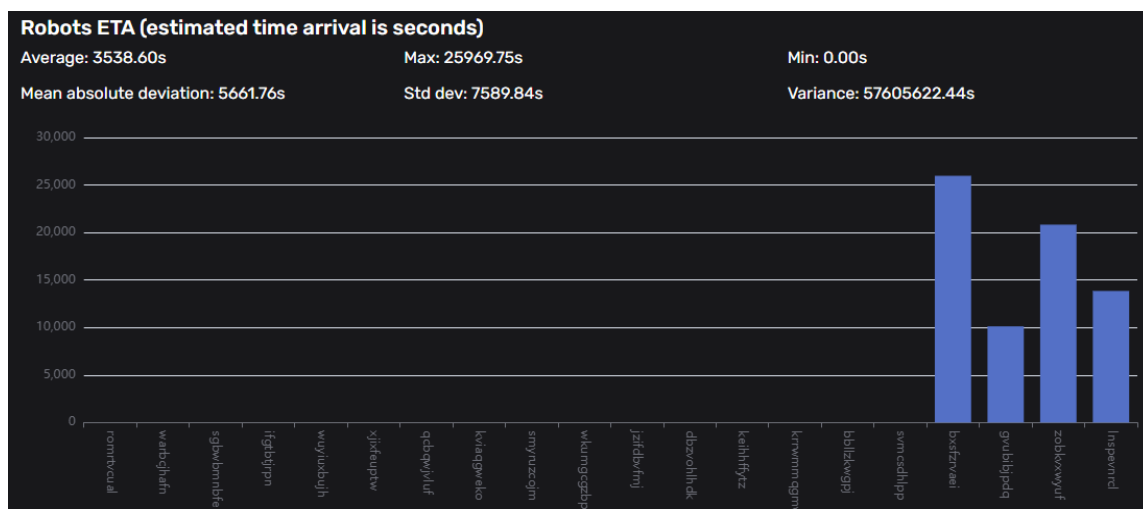


Рисунок 4.11 – ЕТА роботів за алгоритмом Ant Colony

Статистичні показники зазначені в таблиці 4.11:

Таблиця 4.11 – Метрики ЕТА алгоритму Ant Colony

Метрика	Значення
Середній ЕТА	3538.60 с.
Максимальний ЕТА	25969.75 с.
Мінімальний ЕТА	0 с.
Середнє абсолютне відхилення ЕТА	5661.76 с.
Стандартне відхилення ЕТА	7589.84 с.
Дисперсія ЕТА	57605622 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи.

Статистичні показники зазначені в таблиці 4.12:

Таблиця 4.12 – Метрики пулу задач алгоритму Ant Colony

Метрика	Значення
Середня к-сть задач	2.05 задач
Максимальна к-сть задач	15 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	3.28 задач
Стандартне відхилення к-сті задач	4.47 задач
Дисперсія к-сті задач	19.95 задач ²

Середній час розподілу алгоритмом: 600 мс.

Хоча середнє значення к-сті задач на одного робота є низьким, окремі роботи отримали надмірну к-сть задач, що є неприйнятним до рівномірного навантаження. Розподіл задач характеризується високою варіативністю (рисунок 4.12).

Середній час обробки задач є прийнятним, особливо з огляду на складність алгоритму.

