НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

“На правах рукопису”
УДК 004.42

До захисту допущено:
В.О. Завідувача кафедри
_____ Михайло Новотарський
“__” _____2024 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”
на тему: «Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі»

Виконав:
студент ІІ курсу, групи ІМ-31мп
Нікіта Лазаренко _____

Науковий керівник:
Проф. каф. ОТ, д.т.н.
Анатолій Сергієнко _____

Консультант з нормоконтролю:
Проф. каф. ОТ, д.т.н.
Валерій Жабін _____

Рецензент:
ас. каф. СП і СКС, д.філ.
Олексій Молчанов _____

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

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки Кафедра обчислювальної техніки

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

В.О. Завідувача кафедри

Михайло НОВОТАРСЬКИЙ

“____”____2024 р

ЗАВДАННЯ

на магістерську дисертацію студенту

Лазаренку Нікіті В'ячеславовичу

1. Тема дисертації: “Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі”, Науковий керівник дисертації: професор каф. ОТ, д.т.н. Сергієнко А.М., затверджені наказом по університету від «08» листопада 2024 р. № 5016-с
2. Термін подання студентом дисертації 28.11.2024
3. Об’єкт дослідження: процеси управління та виконання задач на графічних процесорах у розподілених системах, включаючи технічні, програмні та організаційні аспекти, які забезпечують ефективну та оптимізовану роботу багатокористувацького менеджера задач.
4. Предмет дослідження: методи та засоби, необхідні для розробки та впровадження багатокористувацького менеджера задач графічного процесора, орієнтованого на запуск програмного коду в розподіленій системі.
5. Перелік завдань, які потрібно розробити: дослідити та визначити функціональність, яка задовольнятиме експериментаторів, що бажають здійснювати запуск коду на розподіленій системі, реалізувати відповідну платформу, здійснити її тестування.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: 35 рисунків.
7. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Проф. каф. ОТ д.т.н. Жабін Валерій Іванович		
Розділ 1	Проф. каф. ОТ, д.т.н. Сергієнко Анатолій Михайлович		
Розділ 2	Проф. каф. ОТ, д.т.н. Сергієнко Анатолій Михайлович		
Розділ 3	Проф. каф. ОТ, д.т.н. Сергієнко Анатолій Михайлович		
Розділ 4	Проф. каф. ОТ, д.т.н. Сергієнко Анатолій Михайлович		

9. Дата видачі завдання 02.09.2024

календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми роботи	01.09.2024 – 08.09.2024	
2	Вивчення та аналіз завдання	08.09.2024 – 17.09.2024	
3	Вибір технологій для розробки системи	17.09.2024 – 23.09.2024	
4	Розробка архітектури та загальної структури системи	23.09.2024 – 13.10.2024	
5	Програмна реалізація	13.10.2024 - 15.11.2024	
6	Тестування системи	15.11.2024 - 18.11.2024	

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
7	Оформлення пояснювальної записки	18.11.2024 - 25.11.2024	
8	Захист	16.12.2024	

Студент _____ Нікіта Лазаренко

Науковий керівник _____ Анатолій Сергієнко

АНОТАЦІЯ

Магістерська дисертація присвячена розробці багатокористувацького менеджера задач для графічних процесорів, орієнтованого на ефективне управління обчислювальними ресурсами у розподілених системах. У даній роботі було розглянуто наявні системи-аналоги, обґрунтовано необхідність створення розробленої системи. Представлене програмне забезпечення надає можливість управляти задачами та здійснювати їх планування, підтримує масштабованість, дозволяє розподіляти ресурси одного процесора, а також має зручний графічний інтерфейс.

ABSTRACT

The master's thesis is devoted to the development of a multi-user task manager for GPUs, focused on the efficient management of computing resources in distributed systems. In this work, the existing analog systems were reviewed and the necessity of creating the developed system was substantiated. The presented software provides the ability to manage and schedule tasks, supports scalability, allows allocating resources of a single processor, and has a user-friendly graphical interface.

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі

студентом Лазаренком Нікітою В'ячеславовичем

Обсяг та структура роботи: Дана магістерська дисертація містить наступні розділи: вступ, чотири основні розділи з пунктами й підпунктами. Загалом магістерська робота містить 106 сторінок, 35 рисунків, а також 17 таблиць. При написанні магістерської роботи також було використано 31 джерело.

Актуальність роботи: Актуальність проєкту полягає у зростаючій потребі в інструментах для підвищення ефективності використання обчислювальних ресурсів, зокрема графічних процесорів, у розподілених системах. У сучасних умовах розвитку технологій, таких як штучний інтелект, машинне навчання та моделювання великих даних, вимоги до швидкості обчислень та ефективності розподілу ресурсів постійно зростають.

Розробка багатокористувацького менеджера задач графічного процесора вирішує критичні завдання, як-от підвищення продуктивності обчислень, оптимізація розподілу навантаження між користувачами та системами, а також забезпечення прозорості та контролю над виконанням задач. Завдяки впровадженню інноваційних підходів цей проєкт відповідає викликам сучасного ринку високопродуктивних обчислень, роблячи його актуальним як для індивідуальних розробників, так і для великих організацій.

Мета роботи: створення багатокористувацького менеджера задач для підвищення ефективності використання обчислювальних ресурсів у системах, що містять декілька робочих станцій з одним або більше графічних процесорів.

Завдання дослідження: під час створення системи було виконано такі етапи:

- дослідження наявних систем менеджменту обчислювальних ресурсів
- вибір технологій для створення системи менеджменту обчислювальних ресурсів
- розробка архітектури системи
- написання програмного коду системи
- тестування та випробування системи

Об’єкт дослідження: процеси управління та виконання задач на у розподілених системах, включаючи технічні, програмні та організаційні

аспекти, які забезпечують ефективну та оптимізовану роботу багатокористувацького менеджера задач.

Предмет дослідження: Предметом дослідження є методи та засоби, необхідні для розробки та впровадження багатокористувацького менеджера задач графічного процесора в розподіленій системі.

Результати дослідження: нова система менеджменту обчислювальних ресурсів, в якій інтегровані в інтеграції інноваційних підходів до управління обчислювальними ресурсами графічних процесорів у багатокористувацькому розподіленому середовищі. Робота включає в себе алгоритми динамічного управління ресурсами, що враховують специфіку задач користувачів, реальні навантаження, а також можливість розподіляти ресурси одного графічного процесора, маючи при цьому зручний графічний інтерфейс.

Практична цінність: Практична цінність даної роботи полягає в здатності створеної системи менеджменту обчислювальних ресурсів забезпечити користувачів ефективним інструментом для управління ресурсами графічних процесорів у розподілених системах, зокрема завдяки оптимізації ресурсів, підвищенню продуктивності розробників і зниженню часу виконання задач.

Ключові слова: РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, ГРАФІЧНИЙ ПРОЦЕСОР, МЕНЕДЖЕР ЗАДАЧ, БАГАТОКОРИСТУВАЦЬКА СИСТЕМА, ОПТИМІЗАЦІЯ РЕСУРСІВ.

Пояснювальна записка
до магістерської дисертації

на тему “Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі”

Зміст

СПИСОК СКОРОЧЕНЬ **12**

ВСТУП **14**

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **16**

1.1. Визначення понять **16**

1.2. Менеджери задач графічного процесора **17**

1.2.1. Kubernetes 17

1.2.2. Slurm. 19

1.2.3. Google Colab 20

1.3. Порівняння існуючих технологій **23**

1.4 Вимоги до програмного забезпечення **24**

1.5. Постановка задачі створення системи менеджменту обчислювальних ресурсів **25**

Висновок до розділу 1 ..... **26**

РОЗДІЛ 2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ **27**

2.1. Аналіз структури програмного забезпечення **27**

2.2. Технічна реалізація програмного забезпечення **32**

2.2.1. Засоби розробки. 32

2.2.2. Реалізація клієнт-серверної взаємодії. 36

2.2.3. Робота системи через користувацькі інтерфейси. 44

Висновок до розділу 2 ..... **53**

РОЗДІЛ 3 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ **54**

3.1. Юніт-тестування **54**

3.2. Системне тестування **58**

3.3. Якісне порівняння результатів роботи системи й аналогів **59**

3.3.1. Взаємодія зі Scrum. 59

3.3.2. Взаємодія з K8s. 60

3.3.3. Взаємодія з розробленою системою. 61

Висновок до розділу 3 ..... **63**

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЄКТУ **64**

4.1. Опис реалізації стартапу 64
4.2. Загальна ідея стартапу 67
4.3. Технічний аудит проєкту 73
4.4. Аналіз та вивчення ринкових можливостей для старту проєкту 75
4.5. Аналіз та розробка ринкової стратегії 83
Висновок до розділу 4 86
ВИСНОВКИ 87
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 89
ДОДАТОК 1 93
Акт запровадження 93
Переклад акту запровадження 94
Частина програмного коду 95

СПИСОК СКОРОЧЕНЬ

ЦП – Центральний процесор

ГП – Графічний процесор

GUI – Graphical User Interface (Графічний інтерфейс)

K8s – Kubernetes

Slurm – Simple Linux Utility for Resource Management

CLI – Command Line Interface

ОС – Операційна система

ТП – Тензорний процесор

REST – Representational State Transfer

API – Application Programming Interface

NPM – Node Package Manager (Менеджер пакетів)

WSL – Windows Subsystem for Linux

БД – База даних

IP – Internet Protocol

HTTP – Hypertext Transfer Protocol

URL – Uniform Resource Locator

JSON – JavaScript Object Notation

UML – Unified Modeling Language

ЮТ – Юніт-тестування

TRL – Technology Readiness Levels

AI – Artificial Intelligence

ML – Machine Learning

HPC – High Performance Computing

CUDA – Compute Unified Device Architecture

OpenCL – Open Computing Language

CCPA – California Consumer Privacy Act

ВСТУП

Традиційно, у обчислювальній техніці за всі основні обчислення, управління оперативною пам’яттю, виконання програм, а також координацію роботи всіх компонентів відповідає центральний процесор (ЦП). Хоч він має високу універсальність і може виконувати широкий спектр задач, він зазвичай має обмежену кількість ядер. На відміну від нього, графічний процесор (ГП) містить значно більшу кількість ядер з невеликими обчислювальними можливостями, але які здатні виконувати багато однотипних задач одночасно.

Потреба у виконанні багатьох простих паралельних операцій була зумовлена необхідністю опрацювання графіки для відображення складних візуальних сцен у комп’ютерних програмах, зокрема іграх. Проте з часом розвиток технологій призвів до розширення можливостей ГП [1].

Сьогодні ГП використовуються багатьма організаціями та науково-дослідницькими інститутами [2], що спричинило появу програмного забезпечення, яке надає змогу розподіляти обчислювальні ресурси систем, заснованих на ГП. Програми на базі паралельних обчислень використовуються у різноманітних проєктах, де необхідні високопродуктивні обчислення й оптимізація використання ресурсів. Вони підходять для завдань у галузі штучного інтелекту (AI), зокрема машинного навчання (ML), де виконується тренування моделей на великих наборах даних, а також для аналізу великих даних у фінансових [3], медичних [4] і соціальних [5] дослідженнях. У наукових дослідженнях подібні системи використовуються для складних симуляцій, таких як моделювання клімату [6], молекулярна динаміка [7] й аналіз генетичних даних [8]. У сфері візуалізації вони, своєю чергою, є корисними для рендерингу анімацій, створення віртуальної реальності [9] або архітектурних візуалізацій. У криптографії подібні програми сприяють обчисленню криптографічних алгоритмів [10] і підтримці невеликих блокчейн-проєктів. У сфері кібербезпеки вони допомагають аналізувати мережевий трафік для виявлення загроз [11], а в

системах реального часу – в розпізнаванні образів, зокрема в автономних транспортних засобах [12].

Як можна побачити, подібні системи застосовуються в багатьох сферах. Незважаючи на це, вони не завжди здатні забезпечити ефективний і справедливий розподіл ресурсів — деякі з них не оптимізовані для використання одного процесора кількома користувачами, що є актуальним для невеликих організацій. Інші ж позбавлені графічного інтерфейсу (GUI).

Отже, актуальною є задача створення багатокористувацького менеджера задач для ефективного використання обчислювальних ресурсів (у системах, що містять декілька робочих станцій з одним або більше ГП), який даватиме змогу розподіляти навантаження від одного або декількох процесів між багатьма обчислювальними машинами або користувачами, які використовують власні обчислювальні потужності. Такий менеджер задач матиме зрозумілий GUI, що забезпечить інтуїтивне використання системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Визначення понять

Коли мова заходить про менеджер задач графічного процесора, перш за все варто визначити такі поняття, як менеджер задач, процесор (зокрема графічний) та крупнозернистий паралелізм, оскільки саме на них базується дана робота.

Менеджер задач – це програма, яка використовується для організації, відстеження та керування різними завданнями в межах системи. Він служить центром для управління робочим процесом, розподілу обов'язків, моніторингу прогресу та забезпечення виконання завдань у визначені терміни. Крім того, він здатний надсилати сповіщення про стан системи та конкретних задач.

Процесор – це будь-який пристрій або компонент у системі, який опрацьовує вхідні дані або маніпулює ними для отримання результату.

Графічний процесор (ГП, або GPU) – це процесор, призначений для прискорення, створення та відтворення зображень, відео та графіки. Незважаючи на те, що початково ГП були розроблені для роботи з рендерингом для комп'ютерних ігор, з часом вони еволюціонували до високопаралельних процесорів з обчислювальною потужністю, що значно перевищує їхні початкові можливості, орієнтовані на графіку.

Крупнозернистий паралелізм базується на терміні «зернистість», що є середнім проміжком часу між актами синхронізації паралельних процесів. У крупнозернистому паралелізмі цей проміжок часу становить не менше десятків мікросекунд. На прикладі подібних систем крупнозернистим паралелізмом може бути паралельне виконання кількох задач з різними параметрами, оскільки задачі не повинні синхронізувати результати виконання між собою частіше ніж раз на секунду.

1.2. Менеджери задач графічного процесора

1.2.1. Kubernetes.

У контексті використання ГП у системі Kubernetes, або K8s [13] дозволяє керувати в межах кластеру машин робочим навантаженням у вигляді контейнерів, які є основними одиницями для запуску програм.

Незважаючи на те, що K8s не був розроблений саме для роботи з ГП, він має досить великі можливості, які роблять його популярним вибором для управління завданнями, що виконуються ГП у розподілених системах.

K8s містить менеджер задач, що враховує вимоги до ГП під час призначення завдань, гарантуючи, що робочі навантаження, які потребують ресурсів ГП, розподіляються між вузлами, обладнаними сумісними ГП. K8s також надає механізми для ізоляції ресурсів ГП, щоб запобігти перешкодам між завданнями, що виконуються на одному вузлі.

K8s забезпечує динамічне масштабування, дозволяючи додавати або видаляти вузли з підтримкою графічного процесора – залежно від вимог робочого навантаження.

Такі бібліотеки, як TensorFlow, PyTorch та подібні їм, які використовують ГП для глибокого навчання та наукових обчислень [14], можна легко інтегрувати з K8s. Це дозволяє ефективно запускати робочі навантаження в середовищах, керованих K8s.

Управління ресурсами ГП в K8s пов’язане з такими проблемами, як забезпечення належного планування та ефективного використання ресурсів – зокрема, K8s не здатний розподіляти ресурси одного ГП між кількома користувачами чи робочими станціями.

Одним із недоліків K8s є також його обмежений GUI, що орієнтується передусім на операції, пов’язані з масштабуванням процесу розгортання програмного коду, ініціюванням регулярних оновлення чи перезапуском окремих модулів, не надаючи при цьому інформації, специфічної до систем, що складаються з ГП, як-от розподіл навантаження чи список користувачів у черзі.

Існує стороннє програмне забезпечення, що додає GUI до K8s, що надає подібну функціональність. Одним із таких продуктів є Octant, приклад роботи якого можна побачити на рисунку 1.1.

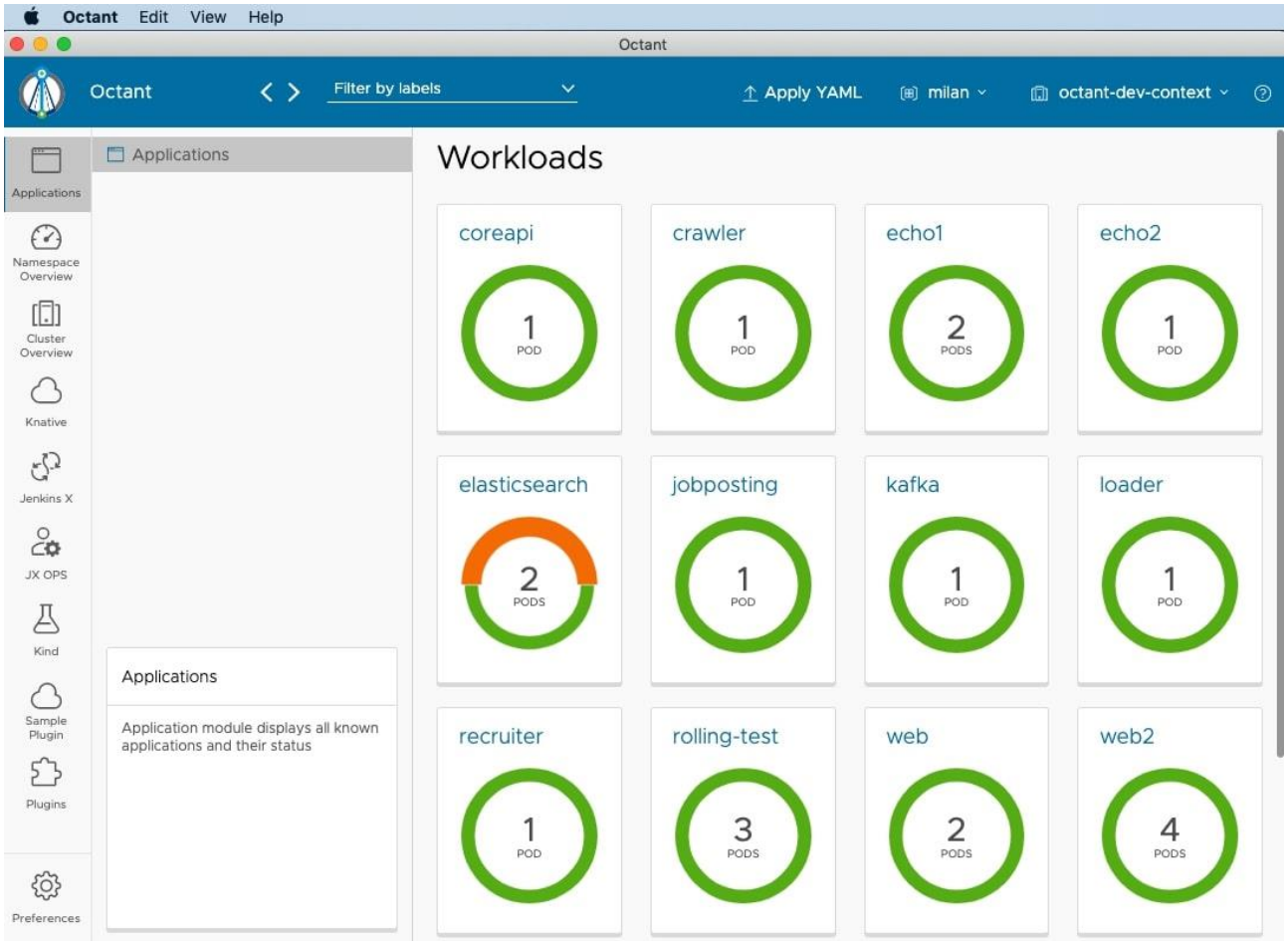


Рис. 1.1. Octant – стороння система, що додає GUI до K8s

Хоч такі рішення і здатні частково надати необхідні можливості для роботи з системами, подібними тій, що розробляється, вони не є частиною цілісної системи.

У цілому, K8s надає досить потужну платформу для керування розподіленими системами, проте містить також описані вище недоліки. Його гнучкість, масштабованість і підтримка екосистеми роблять його вагомим вибором для розподілення робочих навантажень, але його обмежені можливості відносно систем, що складаються з ГП, особливо для роботи невеликих дослідницьких команд, яким актуально розподіляти ресурси одного ГП, а також відсутність відповідного GUI демонструють, що він не є універсальним рішенням.

1.2.2. Slurm.

Slurm (Simple Linux Utility for Resource Management) — це популярний менеджер задач із відкритим кодом, який використовується у високопродуктивних обчислювальних середовищах. Він призначений для ефективного керування ресурсами та їх розподілу (таких як обчислювальні вузли, пам’ять, процесори, зокрема ГП) для виконання обчислювальних завдань, моделювання, аналізу даних у кластерах тощо.

Slurm чудово підходить для планування задач, починаючи від невеликих і закінчуючи великими за масштабом обчислювальними завданнями. Він забезпечує справедливий розподіл ресурсів між користувачами та групами, реалізуючи політики розподілу ресурсів і пріоритизації завдань. Slurm також підтримує залежності завдань і механізми випередження, гарантуючи, що завдання з вищим пріоритетом можуть переривати завдання з нижчим пріоритетом, якщо необхідно.

Slurm має функції для керування ресурсами ГП, дозволяючи користувачам вказувати вимоги до графічного процесора в описі задач і забезпечуючи планування задач на вузлах із доступними графічним процесором, аналогічно до того, як це реалізовано в K8s.

Водночас, налаштування, конфігурування та керування Slurm може бути складними, особливо для користувачів, які не знайомі з високопродуктивними обчислювальними системами або плануванням задач у цілому. Щоб оптимізувати його використання, потрібен певний рівень знань. Така складність може бути надмірною для команд, що займаються відносно невеликими обчисленнями.

Даному рішенню також бракує GUI для відповідних задач, адже робота зі Slurm відбувається за допомогою CLI. Приклад моніторингу системи у Slurm наведено на рисунку 1.2.