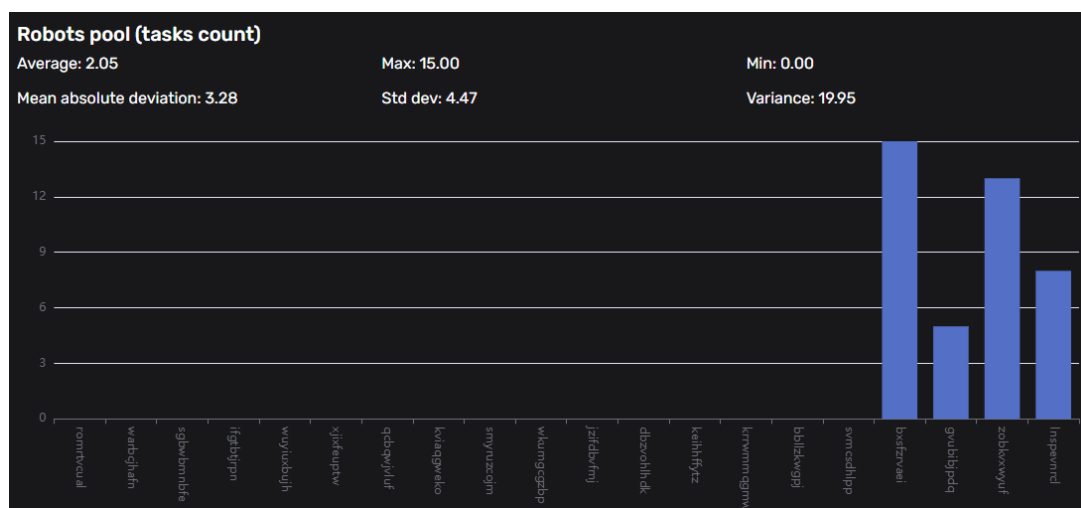


Рисунок 4.12 – Пул (к-сть задач) роботів за алгоритмом Ant Colony

4.2.7 Genetic Task

Максимальне ЕТА понад годину, середнє значення залишилося в межах кількох хвилин. Висока дисперсія пояснюється наявністю одного чи кількох роботів із помітними навантаженням, тоді як більшість роботів мали мінімальну або нульову ЕТА. Це може вказувати на ефективну селекцію, але потребує додаткового аналізу на сталості поведінки (рисунк 4.13).

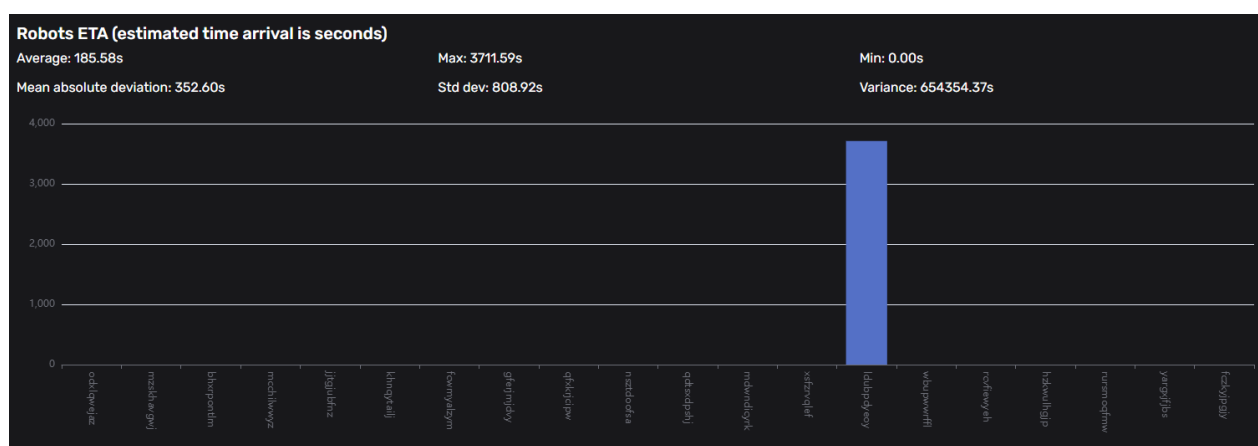


Рисунок 4.13 – ЕТА робіт за алгоритмом Genetic Task

Статистичні показники зазначені в таблиці 4.13:

Таблиця 4.13 – Метрики ЕТА алгоритму Genetic Task

Метрика	Значення
Середній ETA	185.58 с.
Максимальний ETA	3711.59 с.
Мінімальний ETA	0 с.
Середнє абсолютне відхилення ETA	352.6 с.
Стандартне відхилення ETA	808.92 с.
Дисперсія ETA	654354 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи.

Статистичні показники зазначені в таблиці 4.14:

Таблиця 4.14 – Метрики пулу задач алгоритму Genetic Task

Метрика	Значення
Середня к-сть задач	0.1 задач
Максимальна к-сть задач	2 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	0.19 задач
Стандартне відхилення к-сті задач	0.44 задач
Дисперсія к-сті задач	0.19 задач ²

Середній час розподілу алгоритмом: 400 мс.

Алгоритм ефективно працював з пулом задач, обравши найкращого робота після виконання декількох етапів селекції, схрещування та інших процесів ітеративно (рисунок 4.14).

Незважаючи на складність обчислень, характерну для еволюційних алгоритмів, час розподілу задач залишився на низькому рівні, що робить цей підхід придатним для використання в системах реального часу та навіть в умовах високої динаміки.

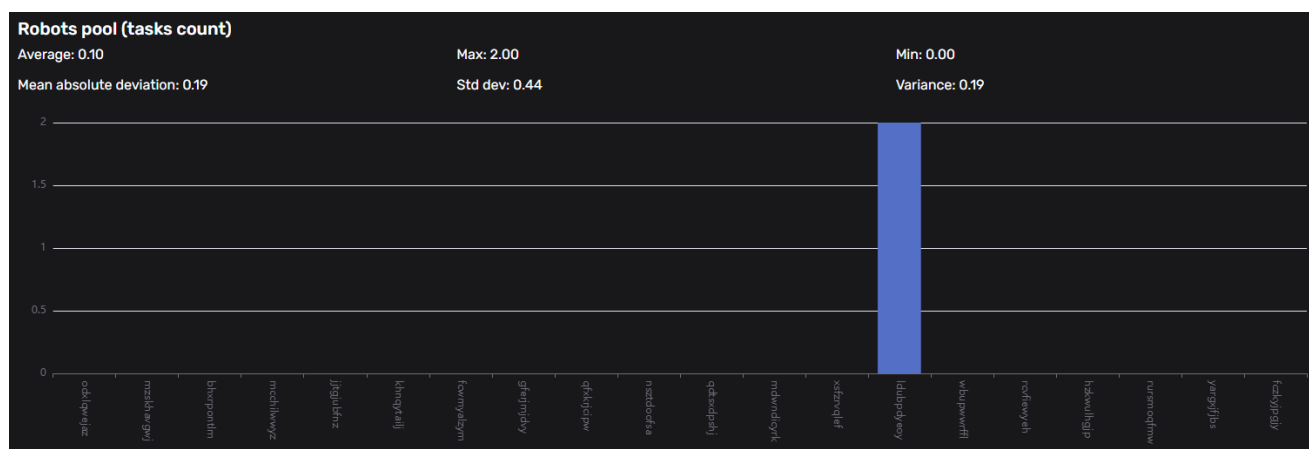


Рисунок 4.14 – Пул (к-сть задач) роботів за алгоритмом Genetic Task

4.2.8 Fuzzy Logic

Алгоритм на основі нечіткої логіки демонструє високий розкид значень ЕТА між роботами. Середнє значення навантаження доволі високе, а максимальне перевищує 25 годин, що вказує на серйозне перевантаження окремих роботів. Високі значення стандартного відхилення та дисперсії свідчать про нестабільність розподілу (рисунок 4.15).

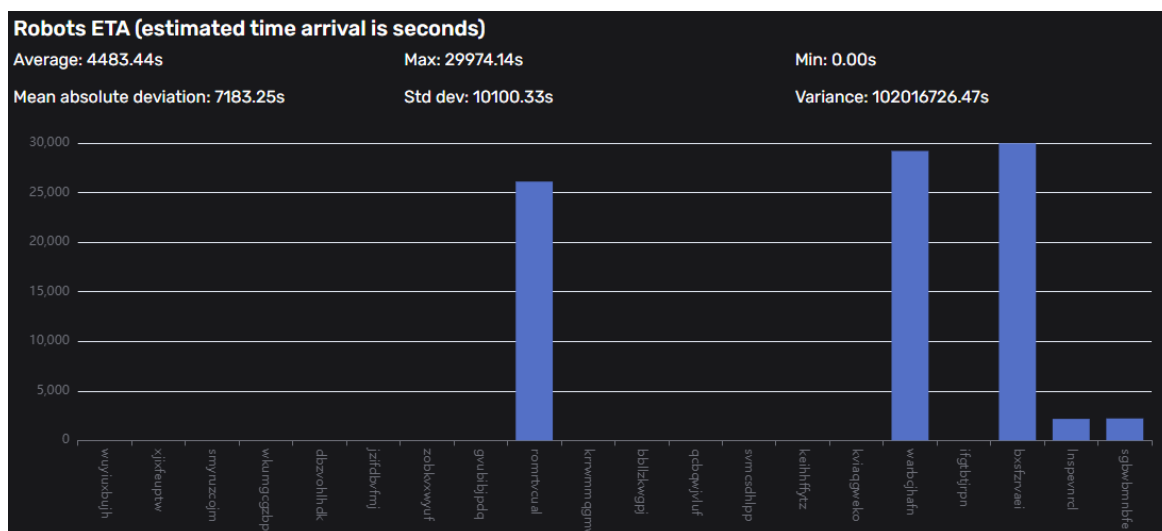


Рисунок 4.15 – ЕТА роботів за алгоритмом Fuzzy Logic

Статистичні показники зазначені в таблиці 4.15:

Таблиця 4.15 – Метрики ЕТА алгоритму Fuzzy Logic

Метрика	Значення
Середній ЕТА	4483.44 с.
Максимальний ЕТА	29974.14 с.
Мінімальний ЕТА	0 с.
Середнє абсолютне відхилення ЕТА	7183.25 с.
Стандартне відхилення ЕТА	10100.33 с.
Дисперсія ЕТА	102016726 с ²

4.2.9 Simulated Annealing

Алгоритм демонструє високу варіативність у завантаженні: деякі роботи залишаються практично без задач, тоді як інші отримують значне завантаження. Це є типовою особливістю метаевристичних методів – вони схильні до виходу в локальні оптимуми залежно від початкових умов. Втім середнє ETA є доволі прийнятним, що свідчить про потенціал цього підходу (рисунок 4.17).

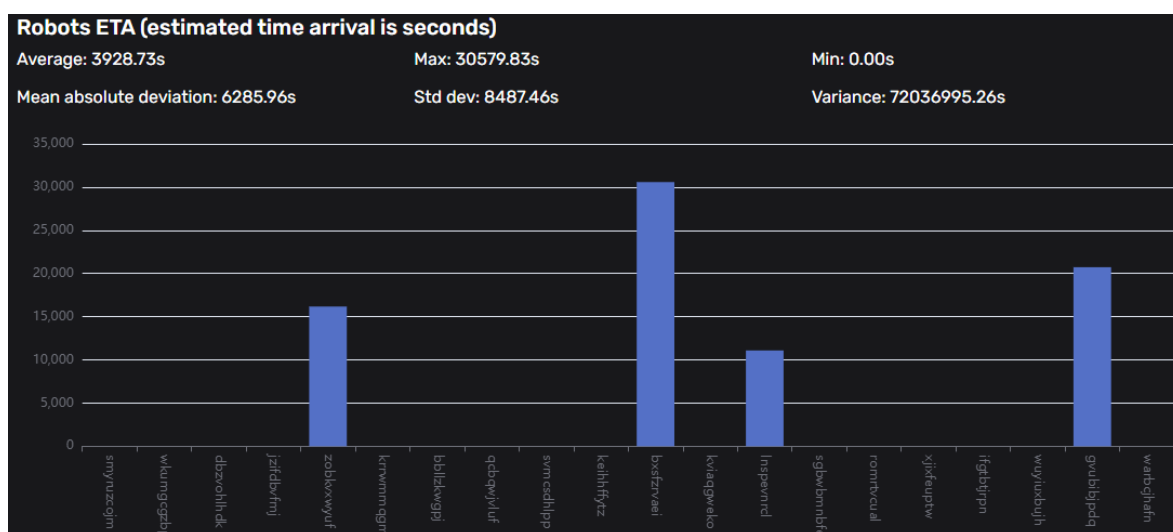


Рисунок 4.17 – ETA роботів за алгоритмом Simulated Annealing

Статистичні показники зазначені в таблиці 4.17:

Таблиця 4.17 – Метрики ETA алгоритму Simulated Annealing

Метрика	Значення
Середній ETA	3928.73 с.
Максимальний ETA	30579.83 с.
Мінімальний ETA	0 с.
Середнє абсолютне відхилення ETA	6285.96 с.
Стандартне відхилення ETA	8487.46 с.
Дисперсія ETA	72036995.26 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи.

Статистичні показники зазначені в таблиці 4.18:

Таблиця 4.18 – Метрики пулу задач алгоритму Simulated Annealing

Метрика	Значення
Середня к-сть задач	2 задачі
Максимальна к-сть задач	15 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	3.2 задачі
Стандартне відхилення к-сті задач	4.22 задач
Дисперсія к-сті задач	17.8 задач ²

Середній час розподілу алгоритмом: 600 мс.

Подібно до ЕТА, к-сть задач відрізняється значним розкидом. Частина роботів багато задач, тоді як інші майже не використовуються. Такі коливання є типовими для алгоритмів, що не використовують жорстке балансування (рисунок 4.18).

Час обробки доволі помірний, особливо враховуючи складність самого алгоритму.