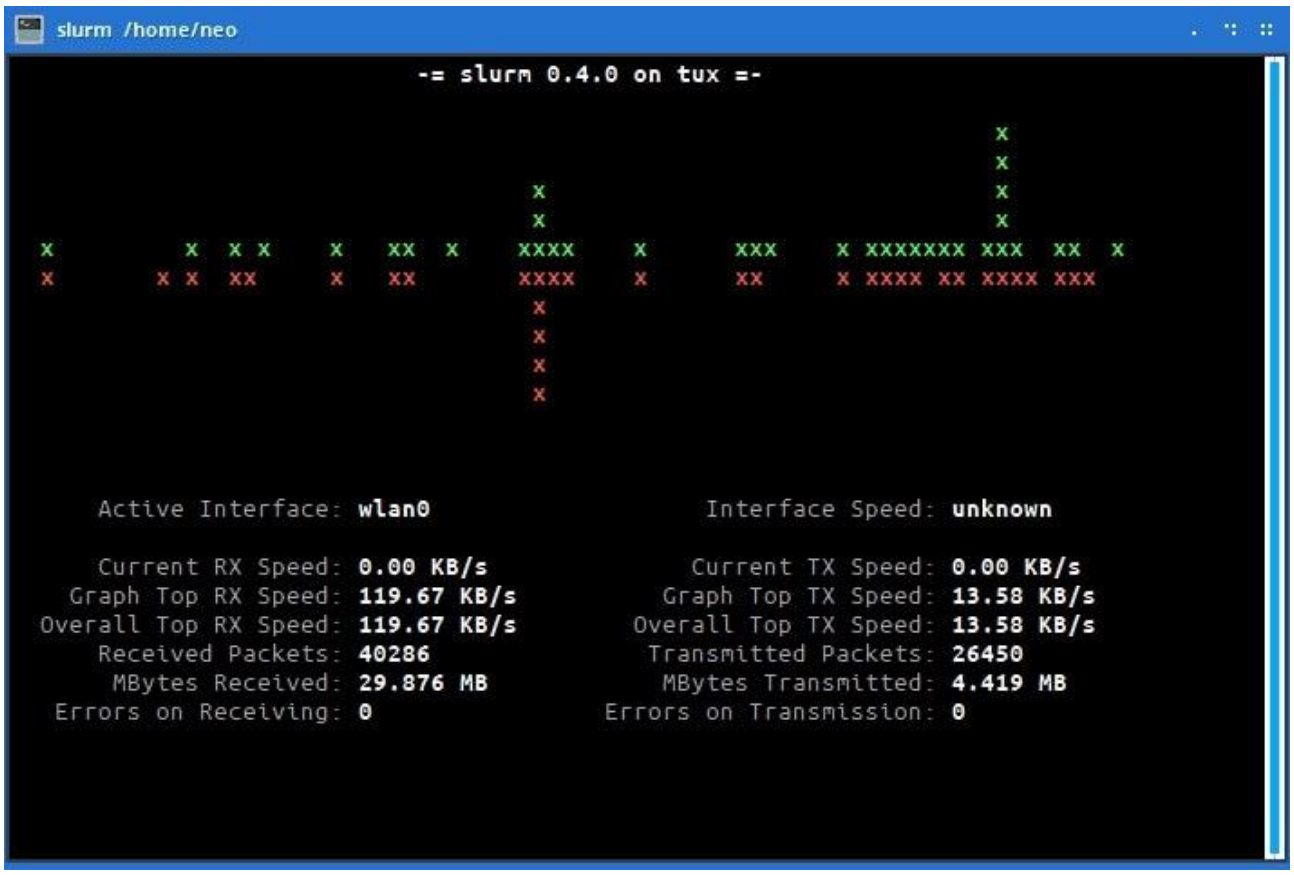


Рис. 1.2. Стан комп’ютерної мережі в Slurm

Варто також зазначити, що Slurm, на відміну від K8s, не запускається нативно на ОС, які не є Unix-подібними.

Узагальнюючи, можемо сказати, що Slurm є широко використовуваним інструментом, що дозволяє планувати задачі різного масштабу та слідкувати за ресурсами системи. Недоліки Slurm часто пов’язані зі складністю при взаємодії з ним та труднощами адаптації до вузькоспеціалізованих випадків використання, а також із відсутністю GUI. Для подолання цих обмежень може знадобитися глибоке розуміння можливостей Slurm і безперервна оптимізація на основі конкретних вимог кластера та робочого навантаження, що може бути невиправданим в невеликих командах.

1.2.3. Google Colab.

На відміну від систем, продемонстрованих вище, Google Colab представляє собою хмарне середовище для записників Jupyter (інтерактивних обчислювальних середовищ, які використовуються переважно для аналізу та

візуалізації даних, а також для статистичного моделювання, машинного навчання тощо), надане компанією Google та головним чином спрямоване на забезпечення вільного доступу до обчислювальних ресурсів, включаючи ГП.

Colab надає безкоштовний доступ до ресурсів ГП, а також такого обчислювального блоку, як тензорний процесор (ТП), дозволяючи користувачам прискорювати обчислення для завдань машинного навчання, опрацювання даних і наукових обчислень.

У ньому попередньо встановлено такі популярні бібліотеки, як TensorFlow, PyTorch, Pandas та інші, що спрощують налаштування для таких завдань.

Серед переваг Google Colab можна також назвати можливість легко ділитися даними з іншими користувачами, надаючи функції редагування та коментування в реальному часі.

Що стосується саме задач графічного процесора при запуску програмного коду на розподіленій системі, то Colab надає можливості для виконання завдань на ГП та ТП. На відміну від попередньо представлених технологій, Colab має потужний GUI, приклад якого зображено на рисунку 1.3.

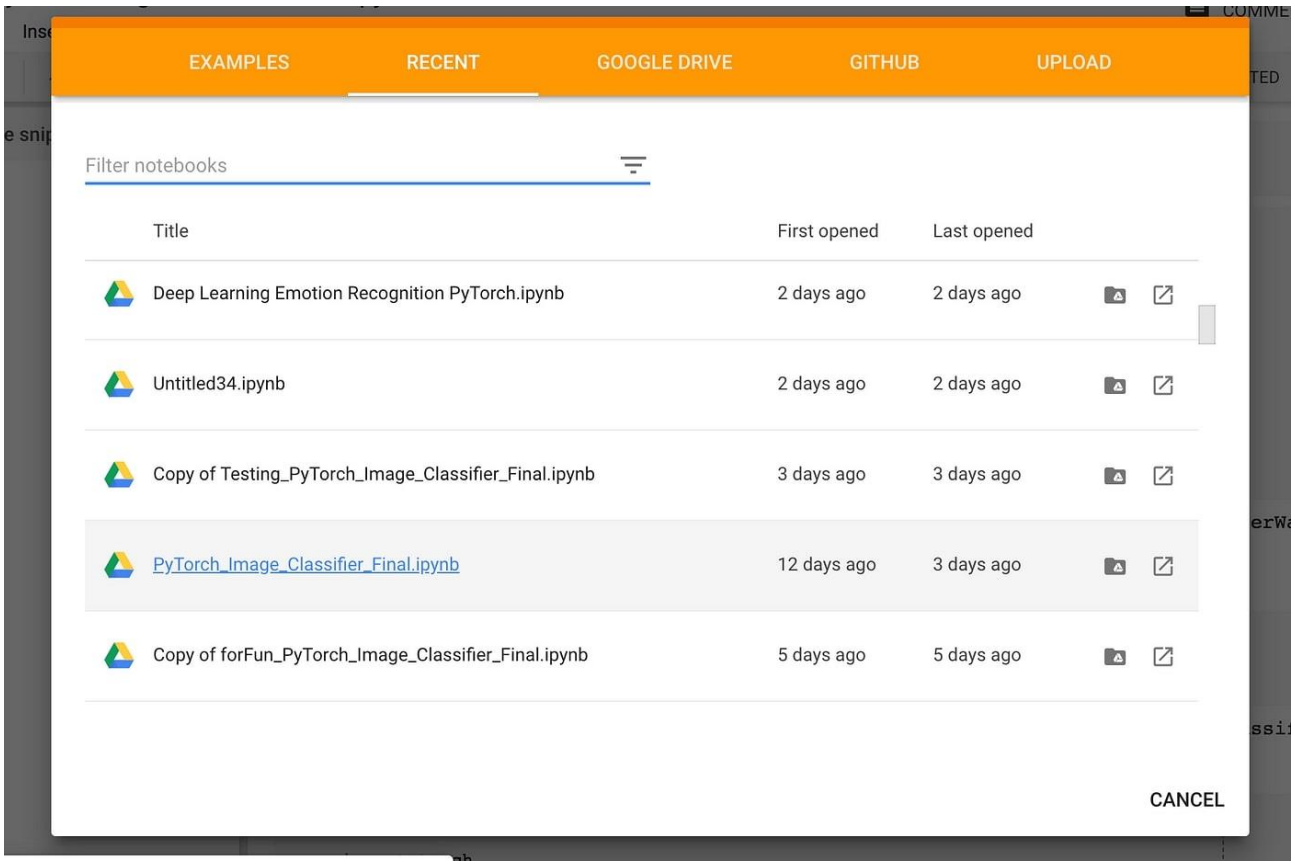


Рис. 1.3. Перелік проєктів у GUI від Google Colab

Його GUI також охоплює можливості вибору ГП та моніторингу доступних ресурсів. Приклад такої функціональності продемонстровано на рисунку 1.4.

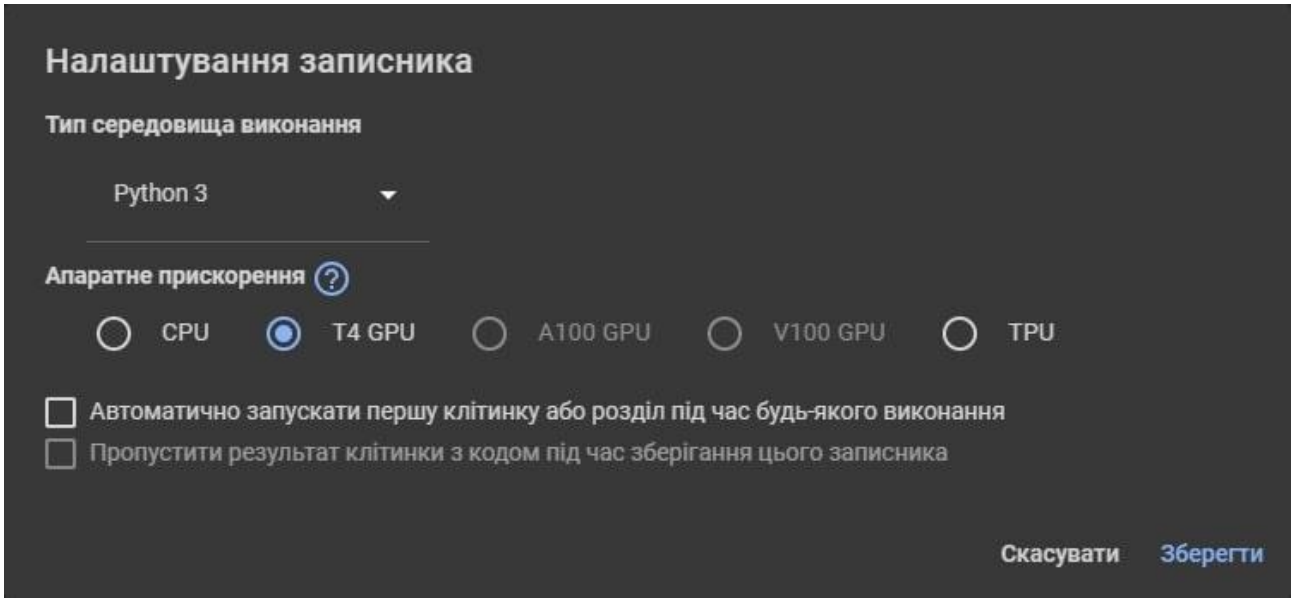


Рис. 1.4. Можливість вибору ГП в GUI від Google Colab для запуску програмного коду

Разом з тим, Colab не надає можливості менеджменту задач [15] та їх планування, як це зроблено у випадку з K8s чи Slurm. Також він не дозволяє запускати задачі на власних процесорах команди дослідників, що робить його можливості недостатніми в порівнянні з можливостями інших подібних систем, зокрема потенційно тієї, що розробляється. Крім того, тривалість сесії в Colab обмежується 48 годинами, після чого експеримент примусово завершується.

1.3. Порівняння існуючих технологій

Дослідивши документації вищезгаданих технологій та використавши кожну з них на практиці, можемо зробити висновок про наявність переваг кожної з них.

Повний перелік переваг зазначених раніше технологій можна побачити в таблиці 1.1.

Таблиця 1.1

Порівняння технологій

Можливість технології	**K8s**	**Slurm**	**Google Colab**
Менеджер задач і планування	+	+	-
Запуск на ГП	+	+	+
Масштабованість	+	+	-
Призначення ресурсів	+	+	-
Управління кластером	+	+	-
Можливість розподіляти ресурси одного ГП	-	-	-
Наявність колаборації	-	-	+
Безкоштовний доступ до ГП	-	-	+
Легкість використання	+	-	+
Наявність GUI	-	-	+
Конфігурування	+	+	-

Як можна бачити з таблиці 1.1, кожна із зазначених технологій має як переваги, так і недоліки. Серед недоліків є ключові обмеження, які спонукають до розробки окремої системи, адже кожна з наведених технологій частково не покриває всіх необхідних можливостей.

1.4 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення, зазначені нижче, представляють собою перелік властивостей, які повинна мати система, для того, щоб задовольнити потреби користувачів даної системи.

Програмне забезпечення, що розробляється, повинно відповідати всім критеріям, зазначеним у таблиці 1.1. Основним його призначенням є надання можливості управління задачами та їх планування.

Ключовою вимогою також є можливість запуску програмного коду на розподіленій системі, що працює на ресурсах ГП. Крім того, система повинна бути масштабованою для того, щоб відповідати змінам при використанні кількома користувачами та збільшенні робочого навантаження.

Система має надавати ресурси користувачам згідно з системою пріоритетів задач для того, щоб забезпечити справедливий розподіл ресурсів. Також має бути можливість гнучкого керування ресурсами: розподіл не тільки ресурсів певного кластеру, а й окремого ГП.

Варто зазначити, що оскільки система, яка розробляється, має дозволяти розподіл ресурсів ГП між декількома користувачами, які використовують власні ГП-и, то наявність колаборації та можливість безкоштовного доступу до ГП надані за замовчанням.

Програмне забезпечення повинно мати інтуїтивно зрозумілий GUI для забезпечення легкості використання, а також можливість налаштовувати систему під потреби, що змінюються.

1.5. Постановка задачі створення системи менеджменту обчислювальних ресурсів

Метою даної системи є вирішення основної проблеми при запуску коду та його плануванні на декількох графічних процесорах, оскільки обчислення в межах подібних систем є критично важливим аспектом сучасних обчислювальних досліджень, що дозволяє опрацьовувати великі та складні набори даних. Для досягнення мети роботи було поставлено такі завдання:

- Провести огляд існуючих технологій, що підтримують можливість керування задачами графічного процесора для запуску коду на розподіленій системі
- Сформулювати вимоги до системи, що має бути розроблена
- Розглянути засоби та підходи до розробки системи
- Створити архітектуру системи відповідно до заданих вимог
- Розробити відповідну функціональність системи з урахуванням розглянутих засобів розробки, а також GUI для легкого використання системи
- Провести тестування розробленої системи

Висновок до розділу 1

Після проведення аналізу існуючих технологій, що надають можливості менеджера задач графічного процесора для запуску програмного коду на розподіленій системі, було виявлено, що всі розглянуті технології певною мірою не задовольняють потребам потенційних користувачів системи, тому була обґрунтована необхідність у створенні системи, яка б відповідала зазначеним вимогам.

Такі можливості даної системи, як розподіл ресурсів одного графічного процесора, наявність зрозумілого GUI та конфігурування системи, а також ефективний розподіл обчислювальних потужностей є ключовими показниками, що відрізнятимуть потенційну систему від існуючих аналогів.

У наступному розділі буде проведено аналіз структури системи, розглянуто її архітектуру та функціональність, а також показано реалізацію можливостей взаємодії користувача з нею.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз структури програмного забезпечення

Для того, щоб зрозуміти необхідну функціональність системи, що розробляється, є сенс розібрати кожну частину теми.

Як уже було зазначено, система перш за все повинна бути багатокористувацькою, тобто розрахована на багатьох користувачів (експериментаторів) або груп користувачів.

Також система має слугувати як менеджер задач, тобто розподіляти задачі за принципом справедливості, розподіляючи наявні обчислювальні ресурси так, щоб не залишати окремих користувачів без необхідних ресурсів, але при цьому й не надаючи їм надлишкових потужностей.

Наявність терміну «графічний процесор» в темі роботи означає, що система передусім розрахована на ГП, оскільки більшість експериментів, які містять велику кількість даних або які вимагають значних обчислювальних ресурсів, використовують саме ГП [16]. Система також повинна мати підтримку обчислень на ТП, а за відсутності даних типів процесорів вона має підтримувати обчислення на ЦП.

Як випливає з назви теми, дана система, зокрема, дозволятиме запускати програмний код, тобто конкретні експерименти на тих потужностях, що були виділені для конкретного користувача або групи користувачів, що здійснюють експеримент.

Виконання коду саме на розподіленій системі означає, що така система повинна надавати можливість працювати з ресурсами багатьох робочих станцій, а отже, система потребує тестування на декількох комп’ютерах.

Для кращої формалізації розв’язання проблеми створення менеджера задач є доцільним продемонструвати його роботу на прикладі UML-діаграм, адже вони

надають чітке візуальне уявлення про структуру та поведінку системи, кожна з функцій якої представляє конкретну відповідальність або можливість, яку система повинна підтримувати [17]. Діаграму прецедентів можна побачити на рисунку 2.1.

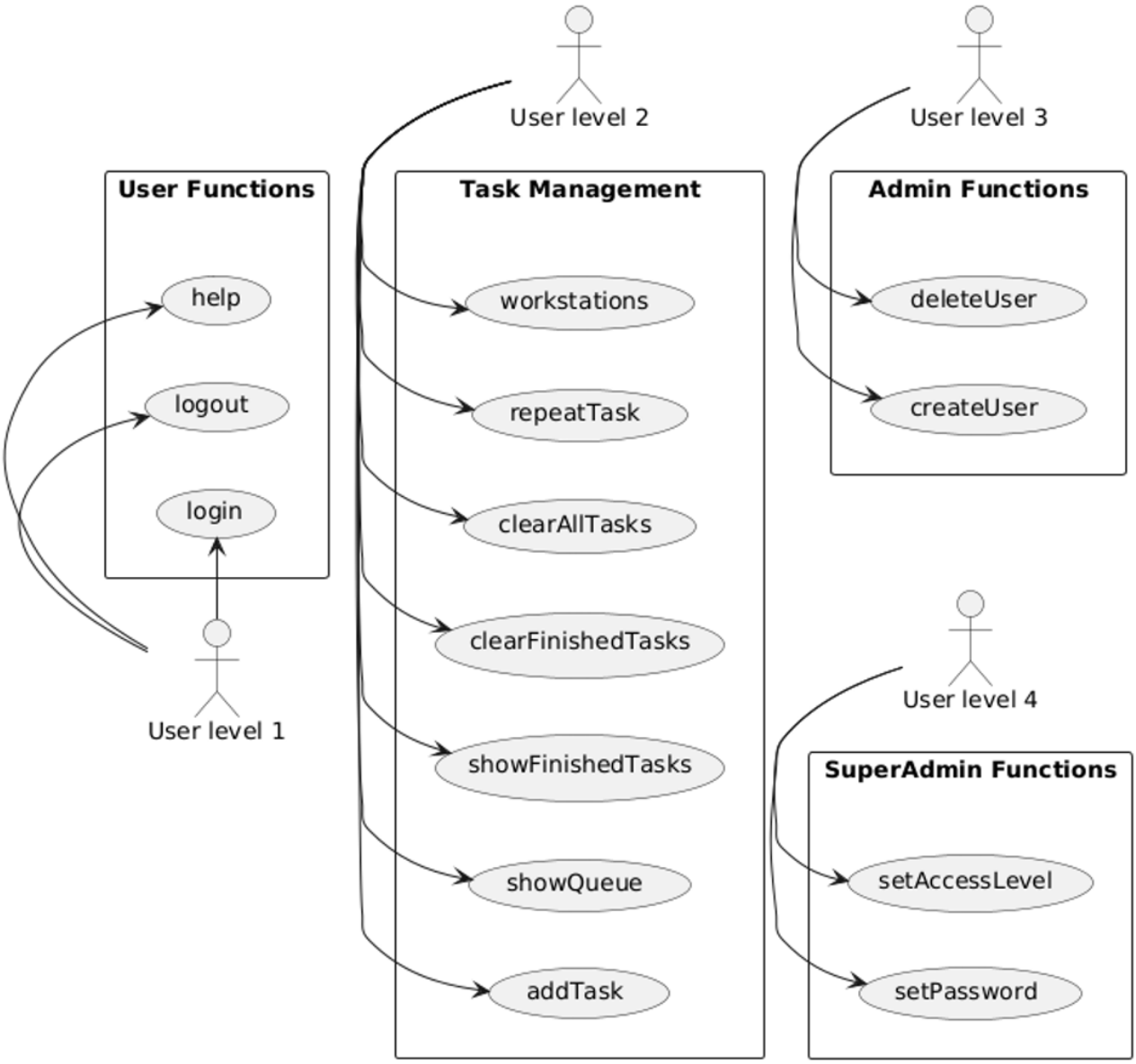


Рис. 2.1. Діаграма прецедентів.

Дана діаграма описує взаємодію користувачів (акторів) із системою, тобто те, як вони використовуватимуть систему для вирішення своїх задач. Для опису системи також створено діаграму класів, яка зображена на рисунку 2.2.

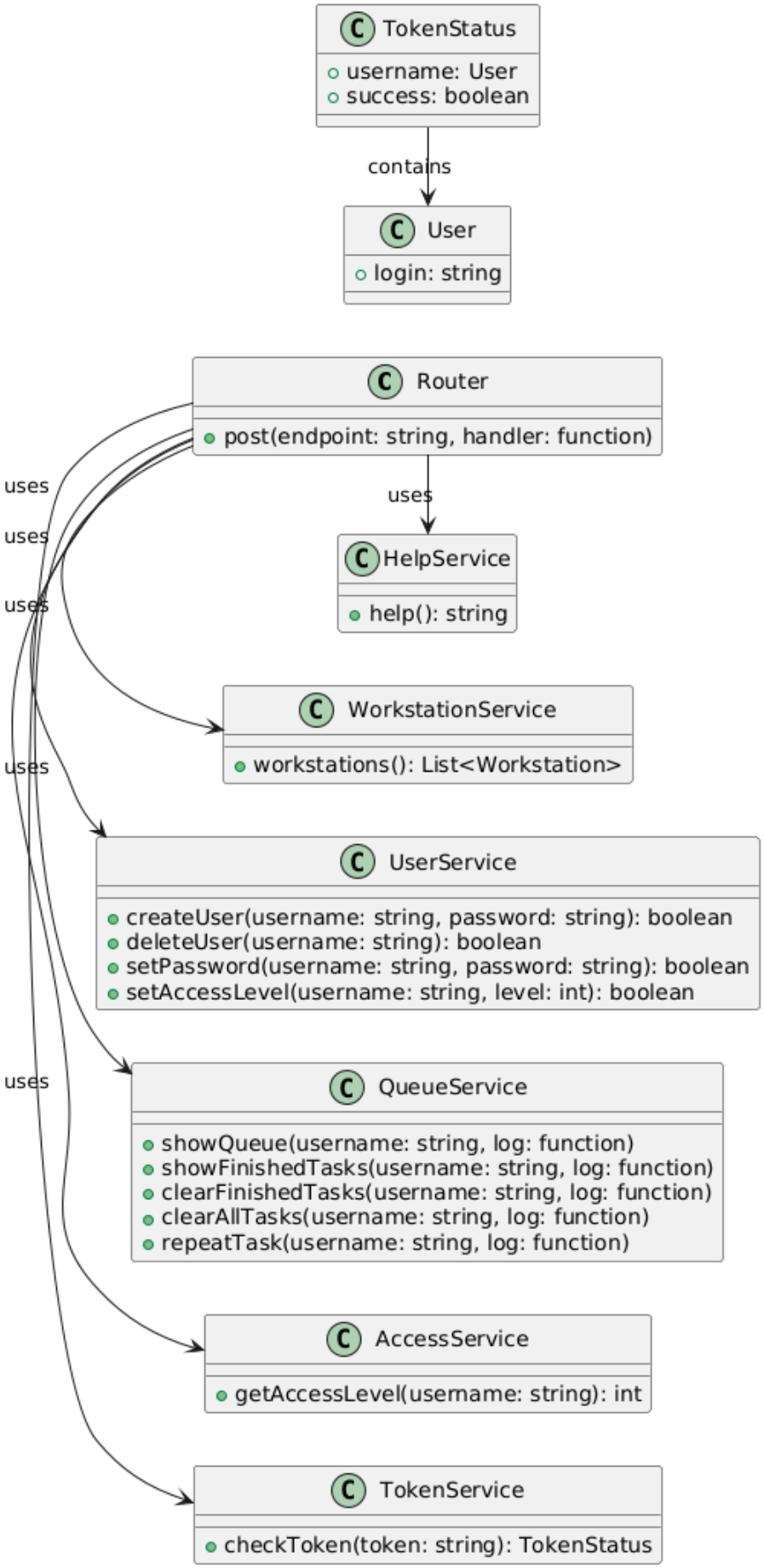


Рис. 2.2. Діаграма класів.

Діаграма класів використовується для моделювання структури системи. Вона описує класи, їхні атрибути, методи та взаємозв’язки між класами в системі. Ця діаграма допомагає зрозуміти внутрішню архітектуру програмного забезпечення.

У процесі розробки також було створено діаграму послідовності для авторизації користувача з подальшим управлінням задачами та виходом із системи після завершення роботи. Ця діаграма використовується для моделювання взаємодії об'єктів у системі з плином часу. Вона показує, які об'єкти взаємодіють між собою, в якому порядку передаються повідомлення та як відбувається виконання сценарію. Діаграму можна побачити на рисунку 2.3.

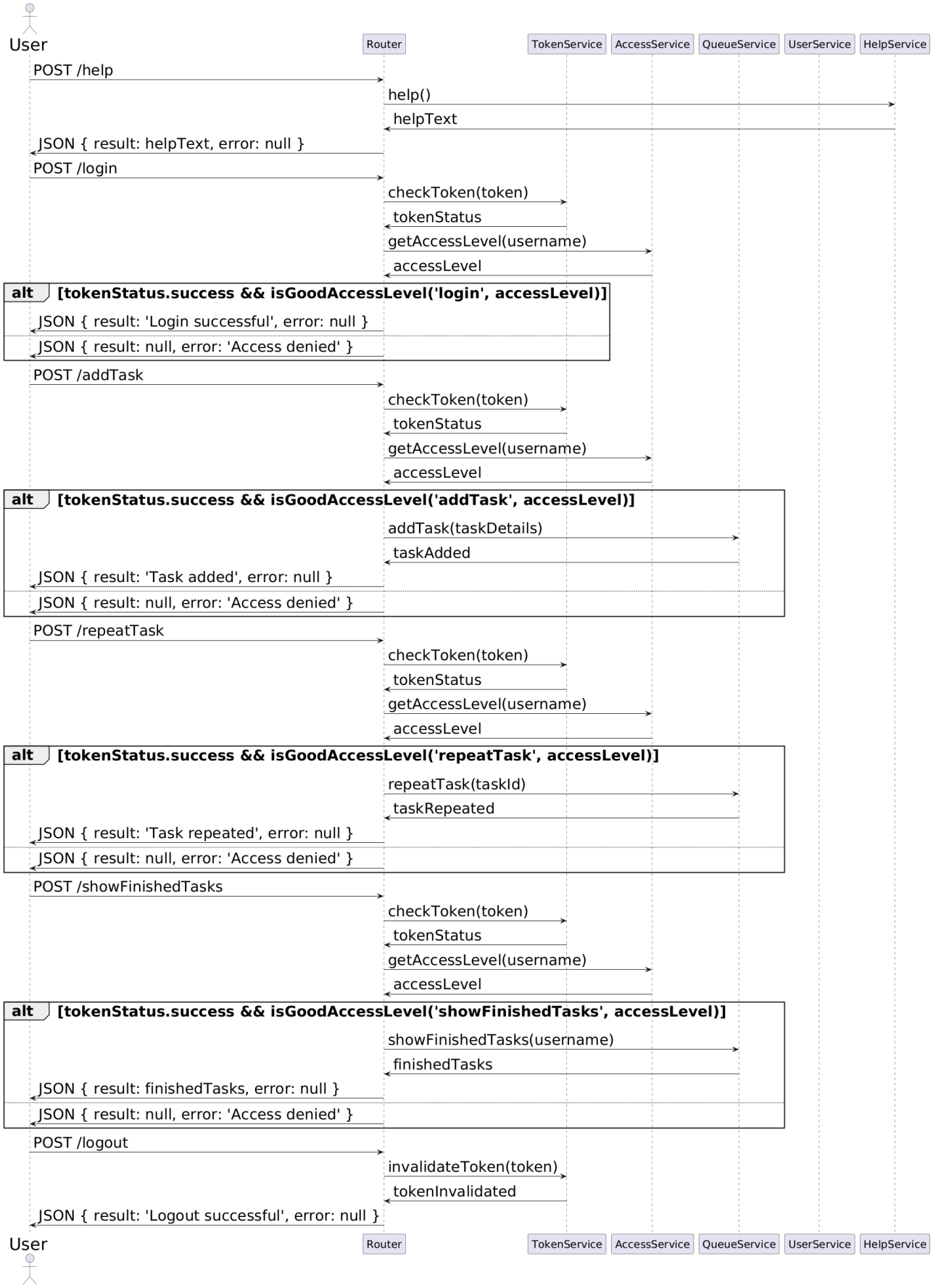


Рис. 2.3. Діаграма послідовності «авторизація-управління-деавторизація».

2.2. Технічна реалізація програмного забезпечення

2.2.1. Засоби розробки.

Під час розробки програмного забезпечення є важливим вдало визначити набір засобів, за допомогою яких ця розробка вестиметься. Серед них важливими є: мова програмування, компілятор (або інтерпретатор), платформа запуску коду, сервер. Також можливе використання рушія, сторонніх фреймворків чи бібліотек.

Для створення системи обрано платформу для розробки клієнт-серверного програмного забезпечення під назвою Node.js [18], яка є рушієм, побудованим на базі іншого рушія – V8, що вбудований у більшість сучасних веб-бравзерів [19]. Node.js здатен виконувати код, написаний мовою JavaScript, яка є досить універсальною і може виконуватися на всіх сучасних обчислювальних машинах за наявності інтерпретатора, який уже вбудований в цю платформу [20].

Серед інших переваг, які має дана платформа, можна назвати асинхронність, адже Node.js базується на асинхронній моделі програмування, що дозволяє опрацьовувати велику кількість одночасних з'єднань без блокувань. Це є важливим аспектом для багатокористувацьких систем, де кількість запитів до сервера може бути значною, через що виникає необхідність ефективніше їх опрацьовувати.

Також дана платформа містить у собі багато модулів і бібліотек, доступних через вбудований менеджер пакетів NPM із вільним їх розповсюдженням. Це дозволяє швидко реалізовувати необхідну функціональність та використовувати готові рішення для роботи з ГП, управлінням чергами задач і веб-інтерфейсом.

Node.js є відмінним вибором для систем, де важлива взаємодія в реальному часі. Наприклад, для менеджера задач, який потребує оновлень статусу виконання задач, можна використовувати веб-сокети для забезпечення миттєвого обміну даними між клієнтом і сервером.

Крім того, цей засіб розробки дозволяє легко інтегрувати серверну частину з сучасними фронтенд-фреймворками (наприклад, React, Vue, Angular). Це

дозволяє створити динамічний та інтерактивний інтерфейс для користувачів, що підвищує зручність використання системи.

Node.js також важливий для розроблюваної системи тим, що працює на різних операційних системах, що покликано спростити використання системи, адже в такому випадку нею можна користуватися за допомогою широкого спектру пристроїв.

Приклад запуску програмного коду на Node.js можна побачити на рис. 2.4.

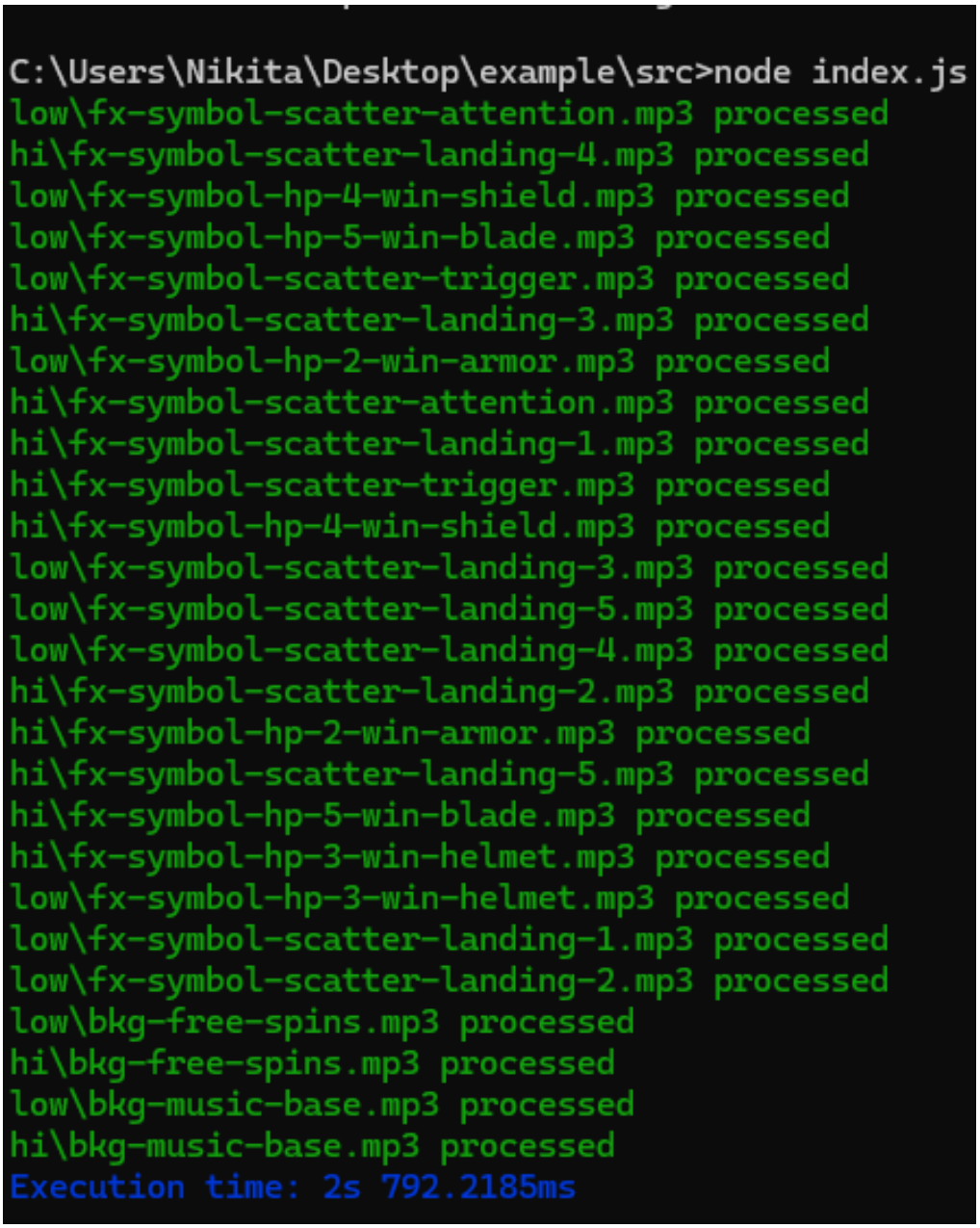


Рис. 2.4. Приклад запуску коду на Node.js.

Даний приклад демонструє простоту запуску програми, що здійснює опрацювання звукових файлів за допомогою вбудованої бібліотеки.

Також для розробки було обрано технологію Windows Subsystem for Linux (WSL) [21], яка слугує прошарком сумісності для запуску виконуваних файлів операційної системи Linux в середовищі системи Windows. Це дозволяє взаємодіяти з системою за допомогою двох операційних систем, не змінюючи обчислювальну машину. При розробці даної системи WSL є особливо корисною, бо дозволяє використовувати програмні утиліти й бібліотеки, розраховані на ОС Linux, маючи при цьому на робочій станції ОС Windows.

Використання WSL у даному випадку є також зручним, оскільки для створення системи використовується таке середовище розробки, як Visual Studio Code, що має вбудовану підтримку WSL, надаючи можливість легко взаємодіяти з проєктом через даний прошарок сумісності. На рисунку 2.5 можна побачити приклад запуску програми, що надає користувачеві стислу інструкцію щодо взаємодії з системою. Це відбувається за допомогою WSL через вбудований у середовище розробки термінал. Можна побачити, що для розроблюваного проєкту, як і для всіх проєктів у межах WSL, створюється окрема директорія на логічному диску, що дозволяє нам виокремити директорії з проєктами, з яких складається система, від інших проєктів на обчислювальній машині з операційною системою Windows, взаємодіючи з потрібними проєктами так, ніби ми використовуємо Linux.

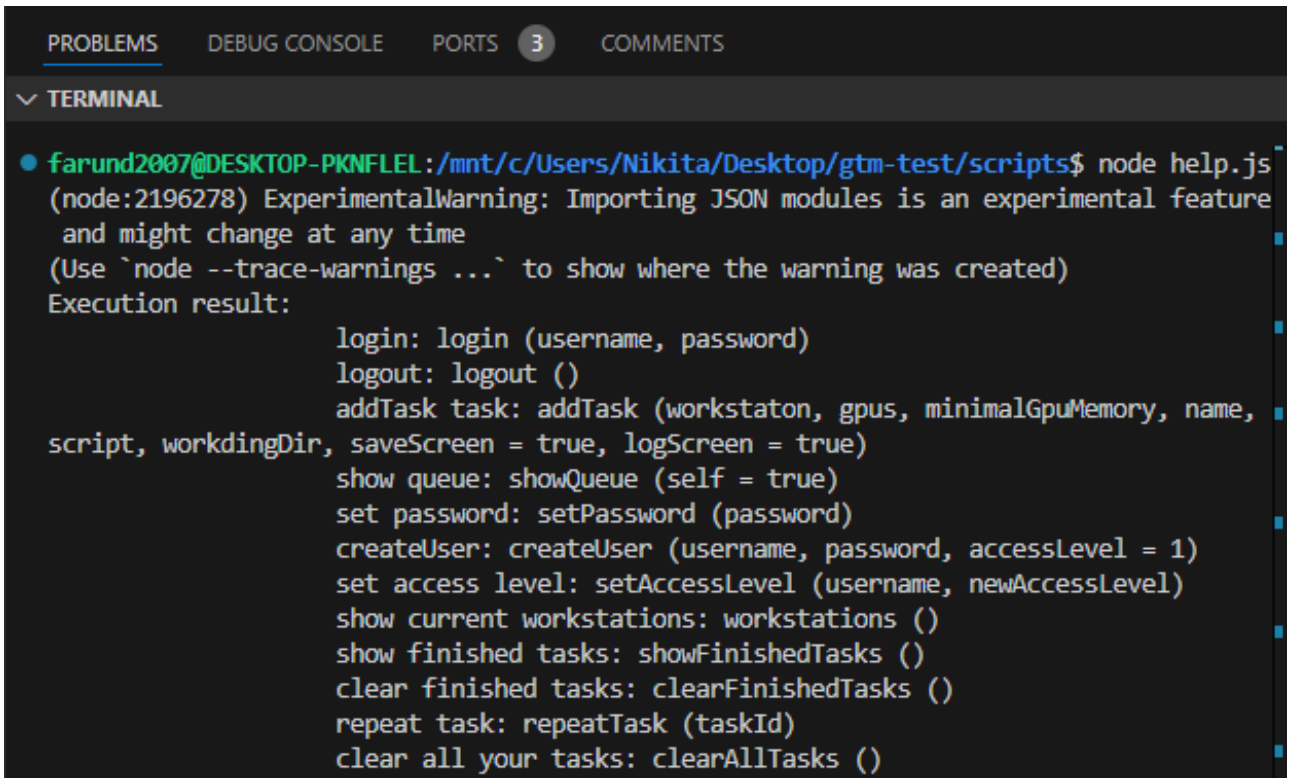


Рис. 2.5. Запуск програми в межах WSL.

Для розробки веб-інтерфейсу даної системи з метою покращення клієнт-серверної взаємодії використовується фронтенд-бібліотека React [22].

React було вибрано передусім тому, що він легко інтегрується з бекендом через REST API [23] або WebSocket [24], що дозволяє швидко взаємодіяти з сервером і отримувати оновлення статусу задач у режимі реального часу. Це критично важливо для менеджера задач, який постійно працює з даними, що оновлюються.

Також для систем з великою кількістю взаємодій і станів React має інструменти для централізованого управління станом, такі як Redux [25] або Context API. Це дозволяє легко керувати станом застосунку, що корисно для моніторингу задач, відстеження прогресу і координації дій різних користувачів, адже кожен користувач (клієнт) має свій веб-інтерфейс.

Крім того, React полегшує створення складних інтерактивних елементів, таких як графіки, панелі моніторингу та віджети для керування задачами. Це є корисним в межах даної системи, адже це дозволяє створити інтуїтивно зрозумілий інтерфейс, де користувачі можуть легко керувати задачами, переглядати статистику ГП та отримувати аналітику в реальному часі.

Не менш важливим є те, що налаштування та запуск базового проєкту за допомогою React на практиці є простим, що особливо корисно при декількох тестових проєктах з експериментами. Приклад запуску проєкту на React продемонстровано на рисунку 2.6.

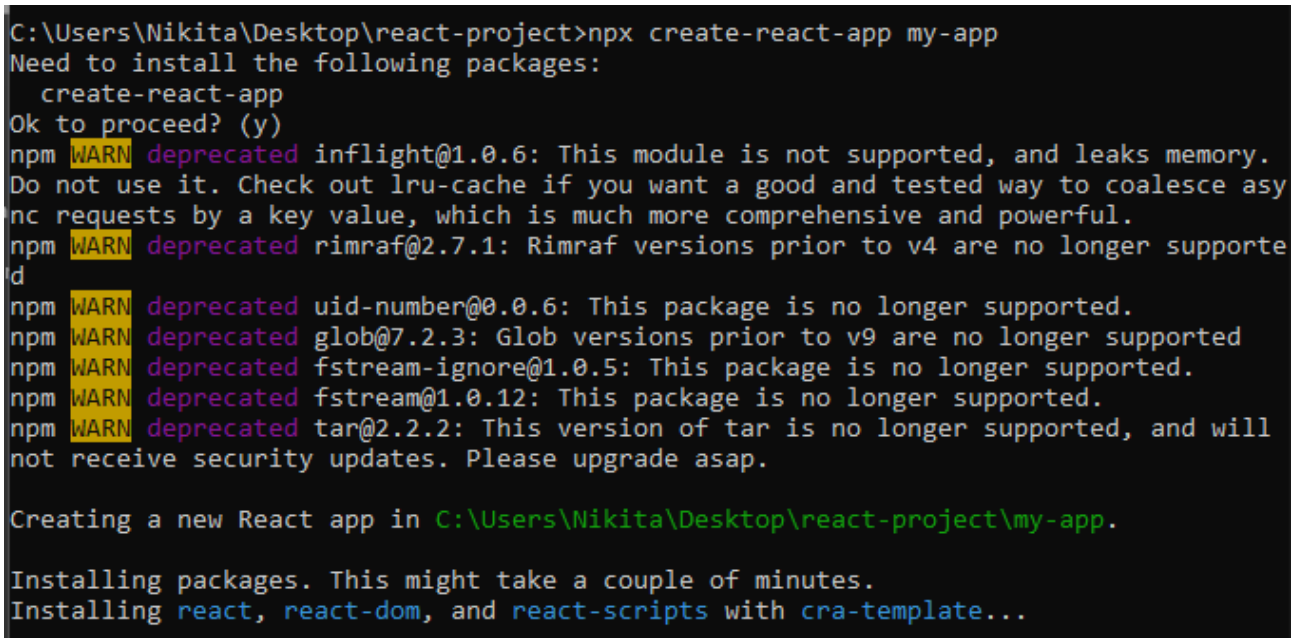


Рис. 2.6. Запуск проєкту за на базі React.

2.2.2. Реалізація клієнт-серверної взаємодії.

Клієнт-серверна взаємодія в багатокористувацькій системі управління задачами на ГП є основою цієї системи й повинна забезпечувати ефективну, безпечну та надійну комунікацію між користувачами (клієнтами) та сервером, який керує ресурсами ГП і виконанням задач. У межах даної системи така взаємодія містить реалізацію авторизації, присвоєння ролей користувачам, програмного інтерфейсу REST API, моніторингу ресурсів, опрацювання помилок та логування, що буде розглянуто нижче. Також система в перспективі зможе підтримувати шифрування з’єднання. Загальний принцип роботи клієнт-серверної взаємодії візуально описаний на рисунку 2.7.