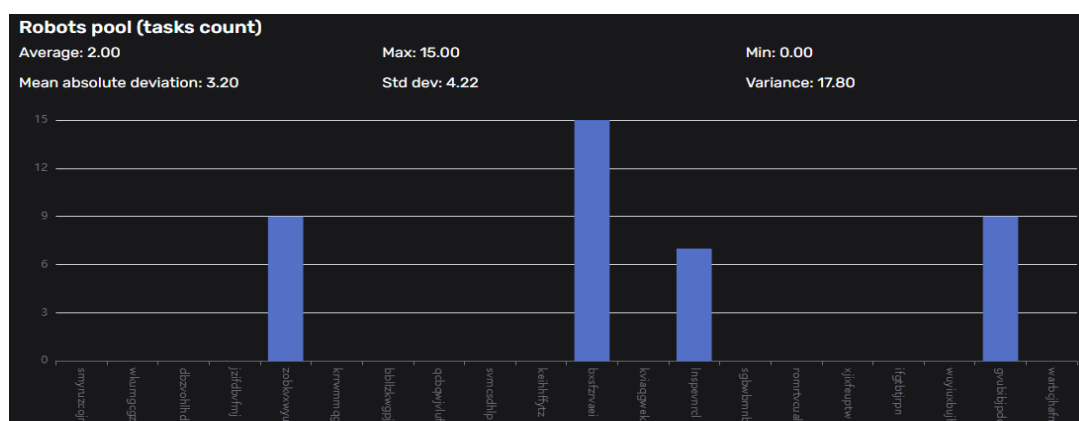


Рисунок 4.18 – Пул (к-сть задач) роботів за алгоритмом Simulated Annealing

4.2.10 Linear Optimization

Алгоритм демонструє середній рівень завантаженості роботів, однак значне стандартне відхилення і висока дисперсія свідчать про те, що частина роботів може залишатись практично без задач, тоді як інші перевантажені. Така ситуація може виникати через математичну специфіку лінійної оптимізації – особливо в умовах, коли розв’язок має багато еквівалентних варіантів (рисунок 4.19).

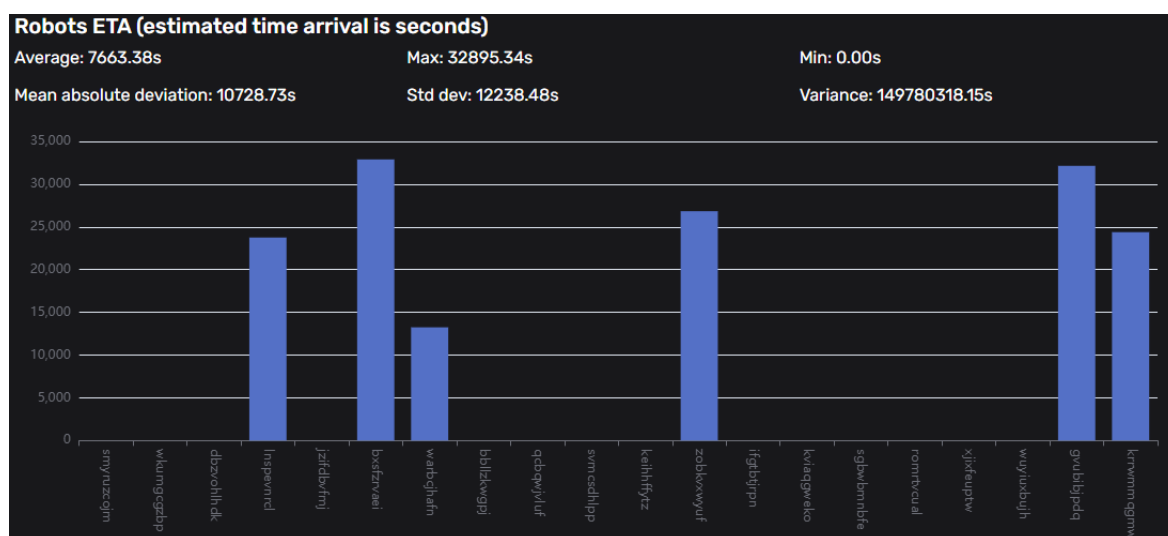


Рисунок 4.19 – ETA роботів за алгоритмом Linear Optimization

Статистичні показники зазначені в таблиці 4.19:

Таблиця 4.19 – Метрики ETA алгоритму Linear Optimization

Метрика	Значення
Середній ETA	7663.38 с.
Максимальний ETA	32895.34 с.
Мінімальний ETA	0 с.
Середнє абсолютне відхилення ETA	10728.73 с.
Стандартне відхилення ETA	12238.48 с.
Дисперсія ETA	149780318.15 с ²

Графік кількості задач ілюструє кількість задач в пулі на кожного робота. Рівномірний розподіл є важливою умовою для стабільного функціонування системи.