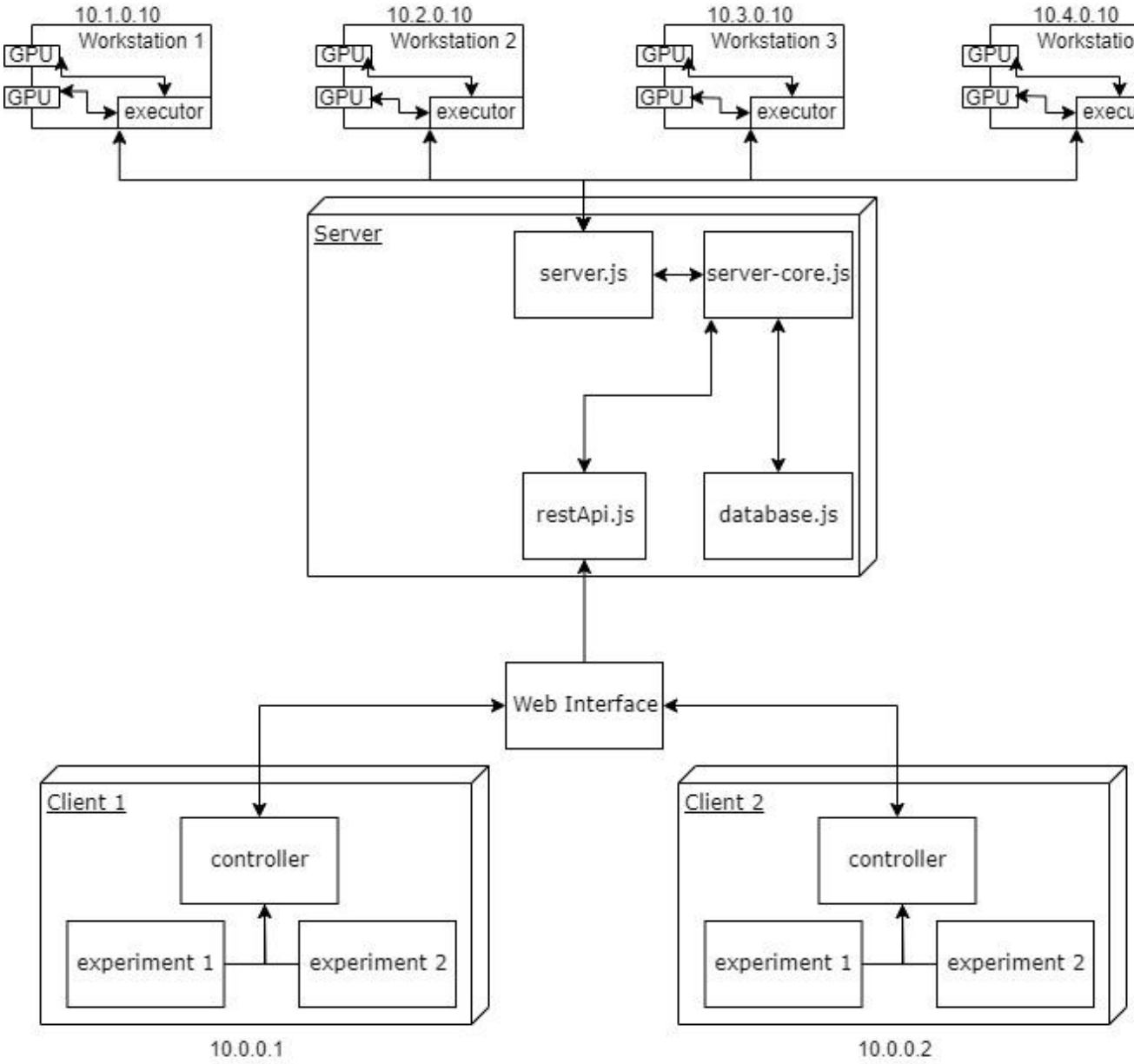


Рис. 2.7. Принцип роботи клієнт-серверної взаємодії.

Як можна побачити з рисунка, система в даному випадку має кілька робочих станцій, кожна з яких має свого виконавця (executor), що здійснює зв'язок між ГП та сервером. Крім того, у системі є декілька клієнтів, кожен з яких, своєю чергою, має декілька експериментів для виконання. Оскільки обчислення для кожного експерименту розподіляються сервером за робочими станціями, то система припускає, що для кожного експерименту вже існує унікальний алгоритм його розбиття на прості обчислення, які і будуть виконуватися на різних робочих станціях. На рисунку вище можна побачити, що

клієнти взаємодіють із сервером через веб-інтерфейс. Приклад самого веб-інтерфейсу можна побачити на рисунку 2.8.

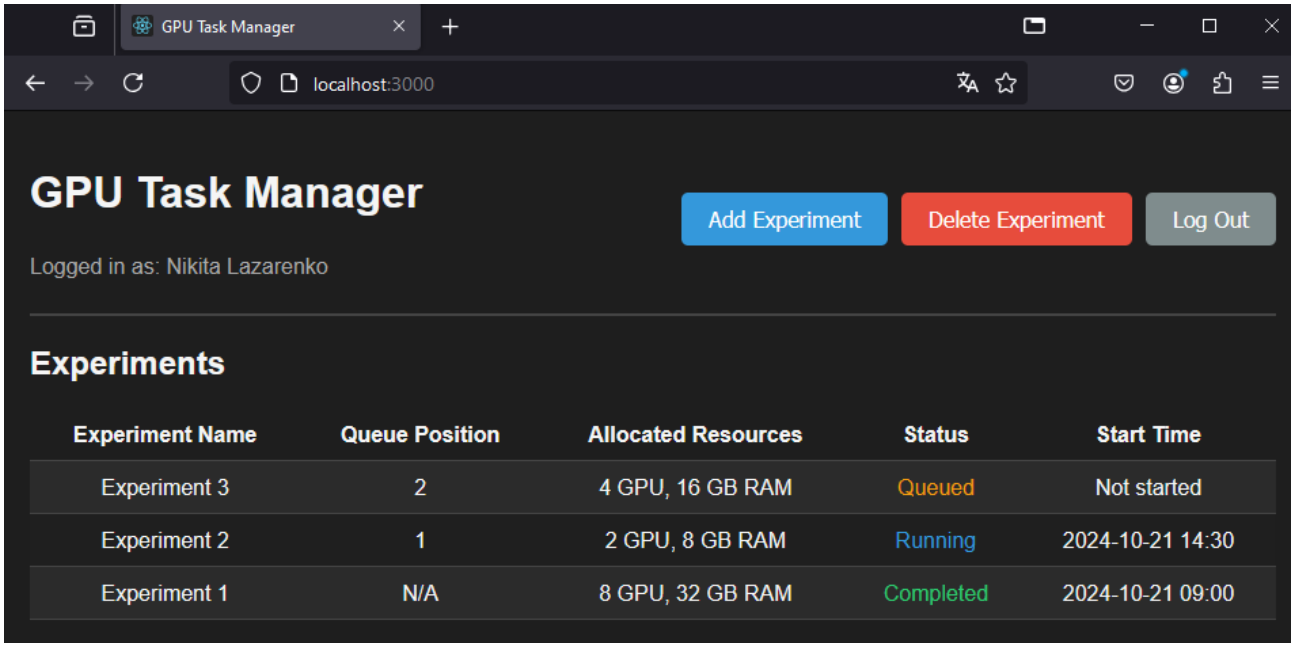


Рис. 2.8. Взаємодія з сервером через веб-інтерфейс.

Як видно з рисунка, веб-інтерфейс є мінімалістичним, але при цьому зрозумілим і містить усе необхідне для взаємодії користувача з системою. Він відображає основні дані, серед яких: ім’я поточного користувача, список експериментів, для кожного з яких зазначено назву, номер у черзі, виділені ресурси, статус і час початку виконання. Також веб-інтерфейс дозволяє цьому конкретному користувачеві додати або видалити експеримент і вийти з облікового запису.

У додаток до зручного веб-інтерфейсу, система також підтримує консольний інтерфейс, який широко використовувався на ранніх стадіях розробки й дотепер надає користувачу більш широкі можливості. На прикладі консольного інтерфейсу розглянемо зазначені в таблиці 2.1 основні функції системи.

Роботу обох інтерфейсів також буде детальніше продемонстровано нижче.

Таблиця 2.1

Функції системи та їхній опис

Функція	**Опис**
login	Дозволяє користувачеві авторизуватися
addTask	Надає можливість додати задачу
showQueue	Показує чергу задач
showFinishedTasks	Показує чергу завершених задач
clearFinishedTasks	Очищує список завершених задач
clearAllTasks	Очищує список усіх задач
repeatTask	Повторює вже додану до цього задачу
setPassword	Встановлює пароль для себе
setPassword(targetUser)	Встановлює пароль для цільового користувача
setAccessLevel	Встановлює рівень доступу для користувача
createUser	Дозволяє створити нового користувача
deleteUser	Дозволяє видалити користувача за іменем
workStations	Виводить список робочих станцій
help	Показує коротку інструкцію
logout	Деавторизація

Усі функції, показані в даній таблиці, доступні для привілейованого користувача. При цьому система не розділяє користувачів за типами, а лише присвоює їм рівень доступу від 1 до 4, у залежності від якого користувачі можуть виконувати ті чи інші функції. Наприклад, користувач із низьким рівнем доступу не може змінювати рівень доступу.

Для визначення рівня доступу користувача в системі існує відповідна функціональність. Мінімальний рівень доступу, необхідний для виконання кожної з функцій, можна побачити в таблиці 2.2.

Таблиця 2.2

Необхідний рівень доступу для виконання функцій

Функція	**Рівень доступу**
login	1
addTask	2
showQueue	2
showFinishedTasks	2
clearFinishedTasks	2
clearAllTasks	2
repeatTask	2
setPassword	1
setPassword(targetUser)	4
setAccessLevel	4
createUser	3
deleteUser	3
workstations	2
help	1
logout	1

Крім того, реалізовано можливість авторизації та перевірки, чи є користувач авторизованим. Алгоритм самої авторизації можемо побачити на рисунку 2.9.

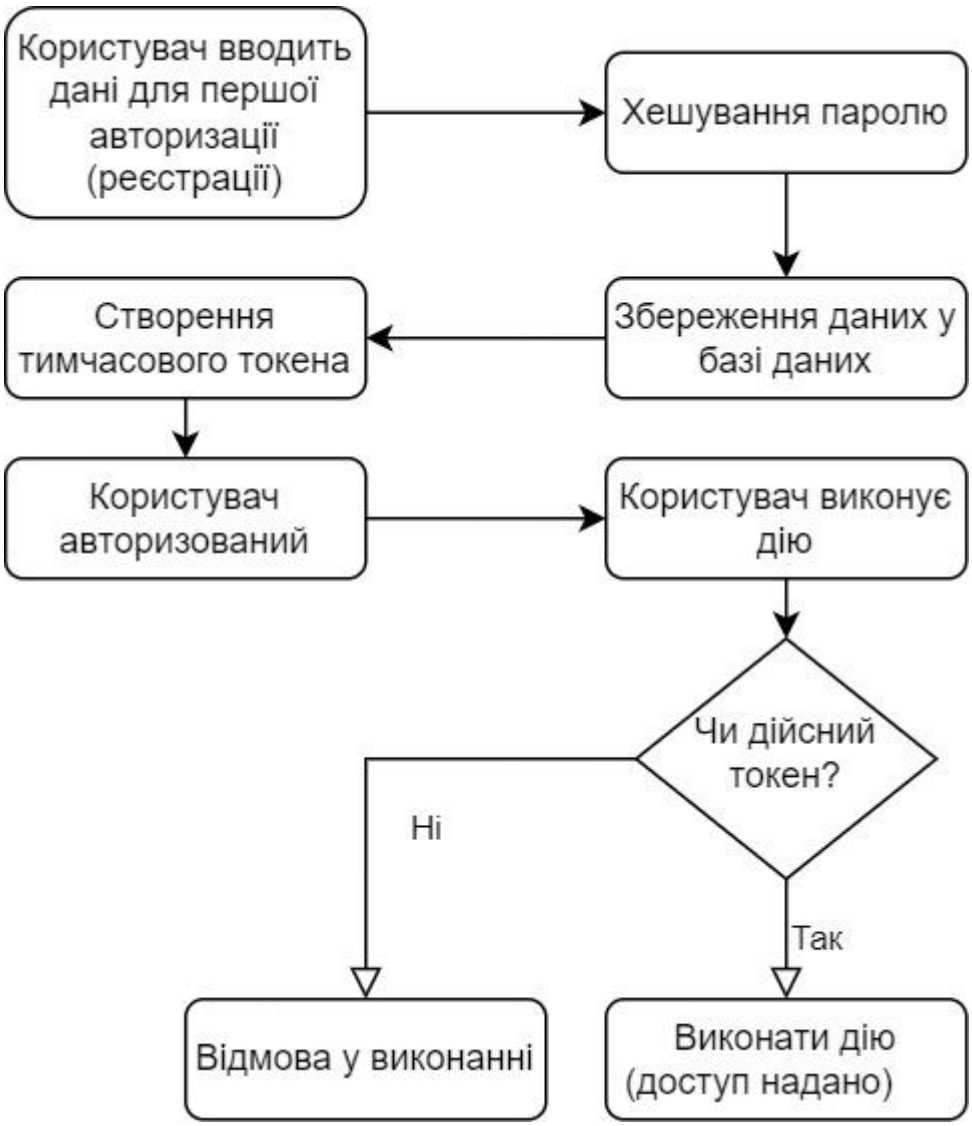


Рис. 2.9. Алгоритм авторизації користувача.

На рисунку продемонстровано процес першої авторизації користувача, коли він зазначає логін і пароль, після чого ці дані зберігаються в БД. Пароль при цьому зберігається в хешованому вигляді, щоб при наступних авторизаціях даного користувача порівнювати його пароль не з паролем, що зберігається в БД, а з його хешем. Це зроблено для підвищення рівня безпеки даних користувача, адже в такому випадку паролі не зберігаються в БД в чистому вигляді, а замінюються на послідовність символів, яка для стороннього спостерігача не становить жодної цінності [26]. Хешування в системі реалізовано за допомогою бібліотеки bcrypt [27], яку включено в Node.js і яка здатна додавати до хешу «сіль», роблячи його різним кожного разу, але при цьому маючи змогу відрізняти хеши різних паролів.

Варто зазначити також про доступ з боку користувача до функцій системи, що реалізовано за допомогою REST API, який є загальним підходом для взаємодії компонентів у розподіленій системі, у нашому випадку – взаємодії клієнта й сервера.

Під час розробки даної системи не використовувалася конкретна мова запитів чи бібліотека для реалізації REST API. Натомість було створено власну реалізацію, що містить ендпоїнти (кінцеві точки), які представляють собою звернення до маршруту у вигляді окремого запиту із зазначенням IP-адреси сервера. Реалізація REST API в даній системі показана на рисунку 2.10.

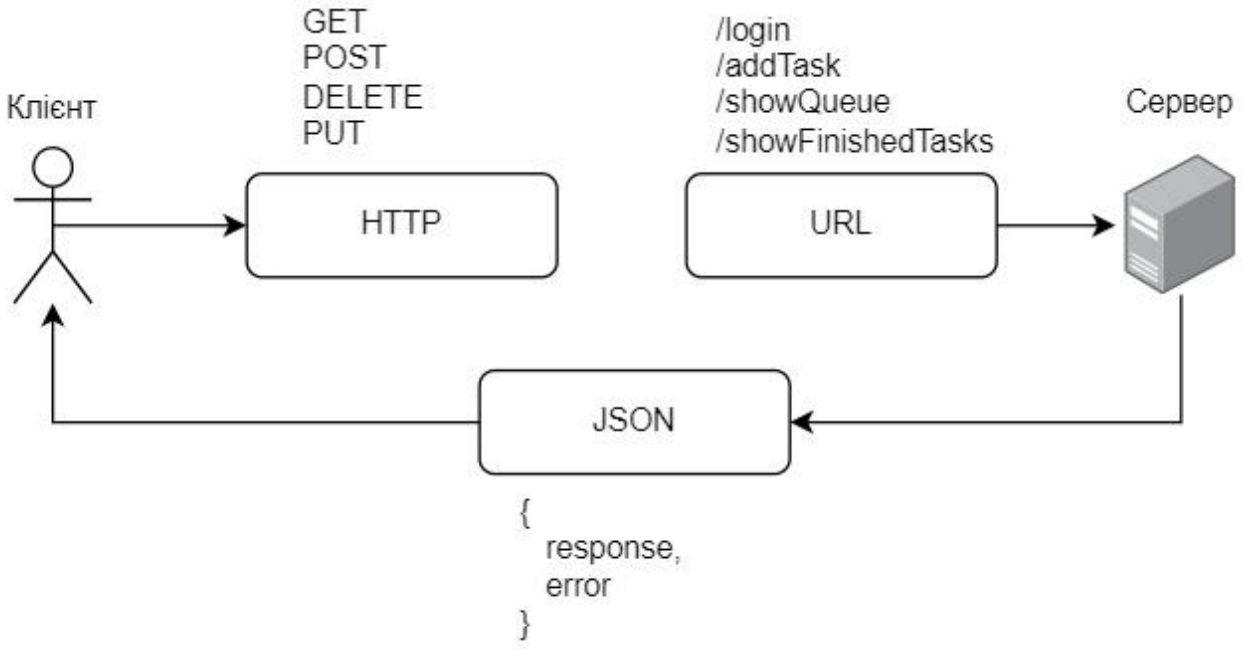


Рис. 2.10. Реалізація REST API з HTTP, URL та JSON.

У даній розробці також реалізовано логування, тобто процес запису важливих дій у системі в спеціальний журнал (лог, або скринлог). Під час запиту ресурсів створюється окрема консоль, у якій виконується новостворений процес для опрацювання задачі. Ця консоль може припинити своє існування після завершення процесу, або ж користувач може перейти в дану консоль і отримати скринлог», який у даному випадку є відбитком виконання задачі. Якщо консоль знищується, система записує скринлог в файл після кожної опрацьованої задачі. Приклад запису скринлогів показано на рисунку 2.11.

```
farund2007@DESKTOP-PKNFLEL:~$ ls
ai                                screenlog.46.gtm_test_train_10_1      screenlog.69.gtm_test_train_51_1
screenlog.17.gtm_test_train_10_1  screenlog.47.gtm_test_train_10_1      screenlog.70.gtm_test_train_51_1
screenlog.18.gtm_test_train_10_1  screenlog.48.gtm_test_train_10_1      screenlog.71.gtm_test_train_51_1
screenlog.19.gtm_test_train_10_1  screenlog.49.gtm_test_train_10_1      screenlog.72.gtm_test_train_51_1
screenlog.26.gtm_test_train_10_1  screenlog.50.gtm_test_train_10_1      screenlog.73.gtm_test_train_51_1
screenlog.27.gtm_test_train_10_1  screenlog.51.gtm_test_train_10_1      screenlog.74.gtm_test_train_51_1
screenlog.28.gtm_test_train_10_1  screenlog.52.gtm_test_train_10_1      screenlog.75.gtm_test_train_51_1
screenlog.29.gtm_test_train_10_1  screenlog.53.gtm_test_train_10_1      screenlog.76.gtm_test_train_51_1
screenlog.30.gtm_test_train_10_1  screenlog.54.gtm_test_train_10_1      screenlog.77.gtm_test_train_51_1
screenlog.31.gtm_test_train_10_1  screenlog.55.gtm_test_train_10_1      screenlog.78.gtm_test_train_51_1
screenlog.32.gtm_test_train_10_1  screenlog.56.gtm_test_train_51_1      screenlog.79.gtm_test_train_51_1
screenlog.33.gtm_test_train_10_1  screenlog.57.gtm_test_train_51_1      screenlog.80.gtm_test_train_51_1
```

Рис. 2.11. Запис скринлогів у файли.

Як експеримент візьмемо задачу сумування багатьох пар чисел. Тоді в кожному скринлозі буде міститися результат для окремої пари. Це продемонстровано на рисунку 2.12.

```
farund2007@DESKTOP-PKNFLEL:/mnt/c/Users/Nikita/Desktop/gtm-test$ cd ~
farund2007@DESKTOP-PKNFLEL:~$ cat screenlog.116.gtm_test_train_10_1
Script is scripts/run.sh train --a=10 --b=1
11
farund2007@DESKTOP-PKNFLEL:~$
```

Рис. 2.12. Приклад скринлогу для задачі сумування.

Далі наведено приклад більш комплексної задачі, такої як запуск тренування та тестування нейронної мережі для 3D-визначення об’єктів. На рисунку 2.13 можна побачити результат виконання, записаний в скринлог, для конкретного набору даних.

```
Script is scripts/run.sh evaluate --config_path=./configs/pointpillars/car/xyres_24.proto --model_dir=/home/io3/TANet/pointpillars_wi
middle_class_name PointPillarsScatter
Restoring parameters from /home/io3/TANet/pointpillars_with_TANet/models-pointpillars/car_pointpillars_xyres_24_pure/voxelnet-296960.
remain number of infos: 3769
Generate output labels...
fps: 58.095973034538645 [99.97%][==================>][15.19it/s][04:49>00:00]
fps by total: 46.86107388533906
net._total_inference_count: 3769
total_count: 3769
generate label finished(13.01/s). start eval:
avg forward time per example: 0.010
avg postprocess time per example: 0.008
|| total_objects_gt: 26766
Car AP@0.70, 0.70, 0.70:
bbox AP:90.60, 88.34, 86.23
bev  AP:89.91, 84.45, 80.56
3d   AP:83.30, 73.79, 68.09
aos  AP:90.46, 87.69, 85.12
Car AP@0.70, 0.50, 0.50:
bbox AP:90.60, 88.34, 86.23
bev  AP:90.79, 90.01, 89.33
3d   AP:90.79, 89.82, 89.03
aos  AP:90.46, 87.69, 85.12

FPS | 56.39233946780315
Evaluation Mode | 1/2
Car 3D APs | [83.29897731564982, 73.79080523206152, 68.08823110497964]
Objects by difficulties | -1: 0, 0: 0, 1: 0, 2: 0
```

Рис. 2.13. Приклад скринлогу для задачі тренування нейронної мережі.

На рисунку можна бачити приклад більш комплексних даних, що свідчить про можливість системи виконувати складніші задачі.

Також у роботі присутнє опрацювання помилок, що реалізовано як на рівні REST API, так і на рівні опрацювання задач. Опрацювання помилок також забезпечено за рахунок наявності юніт-тестів, про що буде детальніше описано в наступному розділі. Приклад опрацювання помилки при запиті з боку клієнта показано на рисунку 2.14.

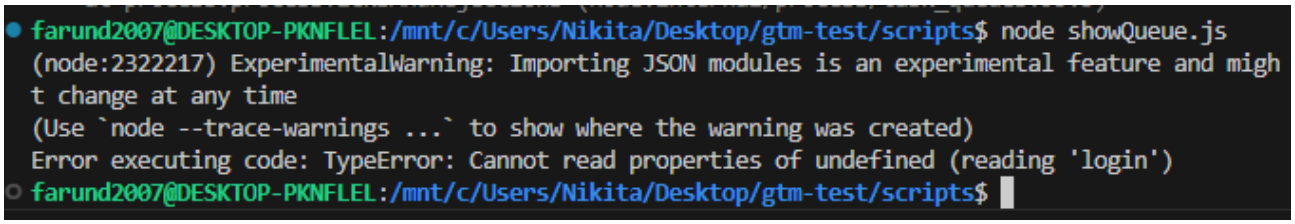


Рис. 2.14. Приклад опрацювання помилки сервером.