Статистичні показники зазначені в таблиці 4.20:

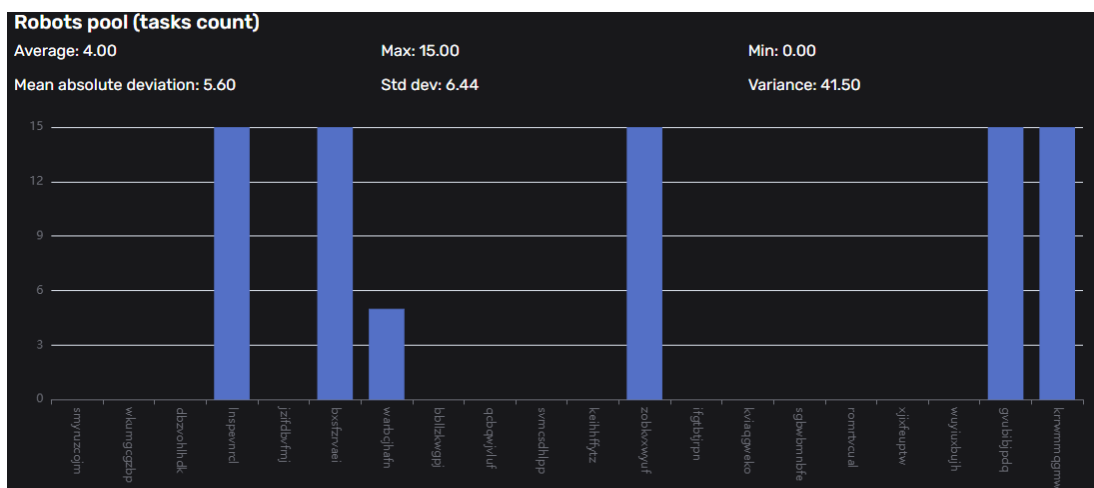
Таблиця 4.20 – Метрики пулу задач алгоритму Linear Optimization

Метрика	Значення
Середня к-сть задач	4 задачі
Максимальна к-сть задач	15 задач
Мінімальна к-сть задач	0 задач
Середнє абсолютне відхилення к-сті задач	5.6 задач
Стандартне відхилення к-сті задач	6.44 задач
Дисперсія к-сті задач	41.5 задач ²

Середній час розподілу алгоритмом: 550 мс.

Розподіл задач між роботами також демонструє нерівномірність. При середньому значенні у 1.3 задачі на робота, окремі мають до 15 задач, що створює ризик локального перевантаження (рисуюнок 4.20).

Час призначення задач у межах 550 мс є доволі помірним, що демонструє життєздатність підходу в умовах розподілу в режимі реального часу



Рисуюнок 4.20 – Пул (к-сть задач) роботів за алгоритмом Linear Optimization

Висновки до розділу 4

Протестовані алгоритми охоплюють різні підходи – від простих евристик до складних метавевристичних методів. Завдяки збору метрик, стало можливим об’єктивно оцінити продуктивність кожного методу в різних умовах, включаючи врахування різної тривалості задач. Детальне порівняння зображено в таблиці 4.21.

Таблиця 4.21 – Порівняння алгоритмів

Алгоритм	Переваги	Недоліки	Найкращі сценарії використання
Latest Task Finished	Добре збалансоване навантаження; Низьке середнє ЕТА	Не враховує ЕТА; Повільніше за прості евристики	Роботи з однорідними задачами та частими оновленнями статусу
Least Connections	Мінімізація черг; Простий у реалізації	Ігнорує час виконання; Високий ЕТА	Високонавантажені системи з короткими задачами
Load Balancing	Добра рівномірність ЕТА; Низька дисперсія	Не завжди точний при довгих задачах	Загальні сценарії, де важливе зменшення середнього часу
Random	Найпростіша реалізація; Низьке навантаження на систему	Висока дисперсія ЕТА; Не передбачуваний	Тестування, стресові ситуації, базовий фолбек-механізм
Round Robin	Рівномірний розподіл; Простий	Ігнорує час задач; Перевантажує робітників	Сценарії з однаковими роботами та фіксованим часом виконання задач
Ant Colony	Адаптивність; Добре масштабується	Висока дисперсія ЕТА;	Динамічні середовища з багатьма змінними, навчанням на історичних даних
Genetic Task	Низький ЕТА; Добре завантаження робітників	Потребує час на стабілізацію	Оптимізація розклад для складних задач, де важлива мінімізація часу

Fuzzy Logic	Гнучкість;	Нестабільні результати	Гібридні системи з нечіткою природою вхідних параметрів
Simulated Annealing	Пошук оптимуму; Гнучкість	Високий ЕТА	Планування в складних задачах, коли важливо уникнути локальних мінімумів
Linear Optimization	Низьке середня ЕТА	Потребує моделі задачі	Сценарії, де цільова функція та обмеження чітко визначені і можуть бути лінійно змодельовані

Жоден з алгоритмів не є універсально найкращим – ефективність залежить від конкретного контексту, типу задач, тривалості виконання, оновлень станів робіт та пріоритетів системи (час, рівномірність, адаптивність тощо).

Для досягнення максимальної продуктивності у реальних сценаріях доцільно використовувати гібридні підходи, наприклад:

– Genetic + Load Balancing – для оптимізації розкладу з урахуванням рівномірності;

– Fuzzy + Ant Colony – для адаптивних сценаріїв – для адаптивних сценаріїв із динамічними середовищем;

– Linear + Rebalancer – для високоточних призначень з періодичним вирівнюванням навантаження.

Наявність розширюваної архітектури дозволяє експериментувати з такими комбінаціями та в майбутньому впроваджувати алгоритми машинного навчання, засновані на поведінкових паттернах системи.

ВИСНОВКИ

У результаті виконання проєкту було реалізовано інтелектуальну систему автоматизованого розподілу завдань для багатокомпонентного середовища з автономними роботами. Метою проєкту стало удосконалення та покращення процесу призначення задач у багатоагентних системах шляхом впровадження гнучкого механізму прийняття рішень з підтримкою різних алгоритмів розподілу, які враховуються характеристики задач, стан та конфігурації виконавців та специфіку середовища. Було здійснено огляд сучасних централізованих і децентралізованих підходів, оптимізаційних та евристичних методів.

У процесі дослідження були визначені ключові проблеми в сучасних підходах до розподілу задач: обмеження централізованих систем, відсутність адаптивності, складність у врахуванні контексту середовища та гетерогенності агентів. На основі аналізу предметної області та сучасних рішень було сформовано вимоги до проєктованої системи.

На основі зібраних вимог була спроектована та реалізована повнофункціональна система з такими ключовими компонентами:

- централізований сервер із REST API, який дозволяє управляти користувачами, роботами, категоріями та завданнями;
- конфігурована клієнтська частина для адміністрування системи;
- модуль алгоритмічного розподілу задач, який підтримує набір різних алгоритмів: від простих евристик (Random, Round Robin, Least Connections) до складних методів штучного інтелекту (Ant Colony, Genetic Algorithm, Simulated Annealing, Fuzzy Logic, Linear Optimization).

Ключовою особливістю системи є можливість конфігурації алгоритмів залежно від особливостей задач і середовища. Особлива увага приділялась тому, щоб більшість алгоритмів були легко замінним та мав параметри конфігурації на різних рівнях. Це дозволило адаптувати систему під різні сценарії використання – від промислових складів до мобільних розподілених платформ. Було реалізовано ряд тестів для перевірки роботи кожного з алгоритмів у симульованих умовах.

Результати тестування показали, що:

- найпростіші алгоритми (Random, Round Robin) забезпечують швидке призначення, але не враховують завантаження роботів;
- евристичні методи (Latest Task Finished, Load Balancing) дозволяють краще балансувати навантаження;
- алгоритми на основі ШІ (Genetic, Ant Colony, Simulated Annealing) демонструють вищу якість розподілу в складних середовищах, але мають більшу обчислювальну складність;
- Fuzzy Logic та Linear Optimization показали одні з найкращих результатів у багатокритеріальних сценаріях, де необхідно враховувати як властивості роботів, так і контекст задачі.