Крім того, система моніторингу дозволяє користувачам з низьким рівнем доступу бачити ресурси, що виділені для їхніх потреб, а також статус їхніх експериментів, що дозволяє краще планувати виконання таких експериментів.

Для користувачів з високим рівнем доступу створено можливість моніторингу всіх доступних потужностей, аби слідкувати за станом системи в цілому.

2.2.3. Робота системи через користувацькі інтерфейси.

Припустимо, що наш користувач має максимальний рівень доступу й може виконувати будь-які функції. Передусім новий користувач повинен здійснити авторизацію для того, щоб мати можливість працювати з системою. На рисунку 2.15 можна побачити процес авторизації за допомогою консольного інтерфейсу.

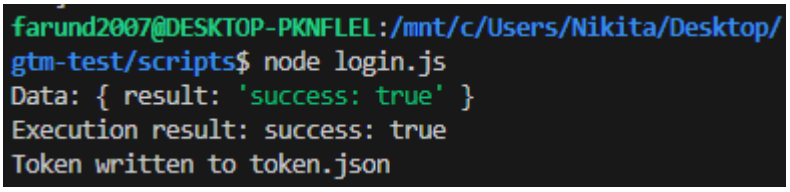


Рис. 2.15. Процес авторизації через CLI

Процес авторизації через веб-інтерфейс реалізовано за допомогою компонента “Login”, який зображено на рисунку 2.16.

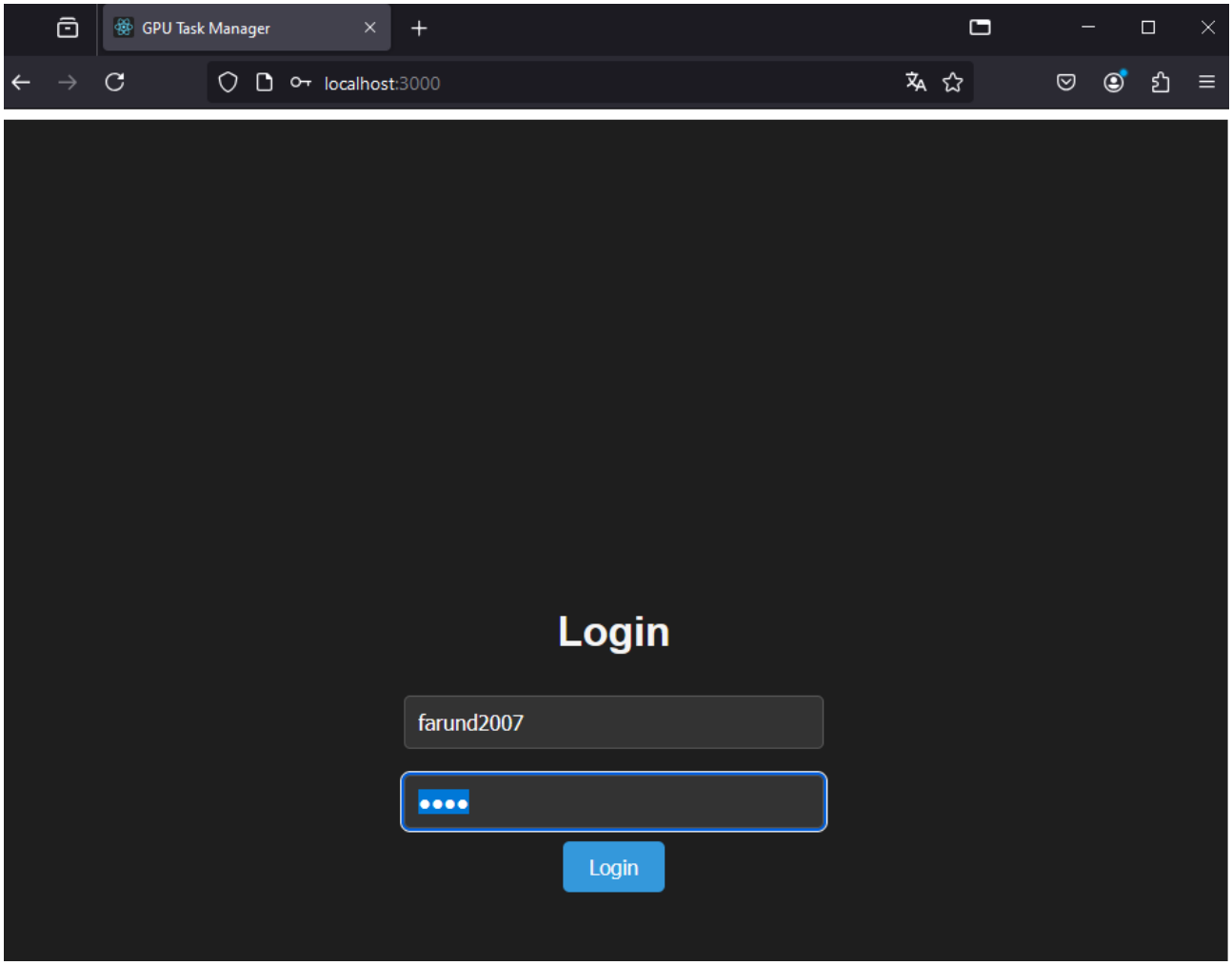


Рис. 2.16. Компонент авторизації

Даний компонент містить два текстових поля: одне для логіна, друге для пароля. При введенні правильних даних і натисканні на кнопку “Login”, спеціальний модуль React Router автоматично переносить користувача на головний екран.

Спробуймо додати задачу для виконання, для цього застосуємо функцію addTask. Для прикладу візьмемо задачу із сумуванням пар чисел, попередньо очікуючи на обчислення 100 секунд для того, щоб залишити задачу активною. Виконання функції addTask через CLI виглядає так, як показано на рисунку 2.17.

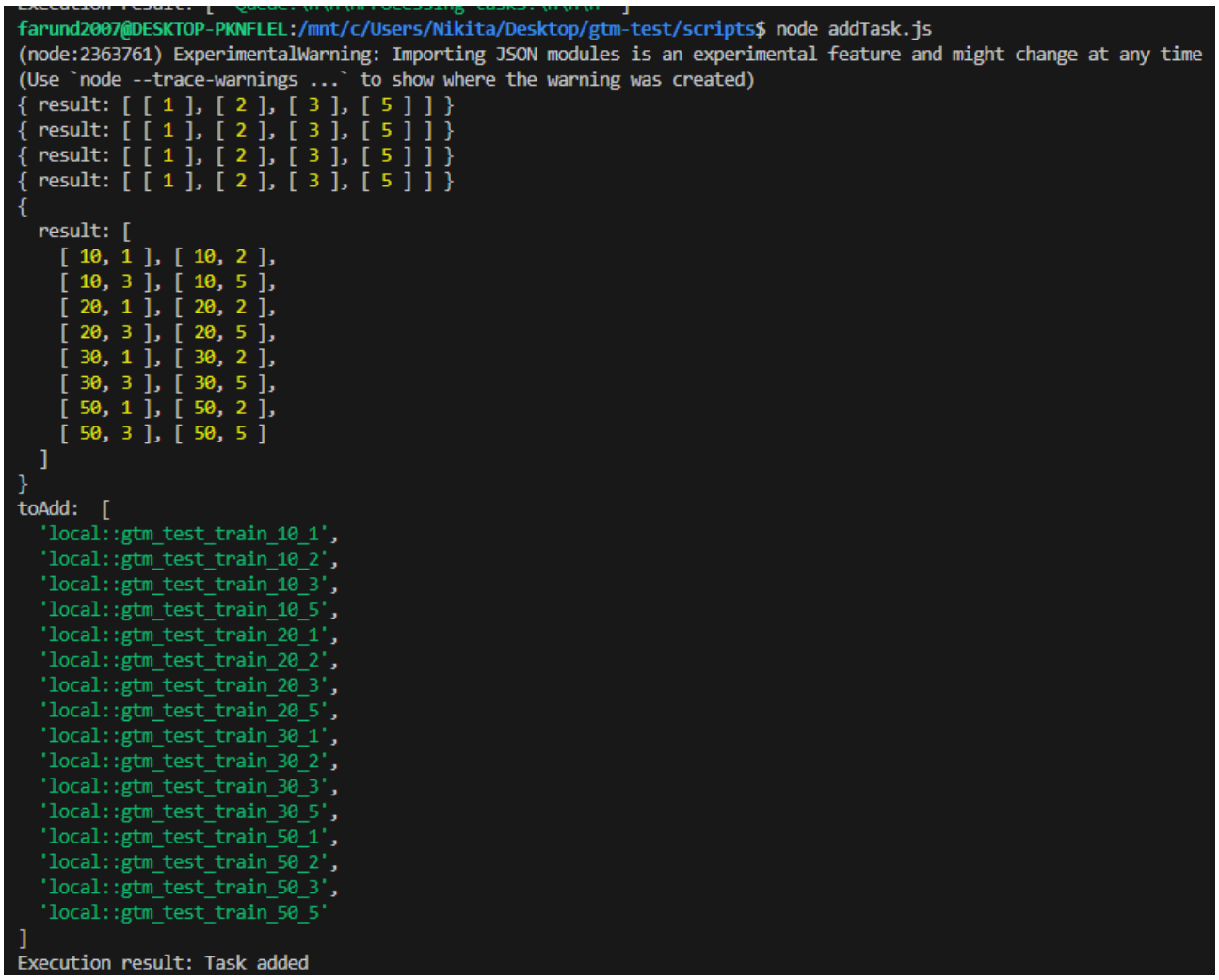


Рис. 2.17. Додавання задачі

З рисунка бачимо, що задача виконується для різних пар чисел. Використання цієї функції в GUI можна побачити на рисунку 2.18.

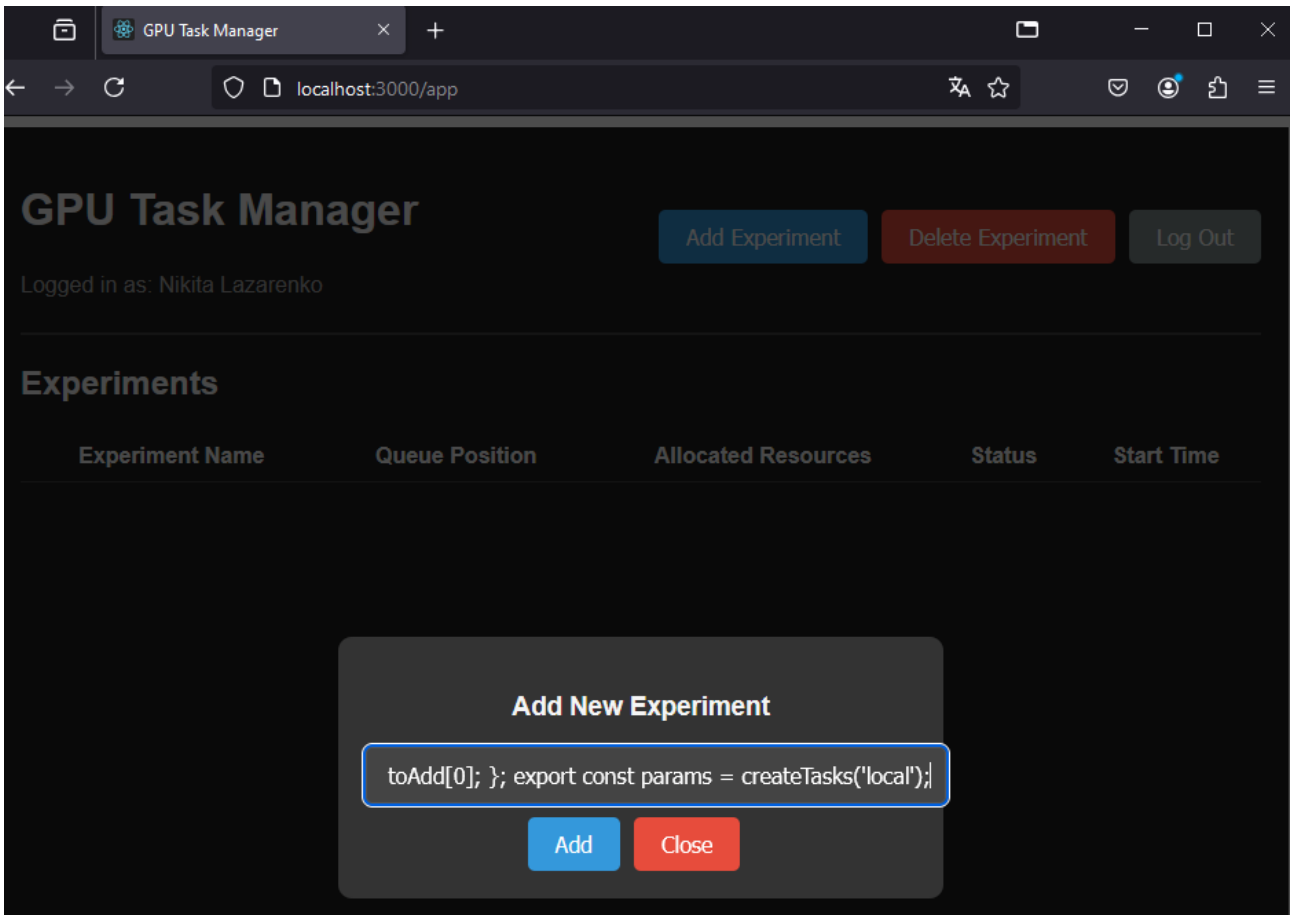


Рис. 2.18. Додавання задачі в GUI

Як бачимо, GUI надає нам можливість додати свій експеримент для виконання. Або закрити вікно. Результат додавання експерименту зображено на рисунку 2.19.

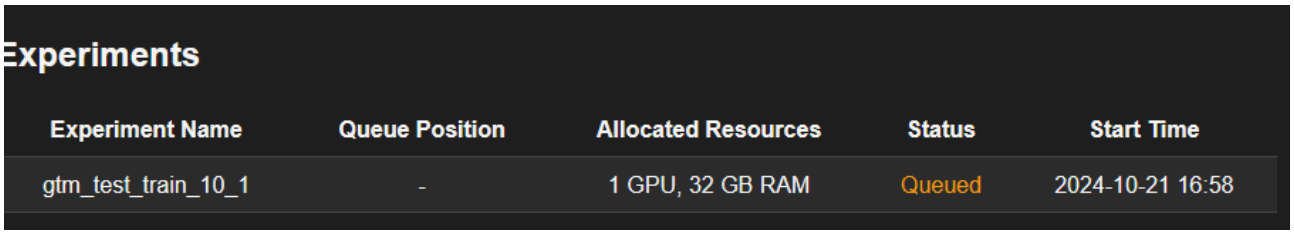


Рис. 2.19. Оновлена черга задач у GUI

Такий самий результат, що й на останньому рисунку, можна отримати в консолі, якщо використати функцію showQueue, яка показує чергу задач. Виконання цієї функції в консольному інтерфейсі показано на рисунку 2.20.

```
farund2007@DESKTOP-PKNFLEL:/mnt/c/Users/Nikita/Desktop/
gtm-test/scripts$ node showQueue.js
(node:2370590) ExperimentalWarning: Importing JSON modu
les is an experimental feature and might change at any
time
(Use `node --trace-warnings ...` to show where the warn
ing was created)
Execution result: [
  'Queue:\n' +
    '\n' +
    '\n' +
    'Processing tasks:\n' +
    'farund2007::local::gtm_test_train_10_1 #121 (1 gpu
s) script: scripts/run.sh train --a=10 --b=1\n' +
    '\n'
]
```

Рис. 2.20. Оновлена черга задач у CLI

Почекавши 100 секунд, викличемо функцію повторно, переконавшись, що задача більше не в черзі. Результат цього виконання можна побачити на рисунку 2.21.

```
farund2007@DESKTOP-PKNFLEL:/mnt/c/Users/Nikita/Desktop/
gtm-test/scripts$ node showQueue.js
(node:2360031) ExperimentalWarning: Importing JSON modu
les is an experimental feature and might change at any
time
(Use `node --trace-warnings ...` to show where the warn
ing was created)
Execution result: [ 'Queue:\n\n\nProcessing tasks:\n\n\
n' ]
```

Рис. 2.21. Порожня черга задач у CLI

У GUI задача, видалена з черги, візуально позначається за допомогою статусу «Виконано».

Задачі в такому статусі можна побачити й у консолі, звернувшись до сервера з запитом showFinishedTasks. Результат виконання відповідної функції продемонстровано на рисунку 2.22.

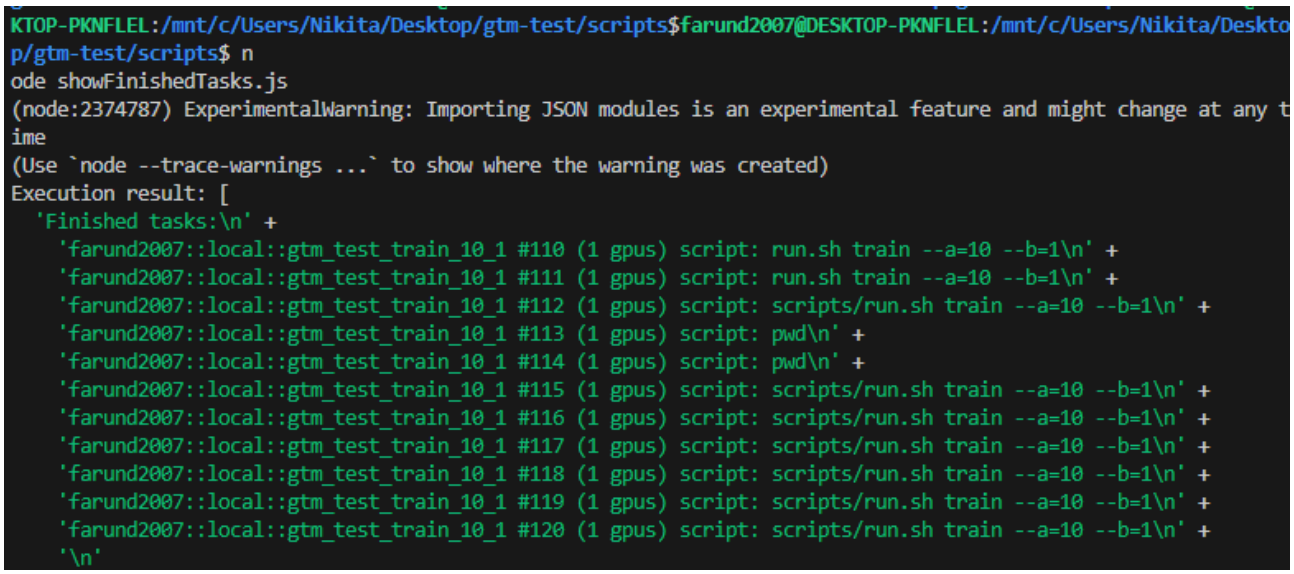


Рис. 2.22. Список завершених до цього задач у CLI

Консоль дозволяє видалити всі виконані задачі одним викликом за допомогою запиту clearFinishedTasks, результат виконання якого зображено показано на рисунку 2.21.

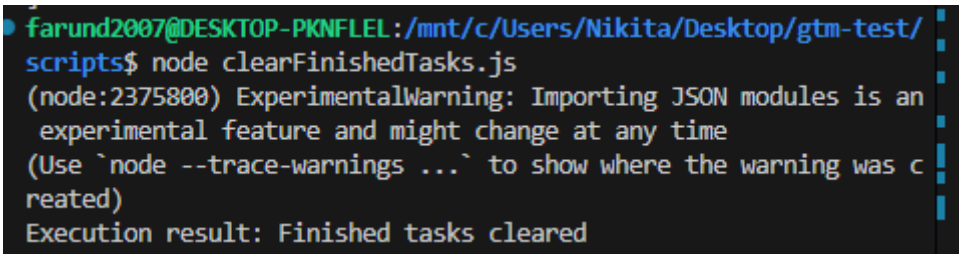


Рис. 2.21. Результат видалення завершених задач через CLI

GUI, на відміну від консольного, наразі дозволяє видалити лише по одній задачі, після чого вона зникає з загального списку.

Водночас консоль має унікальні функції, такі як створення користувача та видалення користувача. Приклад створення користувача показано на рисунку 2.22.

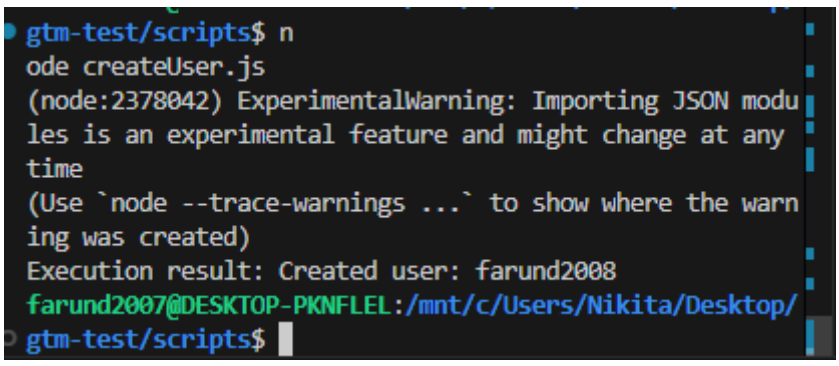


Рис. 2.22. Створення користувача через консоль

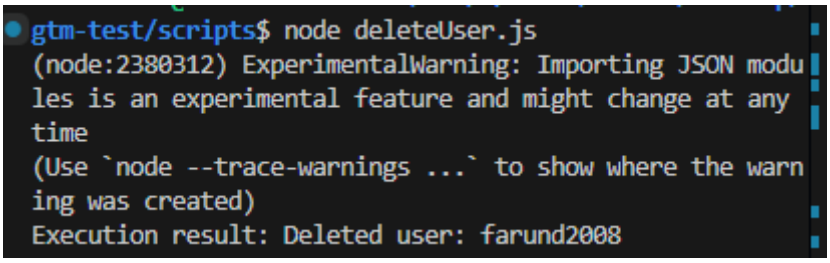
Під час виконання даного запиту для наочності створюємо користувача із заздалегідь визначеним логіном і паролем, після чого можемо побачити другого користувача в базі даних, що зображено на рисунку 2.23.

```
{
    "users":
    [
        {
            "login":"farund2007",
            "accessLevel":7,
            "password":"$2b$10$9AuIMxhlvKSSE/4eQMV0duKqaXh060yPQ60nmsMJNBgegoLjrVaVC",
            "token":"\"0r5f0cxh4x6oos6zu2knkj8\"",
            "dateGenerated":1729515148194
        },
        {
            "login":"farund2008",
            "password":"$2b$10$pXJc388WmQRreWhCwhw9zu5i8bEegy4OBWqDPLqXYA7JGR7tTnxBW",
            "accessLevel":7
        }
    ]
}
```


Рис. 2.23. Два користувачі в БД

Як видно з рисунка, другий користувач наразі не має токена та дати створення токена, оскільки він іще жодного разу не авторизувався.

Так само виконаємо запит deleteUser на видалення користувача. Результат побачимо на рисунку 2.24.



```
gtm-test/scripts$ node deleteUser.js
(node:2380312) ExperimentalWarning: Importing JSON modu
les is an experimental feature and might change at any
time
(Use `node --trace-warnings ...` to show where the warn
ing was created)
Execution result: Deleted user: farund2008
```


Рис. 2.24. Результат видалення користувача

Перевіримо після цього БД і переконаємося, що користувача більше не існує. Це наочно показано на рисунку 2.25.

Рис. 2.25. Результат видалення користувача

CLI також має функції setPassword і setAccessLevel, які встановлюють пароль і рівень доступу, відповідно. На рисунку 2.26 зображено результат зміни пароля користувачем.

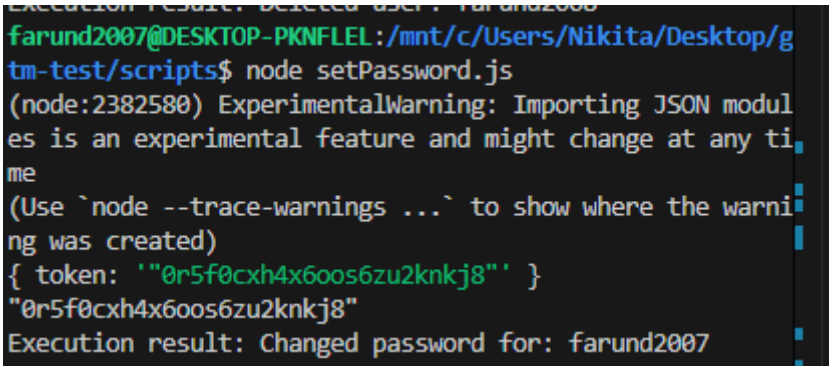


Рис. 2.26. Результат зміни пароля

У підсумку для користувача створюється новий хеш пароля й токен, які записуються в БД, що можна побачити на рис. 2.27.

Рис. 2.27. Відображення зміни пароля в БД

Порівнявши БД до зміни пароля та після, легко помітити зміну відповідних даних. Таким самим чином працює й запит setAccessLevel.

Важливою функцією також є можливість деавторизуватися, яка реалізована за допомогою запиту logout із відповідною функцією. У GUI при деавторизації React Router переносить назад на компонент з авторизацією. У консолі отримуємо повідомлення про успішний logout. Система дозволяє також робити допоміжні запити, такі як help, workstations, repeatTask, clearAllTasks для більшої зручності роботи з нею.

Висновок до розділу 2

У другому розділі було розглянуто архітектуру розробленої системи та її функціональність. На прикладах як графічного, так і консольного інтерфейсів було продемонстровано взаємодію між різними частинами системи: клієнтом, виконавцем і сервером. Також показано реалізацію підходів і компонентів, які забезпечують таку взаємодію – зокрема, реалізацію механізму авторизації, системи рівнів доступу, програмного інтерфейсу REST API, логування, опрацювання помилок і моніторингу ресурсів. Було згадано допоміжні бібліотеки, які надали можливість використання реалізованих методів і компонентів, тим самим зробивши процес розробки більш ефективним.

Було показано, що система здатна виконувати як відносно прості, так і комплексні задачі. При цьому для підтвердження коректності виконання всіх функцій програмного забезпечення є необхідність у здійсненні його тестування, що буде зроблено в наступному розділі.

РОЗДІЛ 3
ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Юніт-тестування

Юніт-тестуванням (ЮТ) називають таке тестування програмного забезпечення, при якому програма розбивається на окремі одиниці, або юніти, кожен із яких окремо перевіряється на правильність роботи.

Такий тип тестування є ефективним для комплексних систем [28], оскільки фокусується на окремих їхніх компонентах.

Часто для ЮТ використовуються спеціальні фреймворки, як-от Jest [29], проте в даній роботі ЮТ реалізовано «з нуля» задля максимальної гнучкості та налаштування під потреби розробки, адже деякі запити до сервера та відповідні їм функції є досить комплексними з точки зору реалізації, до того ж їх може бути корисно тестувати як на боці клієнта, так і на боці самого сервера.

У даній системі ЮТ запроваджено передусім на боці сервера, для чого було створено по одному скрипту на тестування кожного запиту. Було розроблено функціональність, що дозволяє здійснювати один запуск для всіх юніт-тестів. Це було реалізовано за допомогою контролера тестів. Принцип роботи ЮТ в даній системі можна побачити на рисунку 3.1.

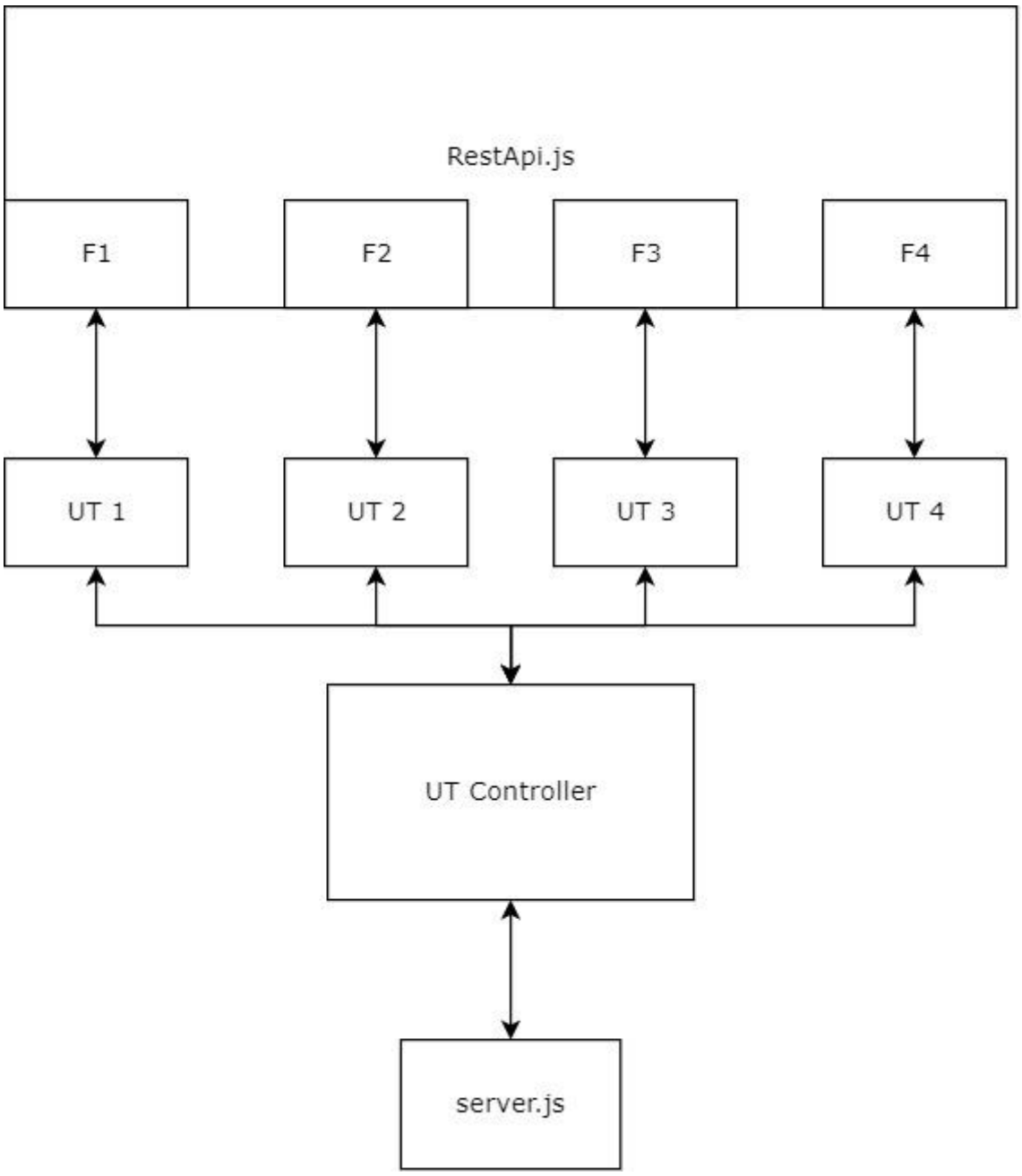


Рис. 3.1. Реалізація ЮТ

Для кожного юніт-тесту було створено один свідомо невдалий сценарій і один успішний. Зручно створювати перший сценарій саме невдалий, аби потім на основі цього знати, як виправити некоректне відпрацювання запиту або функції. [30].

Приклад запуску юніт-тестів і результатів їх виконання показано на рисунку 3.2.

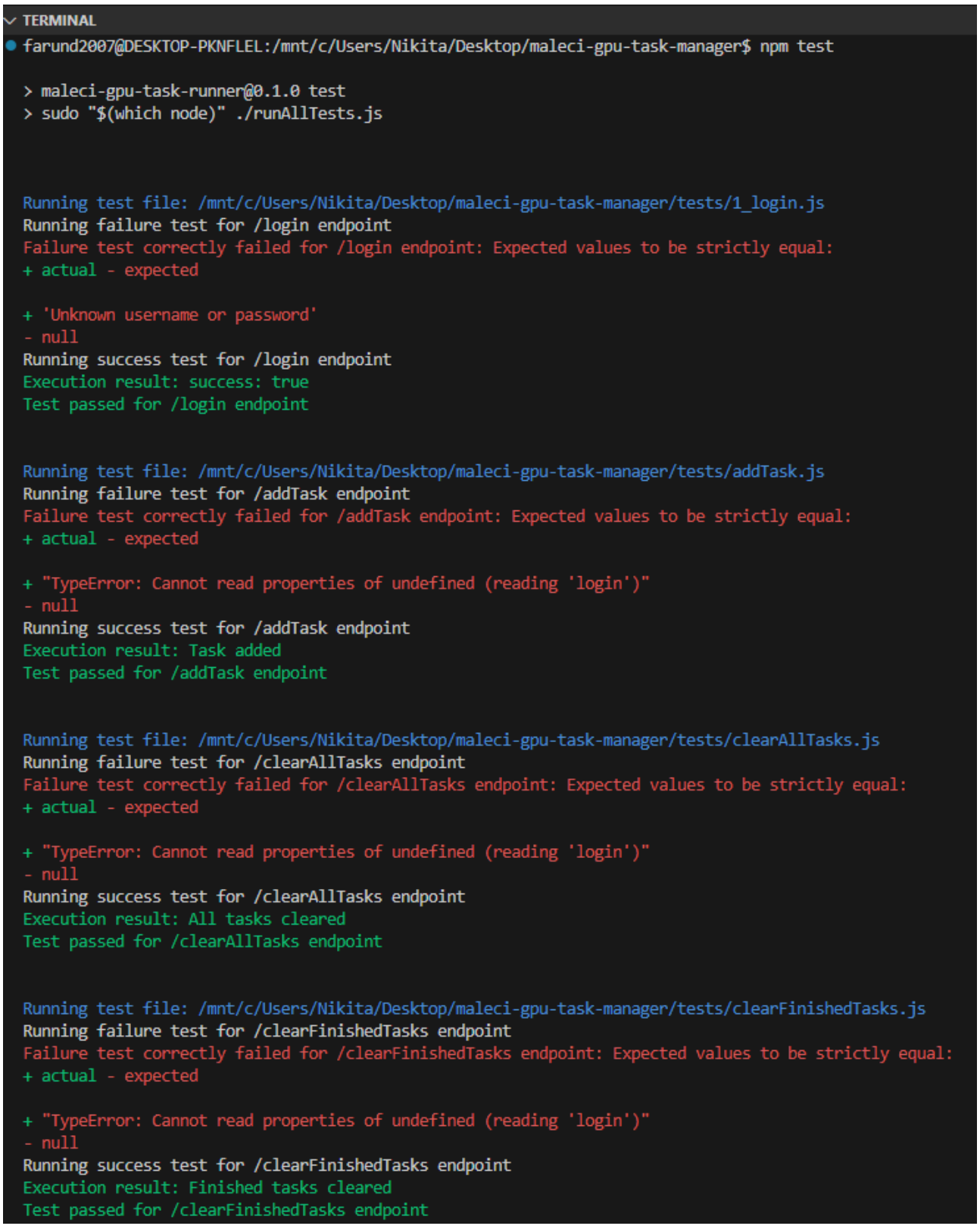


Рис. 3.2. Приклад ЮТ

З рисунка можна побачити, що для кожного тесту існує як провальний, так і успішний кейс. Кожен тест можна розширити до більшої кількості кейсів у залежності від того, наскільки комплексним є компонент програми, що тестується, скільки даних він опрацьовує. Перевіряючи численні варіанти

можливих підстановки даних і різні комбінації викликів, можемо «покрити» якомога більше сценаріїв використання нашої системи. У даному випадку найважливішим є безпека, тому тестування застосовано передусім для перевірки авторизації та дійсності токенів.

На таблиці 3.1. можна побачити покриття запитів до сервера й відповідних функцій юніт-тестами.

Таблиця 3.1

Покриття юніт-тестами

Функція	**Наявність юніт-теста**
login	+
addTask	+
showQueue	+
showFinishedTasks	+
clearFinishedTasks	+
clearAllTasks	+
repeatTask	-
setPassword	+
setPassword(targetUser)	-
setAccessLevel	+
createUser	+
deleteUser	+
workstations	-
help	+
logout	+

Згідно з наведеними в таблиці даними, у системі покрито 80% існуючих запитів.

3.2. Системне тестування

Оскільки дана робота розрахована на багатьох клієнтів, а отже і на декілька обчислювальних машин, то є доцільним провести тестування взаємодії компонентів системи з іншого пристрою [31]. Для кращого покриття пристроїв було обрано смартфон на базі операційної системи iOS. Дізнавшись адресу сервера в локальній мережі, можемо здійснити підключення до нього з мобільного пристрою. На рис. 3.3 можна побачити успішний запуск системи.

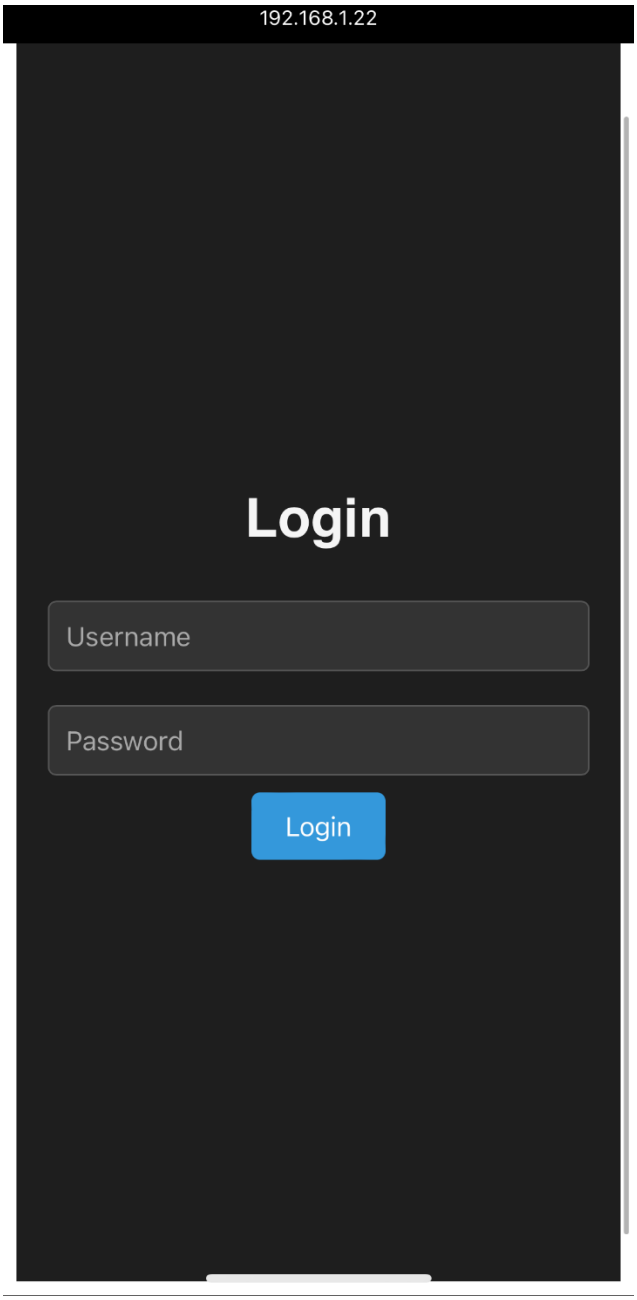


Рис. 3.3. Запуск системи з мобільного пристрою

Увівши дані для входу, система дозволяє користувачу з іншого пристрою потрапити на головний компонент з чергою задач та можливістю взаємодіяти з системою, що зображено на рис. 3.4.

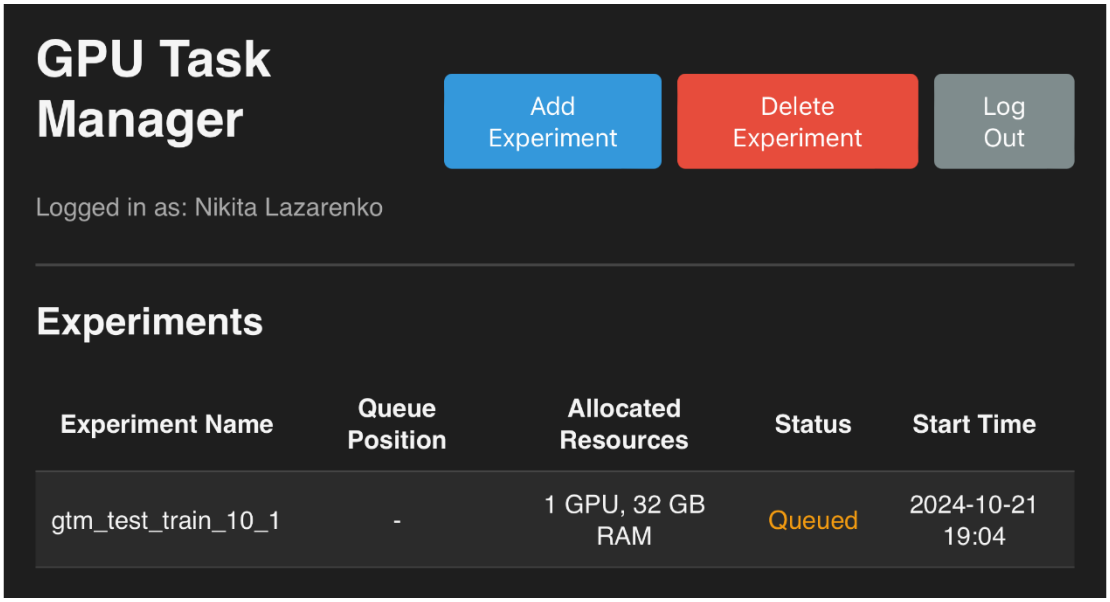


Рис. 3.4. Доступ до головного компонента з мобільного пристрою

3.3. Якісне порівняння результатів роботи системи й аналогів

Оскільки метою роботи є підвищення ефективності використання обчислювальних ресурсів, нижче наведено порівняння системи з найближчими аналогами за даним критерієм, що виражений у зручності взаємодії користувачів з системою, а також швидкості виконання експериментів. Як аналоги взято Scrum і K8s. Для порівняння наведено відгуки дослідників у галузі машинного навчання після використання аналогів і самої системи в Орхуському університеті, де вона була запроваджена. Також наведені метрики щодо затримок під час виконання експериментів.

3.3.1. Взаємодія зі Scrum.

Під час роботи зі Scrum дослідники користувалися кластером із різних робочих станцій, у якому був задіяний Scrum, що не надавав можливості самостійно встановити бібліотеки, а дозволяв лише користуватися

встановленими. До того ж, ураховуючи, що кластер на момент експерименту існував протягом тривалого часу, його оновлення та налаштування під нього Scrum було кропіткою задачею, через що дослідникам довелося відмовитися від цих дій, залишивши роботу з наявними проблемами. В умовах єдиної файлової системи для всієї їхньої роботи з'явилися проблеми з бібліотеками, які компілюються, що призводило до неможливості запуску виконуваних файлів. Крім того, базовий підхід Scrum полягає в можливості виконання декількох експериментів з уже готовим кодом, через що процес відлагодження був незручним, його використання не було інтуїтивно зрозумілим, що призводило до проблем з бронюванням ресурсів під конкретну задачу. Це проявлялося в тому, що для того, щоб перевірити результати виконання експерименту після невеликих змін у програмному коді, доводилося заново надсилати запит на отримання ресурсів, що, своєю чергою, призводило до зайвого очікування.

3.3.2. Взаємодія з K8s.

Працюючи з K8s, дослідникам доводилося регулярно оновлювати версію програмного забезпечення Docker, яке слугувало інструментом для керування ізольованими Linux-контейнерами, що містили експерименти й усі їхні залежності. Це призводило до проблем з сумісністю під час проведення експериментів. За твердженням дослідників, використання K8s є доцільним для завершеного програмного забезпечення, але в наукових дослідженнях з використанням нейронних мереж завершених програм не існує, адже кожна така програма представляє собою безперервне прототипування, що надалі розвивається іншими дослідниками. Це зумовлено тим, що науковий проєкт не вимагає створення комерційної системи, відповідно, така система не досягає високого рівня технологічної готовності продукту (TRL) за шкалою NASA [32]. Більшість наукових проєктів досягають 4-го з 9-и рівнів TRL [33].

Таким чином, K8s, як і Scrum, вимагав занадто більше часу для відлагодження. Крім того, для моніторингу стану експериментів доводилося здійснювати комплексне налаштування K8s під власні потреби, що вимагало

додаткового часу на дослідження. Це є особливо критичним для дослідників в умовах обмеженого бюджету.

3.3.3. Взаємодія з розробленою системою.

При роботі з даною системою дослідники мали найменші затримки між моментом надсилання задачі на виконання та фактичним початком виконання задачі за миттєвої доступності ресурсів. В умовах очікування ресурсів дані можуть відрізнятися. Дані про затримки наведено в таблиці 3.2.

Таблиця 3.2

Затримки під час виконання експериментів, подані в секундах

Назва експерименту	**Опис експерименту**	**Scrum**	**K8s**	**Дана система**
addSum	Тестовий експеримент із сумування пар чисел	30	601	5
3D-object tracking	Тренування та тестування нейронної мережі для 3D-визначення об’єктів	37	608	11
Deep Learning for Financial Investor Network Analysis	Глибинне навчання для аналізу мережі фінансових інвесторів	40	612	13

Згідно з відгуком, однією з головних переваг системи на практиці є відсутність вимог до робочих станцій, крім наявності самих ресурсів, які не залежать від типу процесора, хоч і орієнтовані передусім на ГП.