Таким чином, розроблена система вирішує задачу автоматизованого, адаптивного розподілу завдань у багатокомпонентному середовищі, дозволяючи знизити простой, підвищити рівномірність завантаження агентів і покращити загальну продуктивність. Вона є гнучкою платформою, яку можна адаптувати під реальні потреби робототехнічних систем з різними вимогами до призначення задач. Подальший розвиток системи передбачає глибшу інтеграцію з реальними пристроями, використання розподілених обчислень і навчання моделей розподілу на основі історичних даних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fang Z., Ma T., Huang J., Niu Z., Yang F. Efficient Task Allocation in Multi-Agent Systems Using Reinforcement Learning and Genetic Algorithm. MDPI. URL: <https://www.mdpi.com/2076-3417/15/4/1905> (дата звернення: 03.04.2025).
2. Liu H., Zhang Y., Tang Y., Wang X. A Novel Multi-Robot Task Allocation Model in Marine Plastics Cleaning Based on Replicated Dynamics. MDPI. URL: <https://www.mdpi.com/2071-1050/15/6/5074> (дата звернення: 01.04.2025).
3. Bronsdon C. Multi-Agent Coordination Strategies: From Theory to Practice. Galileo. URL: <https://www.galileo.ai/blog/multi-agent-coordination-strategies> (дата звернення: 10.04.2025).
4. Braquet M., Bakolas E. Greedy Decentralized Auction-based Task Allocation for Multi-Agent Systems. URL: <https://martinbraquet.com/wp-content/uploads/Greedy-Decentralized-Auction-based-Task-Allocation-for-Multi-Agent-Systems.pdf> (дата звернення: 11.04.2025).
5. Purwar A. Automated Planning Tool Makes Work Order Allocation More Efficient. Amazon Science. URL: <https://www.amazon.science/blog/automated-planning-tool-makes-work-order-allocation-more-efficient> (дата звернення: 13.04.2025).
6. Petzoldt C., Niermann D., Maack E., Sontopski M., Vur B., Freitag M. Digital Twin-Based Task Allocation for Multi-Agent Systems in Production Logistics. MDPI. URL: <https://www.mdpi.com/2076-3417/12/24/12645> (дата звернення: 17.04.2025).
7. Amazon's Robot Army Is Watching: New Tools Boost Fulfillment Vigilance. Tomorrow Desk. URL: <https://tomorrowdesk.com/vigilance/amazon-robot-arm> (дата звернення: 25.04.2025).
8. Elmogy A. M. Multi-Robot Task Allocation: A Review of the State-of-the-Art. URL: https://www.researchgate.net/publication/277075091_Multirobot_Task_Allocation_A_Review_of_the_State-of-the-Art (дата звернення: 30.04.2025).
9. Multi-Agent Coordination across Diverse Applications: A Survey. arXiv. URL: <https://arxiv.org/html/2502.14743v1> (дата звернення: 30.04.2025).

10. Consensus-Based Fast and Energy-Efficient Multi-Robot Task Allocation. ScienceDirect. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0921889022001592> (дата звернення: 01.05.2025).
11. Task Allocation in Multi-Robot System Using Resource Sharing. PLOS ONE. URL: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0267982> (дата звернення: 01.05.2025).
12. Yuan Y., Yang P., Jiang H., Shi T. A Multi-Robot Task Allocation Method Based on the Synergy of the K-Means++ Algorithm and the Particle Swarm Algorithm. MDPI. URL: <https://www.mdpi.com/2313-7673/9/11/694> (дата звернення: 10.05.2025).
13. Patel R., Rudnick-Cohen E., Azarm S., Otte M., Xu H., Herrmann J. W. A Comparative Study of Task Allocation Algorithms for Multi-Robot Systems. URL: <https://user.eng.umd.edu/~jwh2/papers/Patel-ICRA-2020.pdf> (дата звернення: 12.05.2025).
14. Patel R., Rudnick-Cohen E., Azarm S., Otte M., Xu H., Herrmann J. W. Decentralized Task Allocation in Multi-Agent Systems Using a Decentralized Genetic Algorithm. Academia.edu. URL: https://www.academia.edu/83199647/Decentralized_Task_Allocation_in_Multi_Agent_Systems_Using_a_Decentralized_Genetic_Algorithm (дата звернення: 15.05.2025).
15. Gong A., Yang K., Lyu J., Li X. A Two-stage reinforcement Learning-based Approach for Multi-entity Task Allocation. arXiv. URL: <https://arxiv.org/abs/2407.00496> (дата звернення 17.05.2025).