Важливою є також можливість моніторингу стану системи як через GUI, так і через CLI самої робочої станції. Це надає додатковий контроль над системою для користувача з високим рівнем доступу. Наприклад, дослідникам була надана можливість терміново зупинити процеси користувачів, не

звертаючись до системи через користувацький інтерфейс. Також при необхідності відлагодження програмного коду експерименту система не вимагала від дослідників робити це через користувацький інтерфейс – у такому випадку він був потрібен лише для того, щоб забронювати потрібні обчислювальні ресурси. Усе це доповнюється можливістю швидкого переходу між двома способами взаємодії з системою. Це виключає затримки при відлагодженні, що неможливо при використанні аналогів.

Таким чином, система вносить порядок в організацію роботи наукової групи, при цьому не обмежуючи дослідників у доступі до ресурсів і повноті контролю над ресурсами.

Висновок до розділу 3

У даному розділі було описано ЮТ програмного забезпечення, яке дозволяє перевіряти коректність роботи окремих компонентів системи. Юніт-тести було розроблено самостійно, що надало максимальну гнучкість та можливість адаптації під специфічні потреби розробки. Основна увага приділялася тестуванню серверних запитів, включаючи перевірку авторизації та валідності токенів, що є критично важливим з точки зору безпеки.

Кожен тест містив провальний і успішний кейс, що допомагає виявляти та виправляти помилки. Також система підтримує запуск усіх тестів за допомогою контролера юніт-тестів. Загалом, юніт-тестами було охоплено 80% запитів до сервера.

Крім цього, для перевірки взаємодії компонентів системи з різних пристроїв було здійснено тестування з мобільного пристрою на базі iOS, що підтвердило можливість коректного доступу до головного компонента системи через мобільний інтерфейс.

Було також наведено переваги даної системи над аналогами на основі відгуків про практичне використання системи дослідниками Орхуського університету.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЄКТУ

4.1. Опис реалізації стартапу

Реалізація стартап-проєкту передбачає комплексний підхід до розробки та впровадження менеджера задач. Процес створення містить такі ключові етапи:

- Аналіз концепції та ринкового потенціалу: На етапі попереднього аналізу проведено дослідження актуальності багатокористувацького менеджера задач графічного процесора для запуску коду на розподіленій системі та оглянуто наявні рішення, їхні переваги й недоліки. Визначено ключові концепції та переваги, які мав намір реалізувати проєкт.
- Аналіз структури програмного забезпечення та формальне розв'язання проблеми: З проведеного аналізу визначено функціональні вимоги системи. Створено діаграми мовою UML для кращого розуміння можливостей системи.
- Вибір технологій: На цьому етапі обрано оптимальний стек технологій, зокрема вибір платформи для менеджера задач, інструментів розробки, технологій зберігання та опрацювання даних.
- Розробка та тестування: Здійснено програмування ядрової функціональності та інтерфейсу застосунку. Впроваджено систему тестування для перевірки правильності роботи та виявлення можливих проблем.
- Впровадження та масштабування: Система готова до масштабування, реагуючи на зростання кількості користувачів та розширюючи функціональність відповідно до потреб аудиторії. Реалізація стартап-проєкту об'єднала в собі чітке формулювання задачі та її ефективну реалізацію за допомогою обраних технологій, результатом чого став менеджер задач, що має переваги в порівнянні зі своїми конкурентами.

Таблиця 4.1

Інформаційний опис проєкту

Назва	**Опис**
1. Тема проєкту	Багатокористувацький менеджер задач графічного процесора для запуску коду на розподіленій системі
2. Автор	Лазаренко Н.В.
3. Скорочена анотація	Розроблений менеджер задач є простою, інтуїтивною та ефективною платформою для багатокористувацького керування завданнями GPU, яка дозволяє запускати програмний код на розподілених системах із максимальною продуктивністю та мінімальними витратами.
4. Термін реалізації	10-12 місяців
5. Назва	Multi-user GPU task manager for distributed systems

Таблиця 4.1 (продовження)

Інформаційний опис проєкту

6. Ресурси	Обладнання: • ноутбук – 2 одиниці • смартфон – 2 одиниці Сервіси та програми для реалізації: • VSCode для написання Front-end та • Back-end • WSL для запуску виконуваних файлів операційної системи Linux в середовищі системи Windows • сервіс Github Premium Фінансування: • платна підписка на Github Premium • платна підписка на хмарне сховище • оплата електроенергії та інтернету
7. Проблеми, які вирішує проєкт	Розроблена система надає можливість запуску програмного коду на розподіленій системі, що працює на ресурсах ГП. Крім того, система є масштабованою для того, щоб відповідати змінам при використанні кількома користувачами та збільшенні робочого навантаження. Система дозволяє розподіл ресурсів ГП між декількома користувачами, які використовують власні ГП, а також надає можливість колаборації та можливість безкоштовного доступу до ГП надані за замовчанням

Таблиця 4.1 (продовження)

Інформаційний опис проєкту

8. Головна мета та завдання проєкту	Створення багатокористувацького менеджера задач для ефективного використання обчислювальних ресурсів у системах, що містять декілька робочих станцій з одним або більше ГП.
9. Результати	Розроблений менеджер задач графічного процесора представляє систему, що увібрала у себе найкращі риси конкурентів, а також має інтуїтивно зрозумілий інтерфейс. Проєкт успішно вирішує низку проблем та відповідає на сучасні потреби дослідників в галузі виконання задач на розподіленій системі.

4.2. Загальна ідея стартапу

Багатокористувацький менеджер задач ГП, який дозволяє запускати програмний код у розподіленій системі з акцентом на простоті використання, спільній роботі та ефективності ресурсів. Акцент робиться на високопродуктивних обчисленнях, що є доступними для розробників, команд та компаній через зручний інтерфейс, спільноту користувачів та можливість розподілу ресурсів. Система дозволяє дослідникам використовувати потужності кількох ГП для виконання складних обчислювальних завдань, таких як тренування моделей машинного навчання, наукові симуляції чи рендеринг.

Таблиця 4.2

Ідея стартапу

Ідея	Застосування	Переваги для користувача
Багатокористувацький менеджер задач ГП, який дозволяє запускати програмний код у розподіленій системі з акцентом на простоті використання, спільній роботі та ефективності ресурсів. Поєднує ресурси від власників ГП з потребами розробників, дослідників та компаній. Забезпечує масштабованість, продуктивність та економічність шляхом автоматичного розподілу завдань на доступних ГП. Можливість зробити високопродуктивні обчислення доступними для всіх — від індивідуальних розробників до компаній.	1. Команди, групи команд або компанії, яким актуально запускати код на розподілених системах, маючи справедливий розподіл ресурсів.	Можливість виконувати експерименти розподіленій системі із максимальною продуктивністю та мінімальними витратами, маючи зручний та інтуїтивно зрозумілий GUI. Користувачі можуть підключати додаткові GPU лише тоді, коли це необхідно. Платформа використовує сучасні протоколи шифрування для безпечного виконання завдань. Можливість чіткого налаштування прав для членів команд. Легкий доступ до ресурсів для швидкого зростання без великих капіталовкладень.

Таблиця 4.2 (продовження)

Ідея стартапу

Надання інструментів для моніторингу виконання задач, що дозволяє користувачам оптимізувати використання ресурсів, знижуючи час опрацювання та підвищуючи ефективність.		Можливість для команд працювати разом над обчислювальними проєктами: ділитися кодом, завданнями та ресурсами.
	2. Окремі розробники, що бажають здійснювати реалізацію власних проєктів	Окремі розробники не потребують власних дорогих графічних процесорів, оскільки можуть орендувати ресурси ГП через платформу. Це дозволяє їм знизити витрати на обладнання, доступаючись до високопродуктивних ресурсів для вирішення складних обчислювальних задач, таких як тренування моделей машинного навчання чи рендеринг.

Таблиця 4.3

Опис унікальних властивостей проєкту

№ п/п	Назва властивості	Пояснення
1	Можливість розподіляти ресурси одного ГП	Можливість розподіляти ресурси одного ГП є важливою для підвищення ефективності використання обчислювальних потужностей у багатокористувацьких середовищах, особливо для розробників чи дослідницьких команд із невеликими власними ресурсами. Ця функція дозволяє одночасно виконувати кілька завдань або програм різними користувачами на одному ГП, що зменшує простої обладнання та підвищує продуктивність системи.
2	Можливість розподіляти права в межах організації	Можливість розподіляти права в межах організації є важливою особливістю, яка сприяє ефективному управлінню ресурсами та забезпечує контрольовану, безпечну й структуровану роботу з системою. Ця функція забезпечує контроль доступу до ресурсів, що запобігає несанкціонованому використанню потужностей та допомагає уникати перевантаження ГП нецільовими задачами. Також це допомагає захистити конфіденційні дані, ізолюючи доступ до чутливої інформації та залишаючи її доступною лише для авторизованих осіб. Розподіл прав у цілому надає гнучкість управління ролями, що дозволяє різним категоріям користувачів працювати з системою відповідно до їхніх функціональних обов'язків.

Таблиця 4.4

Визначення сильних, слабких та нейтральних характеристик проєкту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтраль-на сторона)	S (сильна сторона)
		Мій проєкт	Конку-рент1	Конку-рент2			
1	Бюджетне фінансування	Розробка за класний рахунок	Використання інвестицій	Розробка за класний рахунок	Відсутність фінансування	Часткове фінансування	Повне фінансування проєкту
2	Використання сучасної техніки та якісної матеріальної бази	Використання сучасних технологій	Використання сучасних технологій	Використання сучасних технологій	Застосування застарілих технологій	Часткова технічна комплектація	Застосування новітніх технологій
3	Наявність маркетингової стратегії	Достатнє використання реклами	Достатнє використання реклами	Недостатнє використання реклами	Відсутність рекламування	Часткова трансляція	Активна трансляція
4	Використання бізнес-послуг	Наявна	Наявна	Відсутня	без консультування	Часткове консультування	Ведення проєкту

Таблиця 4.5

Порівняльна таблиця із найближчим конкурентом

№	Характеристика	Проєкти		П1	П2
		Власний стартап-проєкт	**K8s**		
1	Можливість розподіляти ресурси одного ГП	Наявність різноманітних екранів та інтуїтивного інтерфейсу навігації між ними,	Сервіс не надає можливість розподіляти ресурси одного ГП, найменшою функціональною одиницею є один ГП.	+	
2	Наявність колаборації	Надає можливість різним експериментатор взаємодіяти між собою.	Відсутня взаємодія між користувачами	+	

Таблиця 4.5 (продовження)

Порівняльна таблиця із найближчим конкурентом

3	Безкоштовний доступ до ГП	За замовчанням базовий доступ до функціональності системи та потужностей надається безкоштовно	Немає можливості використання потужностей поза власними	+	
4	Наявність GUI	Наявність інтуїтивно зрозумілого веб-інтерфейсу.	GUI відсутній	+	

4.3. Технічний аудит проєкту

Технічний аудит стартапу дозволяє оцінити його інфраструктуру, програмне забезпечення, безпеку, ефективність роботи та здатність масштабуватися. Ось розгорнута оцінка технічних аспектів проєкту:

- Архітектура платформи
    - Тип архітектури: Розподілена система, яка дозволяє запускати задачі на різних ГП на віддалених серверах.
    - Масштабованість: Система повинна бути здатною до горизонтального масштабування для підтримки збільшення кількості користувачів і задач.

- Розподіл задач: Алгоритм розподілу задач між ГП має бути оптимізованим для зменшення часу виконання і мінімізації простоїв обладнання. Система має автоматично обирати найкращі ресурси для кожної задачі в залежності від вимог.
- Комунікація між вузлами: Важливо використовувати ефективні методи передачі даних між вузлами для мінімізації затримок у процесі обчислень, зокрема з урахуванням вимог високої пропускної здатності.

- Інфраструктура
  - Обчислювальні ресурси: Проєкт покладається на обчислювальні потужності ГП, тому необхідно враховувати різні типи ГП і забезпечити підтримку їхніх специфікацій та драйверів.
  - Інтерфейси для користувачів: зручний інтерфейс для моніторингу і управління задачами, з можливістю підключення різних ГП до одного обчислювального завдання.

Таблиця 4.6

Технології реалізації

№	Опис	Технології реалізації	Наявність технологій	Можливість використання технологій
1	Безпека передачі даних між вузлами	Bcrypt	+	Безкоштовно
2	Зберігання даних користувачів	JSON	+	Безкоштовно

Таблиця 4.6 (продовження)

Технології реалізації

3	Реалізація користувацького інтерфейсу	React	+	Безкоштовно
4	Запуск виконуваних файлів операційної системи Linux в середовищі системи Windows	WSL	+	Безкоштовно
5	Реалізація екранів та навігації між ними	React	+	Безкоштовно

4.4. Аналіз та вивчення ринкових можливостей для старту проєкту

Успіх стартапу багато в чому залежить від того, як добре він розуміє та використовує ринкові можливості. Проєкт, що пропонує платформу для запуску задач на ГП в розподілених системах, орієнтований на ринки, де високопродуктивні обчислення мають значення. Це можуть бути області, що потребують значних обчислювальних потужностей, як-от машинне навчання, наукові дослідження, рендеринг 3D-графіки, блокчейн-технології тощо.

- Зростання попиту на обчислювальні потужності
    - Машинне навчання та штучний інтелект (AI/ML): Сучасні технології машинного навчання, особливо глибинне навчання, потребують потужних обчислювальних ресурсів, а ГП є оптимальними для цих завдань завдяки своєму паралельному опрацюванню даних.

- Опрацювання великих даних: Зростання кількості задач, що потребують опрацювання великих даних (Big Data), також сприяє підвищеному попиту на потужні обчислювальні ресурси.
- Ігрова індустрія та рендеринг: Високоякісна графіка в іграх та кіно вимагає масштабованих ГП потужностей для рендерингу [34].
- Блокчейн і криптовалюти: Проєкти, що використовують блокчейн та криптовалюти, також потребують великих потужностей для майнінгу або аналізу транзакцій.

- Хмарні обчислення та розподілені системи
  - Хмарні платформи (K8s, Slurm, Google Cloud): Вже багато хмарних платформ надають можливість орендувати потужності ГП для обчислень. Однак з огляду на зростаючі витрати на такі ресурси, можливість ефективного управління і розподілу задач між ГП може стати важливою конкурентною перевагою.
  - Розподілені обчислення: Розподілені обчислювальні системи стають популярними завдяки їхній здатності розподіляти навантаження серед великої кількості вузлів і масштабувати обчислення в залежності від потреб користувачів.
- Прямі конкуренти
  - кластери ГП та оренда потужностей ГП: Компанії, як-от, Google Cloud GPU, вже надають хмарні послуги для обчислень з використанням ГП. Вони дозволяють запускати великі задачі на графічних процесорах, проте вони не пропонують гнучких інструментів для управління розподіленими обчисленнями серед декількох ГП.
  - HPC-платформи: Інші конкуренти, які спеціалізуються на високопродуктивних обчисленнях, можуть надавати рішення для наукових розрахунків або інженерних задач, однак вони можуть бути менш доступними для індивідуальних розробників і стартапів.
- Непрямі конкуренти

- Інструменти для розподілених обчислень на рівні бібліотек: Бібліотеки та інструменти для роботи з ГП, такі як CUDA, OpenCL, TensorFlow для машинного навчання, вже дозволяють запускати код на графічних процесорах. Однак, ці інструменти не забезпечують ефективного управління великими задачами на розподіленій системі з кількома ГП і не інтегрують різні обчислювальні вузли в єдину систему.
- Інструменти для управління кластерами: Платформи, як-от K8s, дозволяють організовувати та оркеструвати ресурси в кластері, але їх потрібно адаптувати для ефективної роботи з ГП, що може бути складним завданням для окремих розробників.

Таблиця 4.7

Попередній аналіз ринку для стартапу

№	**Назва показника**	**Характеристика**
1	Кількість ключових персон в стартап-проєкті	2
2	Обсяг потенційних продажів	1 млн. дол.
3	Якісна оцінка динаміки ринку	Зростає
4	Наявність обмежень	Немає

Таблиця 4.7 (продовження)

Попередній аналіз ринку для стартапу

5	Вимоги до сертифікації та стандартизації	ISO 9001:2015: Система управління якістю. ISO/IEC 27001: Система управління інформаційною безпекою CE-маркування: Відповідність європейським стандартам безпеки та якості. GDPR: Захист особистих даних користувачів та дотримання європейських нормативів CCPA: Захист особистих даних резидентів штату Каліфорнія. EU AI: відповідність політиці Євросоюзу щодо використання технологій, пов'язаних зі штучним інтелектом
6	Норма рентабельності	60%

Таблиця 4.8

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти	Відмінності поведінки потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Потреба у масштабованості для обчислень великих обсягів даних	Великі компанії у сфері IT, відеоігор, анімації	Пріоритет – стабільність та швидкість обробки;	Підтримка кластерів із багатьох ГП; висока надійність;

Продовження таблиці 4.8

Характеристика потенційних клієнтів стартап-проєкту

2	Необхідність ефективного використання обчислювальних потужностей ГП в багатокористувацькому середовищі	Малі та середні організації, що займаються машинним навчанням, обробкою даних або 3D-рендерингом	Часто працюють з обмеженими ресурсами; шукають доступні рішення для максимізації продуктивності; потребують простого інтерфейсу	Висока ефективність розподілу ресурсів; зручний та інтуїтивний GUI; підтримка різних платформ і операційних систем
3	Брак програмного забезпечення для оптимізації розподілу задач між кількома користувачами	Наукові дослідники та академічні установи	Мають невеликий бюджет; потребують гнучких інструментів для роботи в команді; часто використовують відкритий код	Гнучкість у налаштуванні системи; прозоре логування дій користувачів; підтримка популярних бібліотек (CUDA, OpenCL)
4	Відсутність прозорого механізму розподілу прав та обмежень для користувачів	Організації зі складною внутрішньою структурою та багатьма відділами	Вимагають чіткого розподілу ролей і прав доступу; орієнтовані на безпеку даних	Гнучке управління правами; налаштування політик для різних груп

Таблиця 4.9

Аналіз загроз запуску та роботи стартап-проєкту

№ п/п	Фактор загрози	Опис загрози	Планове реагування компанії
1	Недостатній рівень попиту	Прийняття невдалих маркетингових рішень може призвести до низького рівня попиту на ПЗ	Перегляд маркетингової стратегії
2	Не належний рівень конкурентної спроможності	Велика кількість досліджень даної сфери створює серйозні конкурентні відносини	Застосування виваженої цінової політики за забезпечення якісного надання послуг
3	Агресивність конкурентного впливу	Здійснює вплив на стратегію поширення ПЗ	Використання професійних стратегій продажу
4	Нестабільність політичної ситуації	Балансування курсу	Отримання кваліфікованої юридичної допомоги
5	Складна економічна ситуація	Відсутність фінансування	Пошук додаткового фінансування та заощадження витрат
6	Складність у якісній розробці ПЗ	Складність у залученні кваліфікованих кадрів для розробки та підтримки ПЗ	Налагодження співпраці з технічними університетами для залучення молодих спеціалістів. Пропозиція гнучких моделей співробітництва

Таблиця 4.10

Ступеневий аналіз конкуренції на ринку

№ п/п	Особливості конкурентного середовища	Прояв характеристики конкурентного середовища	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції	Чиста конкуренція	Розробка унікальних функцій (GUI, розподіл ресурсів), адаптація до потреб цільової аудиторії.
2	Рівень конкурентної боротьби	Національний	Активна маркетингова кампанія всередині країни, побудова партнерських відносин із місцевими компаніями.
3	Галузева ознака	Внутрішньогалузева	Постійний аналіз конкурентів, розширення функціоналу для задоволення потреб різних груп користувачів.
4	Конкуренція за видами товарів	Товарно-родова	Впровадження інновацій, забезпечення технічної підтримки,
	Інтенсивність конкуренції	Марочна	Видозміна способів формування мережі згідно з запитами користувача

Таблиця 4.11

SWOT-аналіз стартап-проєкту

Сильні сторони: 1. Унікальний функціонал розподілу ресурсів одного ГП між кількома користувачами 2. нтуїтивний графічний інтерфейс, що забезпечує зручність використання 3. Розподіл прав доступу в межах організацій дозволяє адаптувати систему до різних сценаріїв використання 4. Використання в таких сферах, як наукові обчислення, AI, рендеринг, аналіз даних	**Слабкі сторони**: 1. Система може бути менш ефективним на застарілому обладнанні 2. Для масштабного розгортання може знадобитися значний початковий бюджет 3. Можлива потреба у постійній підтримці й оновленнях системи
Можливості: 1. Вихід на міжнародний ринок із високим попитом на ГП-обчислення 2. Співпраця з виробниками обладнання та постачальниками хмарних обчислень 3. Підтримка роботи з популярними фреймворками, як-от TensorFlow, PyTorch 4. Отримання фінансування для розробки інноваційних рішень у сфері IT	**Загрози**: 1. Високий рівень конкурентів у сфері управління ГП-ресурсами 2. Застарівання системи через появу нових технологій або альтернативних рішень 3. Високі витрати на розробку та ризик недостатньої монетизації 4. Потреба у високому рівні безпеки, щоб захистити дані користувачів

4.5. Аналіз та розробка ринкової стратегії

Розробка ринкової стратегії є важливим етапом для стартапу, що планує вихід на ринок із інноваційним програмним забезпеченням. Оскільки стартап орієнтується на технології обчислень із використанням ГП в розподілених системах, необхідно враховувати специфіку та потреби цільових користувачів, а також прорахувати можливі стратегії для максимального залучення аудиторії.

- Цільова аудиторія
    - Індивідуальні розробники програмного забезпечення та стартапи, що працюють з великими даними або обчисленнями, які потребують високопродуктивних ресурсів.
    - Дослідницькі інститути та наукові організації, що займаються вивченням великих обсягів даних або високопродуктивними обчисленнями (моделювання, обчислювальна фізика, хімія тощо).
    - Криптовалютні компанії, що потребують потужностей для майнінгу або аналізу блокчейн-даних.
    - Великі корпорації та хмарні провайдери, які хочуть оптимізувати використання ресурсів ГП для бізнес-аналітики, рендерингу, аналітики даних.
- Ринкові тенденції
    - Попит на ГП обчислення: Оскільки все більше компаній звертаються до ГП для обчислень в таких сферах, як AI, наука, блокчейн, важливо зосередитись на цих категоріях як потенційних ринках для програмно.
    - Зростання популярності хмарних послуг: Моделі оренди та використання потужностей на вимогу стають все більш популярними серед малих і середніх компаній. Це відкриває можливість для надання програмного забезпечення як хмарного сервісу.

- Унікальність системи
    - Інтуїтивно зрозумілий багатокористувацький інтерфейс: Платформа надає можливість кільком розробникам одночасно використовувати ресурси ГП, ефективно розподіляючи задачі між ними, що значно підвищує продуктивність команди.
    - Можливість розподіляти ресурси одного ГП надає додаткову зручність для дослідників.
- Позиціонування на ринку
    - Інструмент для наукових досліджень: Пропонувати рішення для організацій, що займаються науковими дослідженнями.
    - Мінімізація витрат на ГП: Пропонуючи гнучке використання потужностей через модель оплати за фактичне використання, стартап може привабити малих розробників і стартапи, яким важко дозволити собі інвестиції в дорогі ГП.

В таблиці нижче наведено аналіз факторів, що впливають на стартап-проєкт, та способи реагування на можливості, які ці фактори створюють. Ці дані визначають потенційні шляхи розвитку та стратегії управління програмним забезпеченням.

Таблиця 4.12

Фактори для можливостей

№	**Фактор**	**Опис**	**Реакція на можливість**
1	Продаж платформи компаніям з надання хмарних послуг	Компанії, котрі надають хмарні послуги, можуть бути зацікавлені в купівлі проєкту	Умови для масштабування та росту популярності

Таблиці 4.12 (продовження)

Фактори для можливостей

2	Оренда платформи організаціями чи окремими користувачами	Оренда великих потужностей для забезпечення власних потреб компанії	Організації можуть бути зацікавлені в оренді окремих потужностей для забезпечення потреб своїх користувачів не тільки потужностями як такими, але й функціональністю.

Висновок до розділу 4

Даний розділ став визначальним для формування стратегії, напрямків розвитку та можливого успіху проєкту багатокористувацького менеджера задач графічного процесора для запуску коду на розподіленій системі. Під час аналізу етапів від концепції до реалізації стартапу були виявлені інноваційні рішення та стратегічні підходи, що сприятимуть його конкурентоспроможності на ринку. Вивчення потреб користувачів допоможе вдосконалити функціональність проєкту, зробивши його більш адаптованим до реальних вимог споживачів. Визначені стратегії виходу на ринок мають на меті підвищити ефективність та популярність проєкту. Загальна ідея та стратегічне позиціонування надають проєкту унікальність і конкурентні переваги, а аналіз технічних аспектів і план впровадження підтверджують вибрані підходи, спрямовані на оптимізацію та вдосконалення процесів. Цей розділ є важливим етапом у створенні успішного стартапу, який охоплює всі аспекти — від концепції до виведення на ринок.

ВИСНОВКИ

У даній магістерській роботі було вивчено та проаналізовано ключові аспекти створення та реалізації проєкту багатокористувацького менеджера задач графічного процесора для запуску коду на розподіленій системі. Серед розглянутих питань був аналіз ринку, безпосередньо розробка проєкту, його тестування та створення стратегій його втілення.

Перший розділ показав актуальність проблеми через недостатню функціональність уже наявних на ринку аналогічних систем. Було визначено ключові відмінності проєкту, що якісно вирізняють його серед конкурентів.

У другому розділі було проведено аналіз структури програмного забезпечення та здійснено формальне розв’язання проблеми. Другий розділ також описав технічну реалізацію системи відповідно до визначених вимог. Було продемонстровано роботу системи за допомогою командного рядка, а також користувацького GUI.

У третьому розділі було описано тестування системи: за допомогою як юніт-тестування, так і системного тестування. Для повноцінної перевірки роботи системи було використано декілька пристроїв з різними ОС, що підтвердило широкі адаптивні можливості системи, а також її можливості до подальшого масштабування.

Четвертий розділ охоплює обґрунтування стартапу, етапи його реалізації, аналіз ринкових можливостей та аудит проєкту. Розглянуто вимоги щодо сертифікації та стандартизації, а також фактори для можливостей проєкту.

Отже, можна стверджувати, що створений проєкт відповідає актуальним запитам на використання багатокористувацького менеджера задач графічного процесора для запуску коду на розподіленій системі, спираючись на сучасні підходи та інноваційні технології. Розроблена система надає можливість як

окремим дослідникам, так і дослідницьким компаніям здійснювати свою діяльність з виконання задач на розподіленій системі, маючи зазначені в цьому проєкті переваги над іншими подібними проєктами.

Подальші покращення системи включають кілька напрямків розвитку, серед яких є інтеграція з хмарними сервісами для автоматичного масштабування ресурсів, наприклад, через адаптацію системи до роботи з гібридними середовищами, які комбінують локальні процесори з хмарними обчислювальними потужностями. Для підвищення рівня надійності системи уся майбутня функціональність має бути покрита тестами більше за поточні показники. З погляду користувацького інтерфейсу система має потенціал до розширення функціональності веб-інтерфейсу задля підвищення її привабливості для користувачів із різним рівнем технічної підготовки. Також одним з напрямків є підвищення рівня безпеки через упровадження шифрування та створення системи резервного копіювання та відновлення даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. From Gaming to AI: How GPUs are driving the Technology till Artificial Intelligence [Електронний ресурс] / Adithya Thatipalli. – 2023. – Режим доступу до ресурсу: https://ai.plainenglish.io/from-gaming-to-ai-how-gpus-are-driving-the-technology-till-artificial-intelligence-b5cc81759120.
2. Not just techies, but scientific community loves GPUs [Електронний ресурс] / Akashdeep Arul. – 2022. – Режим доступу до ресурсу: https://analyticsindiamag.com/not-just-techies-but-scientistic-community-loves-gpus.
3. Al-Lozi, E., Alfityani, A., Alsmadi, A., Al-hazimeh, A., Al-Gasawneh, J.: “The role of big data in financial sector”. International Journal of Data and Network Science, 2022.
4. Awrahman, B.J., Fatah, C.A., Hamaamin, M.Y.: “A Review of the Role and Challenges of Big Data in Healthcare Informatics and Analytics”. Computational Intelligence and Neuroscience, 2022.
5. Alqudah, N., Shatnawi, M.: “The Role of Big Data in Social Science: A Case Study Using Hadoop”. International Congress on Information and Communication Technology, 2022.
6. Schnase, J.L.: “Big Data Challenges in Climate Science: Improving the next-generation cyberinfrastructure”. IEEE Geoscience and Remote Sensing Magazine, 2016, vol. 4, no. 3, pp. 10-22.
7. Bernetti, M.; Bertazzo, M.; Masetti, M.: “Data-Driven Molecular Dynamics: A Multifaceted Challenge”. Pharmaceuticals, 2020.
8. Liu Y., Li, N., Zhu, X., Qi, Y.: “How wide is the application of genetic big data in biomedicine”. Biomedicine & Pharmacotherapy, 2021.

9. Teräs, M., Raghunathan S.: “Big Data Visualisation in Immersive Virtual Reality Environments: Embodied Phenomenological Perspectives to Interaction”. International Journal of Soft Computing, 2015.
10. Mehta, K., Jain, R., Mittal, P., Sharma, S.: “Cryptocurrency: A Critical Analysis of Embedded Big Data Analytics”. Proceedings of the International Conference on Innovative Computing & Communication (ICICC), 2022.
11. Hurst, W., Merabti M., Fergus, P.: “Big Data Analysis Techniques for Cyber-threat Detection in Critical Infrastructures”. 28th International Conference on Advanced Information Networking and Applications Workshops, 2014, pp. 916-921.
12. Oleksiienko, Illia and Iosifidis, Alexandros: “Analysis of voxel-based 3D object detection methods efficiency for real-time embedded systems – 2021 International Conference on Emerging Techniques in Computational Intelligence (ICETCI), 2021, pp. 59-64.
13. Kubernetes [Електронний ресурс] / The Kubernetes Authors. – 2024. – Режим доступу до ресурсу: https://kubernetes.io/.
14. Juanes, R., Haghighat, E.: “SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks”. Computer Methods in Applied Mechanics and Engineering, 2021.
15. Colab Enterprise documentation [Електронний ресурс] / Google LLC. – 2024. – Режим доступу до ресурсу: https://cloud.google.com/colab/docs.
16. Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y. M.: “Optimal Speed and Accuracy of Object Detection”. arXiv:2004.10934v1, 2020.
17. An introduction to the Unified Modeling Language [Електронний ресурс] / IBM. – 2024. – Режим доступу до ресурсу: https://developer.ibm.com/articles/an-introduction-to-uml/.
18. Introduction to Node.js [Електронний ресурс] / OpenJS Foundation. – 2024. – Режим доступу до ресурсу: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs.

19. Documentation on V8 [Електронний ресурс] / V8 Project. – 2024. – Режим доступу до ресурсу: https://v8.dev/docs.
20. Chatzimparmpas, A., Bibi, S., Zozas, I., Kerren, A.: “Analyzing the Evolution of Javascript Applications”. ENASE, 2019.
21. Windows Subsystem for Linux [Електронний ресурс] / Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/windows/wsl/.
22. Getting started with React [Електронний ресурс] / MDN contributors. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started.
23. What is a REST API? [Електронний ресурс] / IBM. – 2024. – Режим доступу до ресурсу: https://www.ibm.com/topics/rest-apis.
24. The WebSocket API (WebSockets) [Електронний ресурс] / MDN contributors. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
25. Getting Started with Redux [Електронний ресурс] / Redux documentation. – 2024. – Режим доступу до ресурсу: https://redux.js.org/introduction/getting-started.
26. Lee, C. C., Li, L. H., Hwang, M. S.: “A remote user authentication scheme using hash functions”. ACM SIGOPS Operating Systems Review, 2002.
27. Provos, N., & Mazieres, D.: “Bcrypt algorithm”. USENIX, 1999.
28. Runeson, P.: “A survey of unit testing practices”. IEEE Software, 2006, vol. 23, no. 4, pp. 22-29.
29. Jest: Delightful JavaScript testing [Електронний ресурс] / OpenJS Foundation. – 2024. – Режим доступу до ресурсу: https://jestjs.io/docs/getting-started.
30. Cheon, Y., & Leavens, G. T.: “A simple and practical approach to unit testing: The JML and JUnit way”. ECOOP 2002—Object-Oriented Programming: 16th European Conference Málaga, Spain, June 10–14, 2002 Proceedings 16, 2002.

31. Vilkomir, S., Marszalkowski, K., Perry, C., & Mahendrakar, S.: “Effectiveness of multi-device testing mobile applications”. 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, 2015.
32. Technology Readiness Levels [Електронний ресурс] / National Aeronautics and Space Administration. – 2023. – Режим доступу до ресурсу: https://www.nasa.gov/directorates/somd/space-communications-navigation-program/technology-readiness-levels/.
33. Mankins, J. C.: “Technology readiness levels”. White Paper, April, 6(1995), 1995.
34. Wang, R., Yu, B., Marco, J., Hu, T., Gutierrez, D., & Bao, H.: “Real-time rendering on a power budget”. ACM Transactions on Graphics (TOG), 2016.

ДОДАТОК 1

Акт запровадження

Approved by:
Postdoctoral Researcher
Aarhus University
Illia Oleksiienko
26 november 2024.

ACT

on the use of the results of the dissertation
by Nikita Lazarenko on the topic “Multi-user GPU task manager for distributed systems”, submitted for a master's degree in the scientific and educational programme
“Computer Systems Software Engineering”

I, the undersigned, the postdoctoral researcher Illia Oleksiienko, have drawn up this certificate that the results of the dissertation work of the student Nikita Lazarenko on “Multi-user GPU task manager for distributed systems” were and are being used in the Aarhus University to perform machine learning experiments, used in project “Deep Learning for Financial Investor Network Analysis” – granted by Independent Research Fund Denmark.

The proposed tools have allowed for faster training of models and faster resolution of booking conflicts, while retaining full freedom over the execution of the tasks.

Postdoctoral Researcher Illia Oleksiienko

Переклад акту запровадження

ЗАТВЕРДЖУЮ
постдокторський дослідник
Орхуський університет
Illia Oleksiienko
26 листопада 2024.

АКТ

про використання результатів дисертаційної роботи
Лазаренка Нікіти В'ячеславовича на тему "Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі", поданої на здобуття ступеня магістра за освітньо-професійною програмою
«Інженерія програмного забезпечення комп’ютерних систем»

Я, що нижче підписався, постдокторський дослідник Illia Oleksiienko. склав цей акт про те, що результати дисертаційної роботи студента Лазаренка Нікіти В'ячеславовича на тему "Багатокористувацький менеджер задач графічного процесора для запуску програмного коду на розподіленій системі" використані й використовуються в Орхуському університеті при проведенні експериментів з машинного навчання, що були використані в межах проєкту "Deep Learning for Financial Investor Network Analysis", що фінансується Незалежним дослідницьким фондом Данії.

Запропоновані інструменти дозволили прискорити навчання моделей і швидше вирішувати конфлікти бронювання, зберігаючи при цьому повну свободу над виконанням завдань.

постдокторський дослідник Illia Oleksiienko

Частина програмного коду

```
import {argv} from 'process';

const DEBUG = argv [2] === 'debug';

if (DEBUG) {
   console.log({DEBUG});
}

import debugSettings  from './debug-settings.json' assert { type: 'json' };
import commonSettings from './settings.json' assert { type: 'json' };

const settings = DEBUG ? debugSettings : commonSettings;
import Task from './task.js';
import fs from 'fs';
import {randomElement, shuffleArray, remove} from './core.js';

export const workstations = [];
export const connections = {};
let state = {
   queue: [],
   processingTasks: [],
   finishedTasks: [],
   lastTaskId: 0,
};
const taskIdLoop = 260526052605;
const stateFileName = DEBUG ? './debug.state.json' : './state.json';

export const isGoodAccessLevel = (key, accessLevel) => {
   if (accessLevel >= settings.accessLevels [key]) {
      return true;
   } else {
         console.error(`Access level must be ${settings.accessLevels [key]} or higher (current is
${accessLevel})`);
   }
};

const loadState = () => {
   if (fs.existsSync(stateFileName)) {
      state = JSON.parse(fs.readFileSync(stateFileName));
      state.queue.forEach(a => Object.setPrototypeOf(a, Task.prototype));
      state.processingTasks.forEach(a => Object.setPrototypeOf(a, Task.prototype));
      state.finishedTasks.forEach(a => Object.setPrototypeOf(a, Task.prototype));
```

```
    }
};

const saveState = () => {
    fs.writeFileSync(stateFileName, JSON.stringify(state));
};

loadState();

const nextTaskId = () => {
    state.lastTaskId = (state.lastTaskId + 1) % taskIdLoop;
    return state.lastTaskId;
};

/**
 * @param {Task} task
 * @returns {null | [ComputationalUnit]}
 */
const allocateResources =(task) => {
    const selectedWorkstations = task.workstaton === 'any' ?
        workstations : workstations.filter(a => task.workstaton.includes(a.name));

    if (task.gpus <= 0) {
        const freeCpus =[].concat(
            ...selectedWorkstations.map(a => a.cpus.filter(a => a.isFree()))
        );

        if (freeCpus.length <= 0) {
            return null;
        } else {
            return[randomElement(freeCpus)];
        }
    } else {
        const freeGpus =[].concat(
            ...selectedWorkstations.map(a => a.gpus.shared.filter(
                a => a.isFree () && a.memory >= task.minimalGpuMemory
            ))
        );

        if (freeGpus.length < task.gpus) {
            return null;
        } else {
            shuffleArray(freeGpus);
            return freeGpus.slice(0, task.gpus);
        }
    }
```

```
};

export let processQueue = () => {
    state.queue.sort ((a, b) => b.priority - a.priority);

    for (let i = 0; i < state.queue.length; i ++) {
        const task = state.queue [i];
        const resources = allocateResources(task);

        if (resources === null) {
            continue;
        } else {
            state.queue.splice(i, 1);
            i --;

            resources.forEach (a => a.registerProcess(task));

            const workstatonName = resources[0].workstation;
            const connection = connections[workstatonName];

            state.processingTasks.push(task);

            saveState();

            // eslint-disable-next-line no-loop-func
            connection.sendTask(task, resources, (result, isError) => {
                remove(state.processingTasks, task);
                state.finishedTasks.push (task);
                saveState();

                resources.forEach (a => a.unregisterProcess(task));

                processQueue();
            });
        }
    }
};

export const addTask = (user, workstaton, gpus, minimalGpuMemory, name, script, workdingDir,
saveScreen, logScreen, priorityValue) => {
    const task = new Task (
        nextTaskId (),
        user, workstaton, gpus, minimalGpuMemory, name, script, workdingDir, saveScreen, logScreen,
        priorityValue
    );
```

```
    state.queue.push (
        task
    );

    processQueue ();

    saveState ();

    return task.toString ();
};

export const showQueue = (user = null, log = console.log) => {
    const selectedQueue = user === null ?
        state.queue : state.queue.filter (task => task.user === user);

    const selectedProcessingTasks = user === null ?
        state.processingTasks : state.processingTasks.filter (task => task.user === user);

    const process = queue => queue.map (
        task => task.toString () + (user === null ? '' : ' script: ' + task.script)
    ).join ('\n');

    const text = 'Queue:\n'
        + process (selectedQueue) + '\n\n'
        + 'Processing tasks:\n'
        + process (selectedProcessingTasks) + '\n\n';

    log (text);
};

export const showFinishedTasks = (user, log = console.log) => {
    const selectedFinishedTasks = state.finishedTasks.filter (task => task.user === user);

    const process = queue => queue.map (
        task => task.toString () + (user === null ? '' : ' script: ' + task.script)
    ).join ('\n');

    const text = 'Finished tasks:\n'
        + process (selectedFinishedTasks) + '\n\n';

    log (text);
};

export const clearFinishedTasks = (user, log) => {
    state.finishedTasks = state.finishedTasks.filter (task => task.user !== user);
```

```
    if (log) log ('Finished tasks cleared');
};

export const clearAllTasks = (user, log) => {
    state.finishedTasks = state.finishedTasks.filter (task => task.user !== user);
    state.queue = state.queue.filter (task => task.user !== user);
    state.processingTasks = state.processingTasks.filter (task => task.user !== user);

    if (log) log ('All tasks cleared');
};

export default state;

import {login, checkToken, getAccessLevel, createUser, setPassword, setAccessLevel, deleteUser}
from './database.js';
import express from 'express';
import {isGoodAccessLevel, addTask, showQueue, showFinishedTasks, clearFinishedTasks,
clearAllTasks} from './server-core.js';
import state from './server-core.js';

const router = express.Router();

router.post('/login', (req, res) => {
    const {username, password, keyWord} = req.body;

    try {
        login (username, password, keyWord)
        .then(result => {
            res.json({ result: `success: ${JSON.stringify(result.isSuccess)}`, error: result.error, token:
result.token });
        });
    }
    catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/addTask', async (req, res) => {
    try {
        const { workstaton, gpus, minimalGpuMemory, name, script, workdingDir, saveScreen,
logScreen, priorityValue, token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (checkToken(token) && isGoodAccessLevel('addTask', getAccessLevel(username))) {
            addTask (username, workstaton, gpus, minimalGpuMemory, name, script, workdingDir,
saveScreen, logScreen, priorityValue);
```

```
            res.json({ result: 'Task added', error: null });
        } else {
            res.status(403).json({ error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/showQueue', async (req, res) => {
    try {
        const { token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (tokenStatus.success && isGoodAccessLevel('showQueue', getAccessLevel(username))) {
            const outputs = [];
            const log = (data) => {
                outputs.push(data);
            }
            showQueue(username, log);
            res.json({ result: outputs, error: null });
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/showFinishedTasks', async (req, res) => {
    try {
        const { token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
                        if (tokenStatus.success && isGoodAccessLevel('showFinishedTasks',
getAccessLevel(username))) {
            const outputs = [];
            const log = (data) => {
                outputs.push(data);
            }
            showFinishedTasks(username, log);
            res.json({ result: outputs, error: null });
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
```

```
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/clearFinishedTasks', async (req, res) => {
    try {
        const { token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
                        if (tokenStatus.success && isGoodAccessLevel('clearFinishedTasks',
getAccessLevel(username))) {
            clearFinishedTasks(username);
            res.json({ result: 'Finished tasks cleared', error: null });
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/clearAllTasks', async (req, res) => {
    try {
        const { token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (tokenStatus.success && isGoodAccessLevel('clearAllTasks', getAccessLevel(username))) {
            clearAllTasks(username);
            res.json({ result: 'All tasks cleared', error: null });
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/repeatTask', async (req, res) => {
    try {
        const { taskId, token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (tokenStatus.success && isGoodAccessLevel('addTask', getAccessLevel(username))) {
            const task = state.finishedTasks.find (a => a.taskId === taskId && a.user === username);
            if (task) {
                addTask (
```

```
                task.user, task.workstaton, task.gpus, task.minimalGpuMemory,
                task.name, task.script, task.workdingDir, task.saveScreen, task.logScreen
            );
            res.json({ result: 'Task repeated', error: null });
        } else {
            res.status(400).json({ result: null, error: `No finished task with id ${taskId} for user
${username}` });
        }
    } else {
        res.status(403).json({ result: null, error: 'Access denied' });
    }
  } catch (error) {
    res.status(400).json({ error: error.toString() });
  }
});

router.post('/setPassword', async (req, res) => {
  try {
    const { password, token } = req.body;
    const tokenStatus = await checkToken(token);
    const username = tokenStatus.username.login;
    if (tokenStatus.success && isGoodAccessLevel('setPassword', getAccessLevel(username))) {
      setPassword (username, password)
        .then (result => {
          if (result === true) {
            res.json({ result: 'Changed password for: ' + username, error: null });
          } else {
            res.status(400).json({ result: null, error: result });
          }
        });
    } else {
      res.status(403).json({ result: null, error: 'Access denied' });
    }
  } catch (error) {
    res.status(400).json({ error: error.toString() });
  }
});

router.post('/setAccessLevel', async (req, res) => {
  try {
    const { targetUsername, newAccessLevel, token } = req.body;
    const tokenStatus = await checkToken(token);
    const username = tokenStatus.username.login;
    if (tokenStatus.success && isGoodAccessLevel('setAccessLevel', getAccessLevel(username)))
{
      const result = setAccessLevel (targetUsername, newAccessLevel);
```

```
            if (result === true) {
                res.json({ result: 'Updated accessLevel for: ' + targetUsername, error: null });
            } else {
                res.status(400).json({ result: null, error: result });
            }
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/createUser', async (req, res) => {
    try {
        const { targetUsername, password, accessLevel, token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (tokenStatus.success && isGoodAccessLevel('createUser', getAccessLevel(username))) {
            createUser (targetUsername, password, accessLevel)
                .then (result => {
                    if (result === true) {
                        res.json({ result: 'Created user: ' + targetUsername, error: null });
                    } else {
                        res.status(400).json({ result: null, error: result });
                    }
                });
        } else {
            res.status(403).json({ result: null, error: 'Access denied' });
        }
    } catch (error) {
        res.status(400).json({ error: error.toString() });
    }
});

router.post('/deleteUser', async (req, res) => {
    try {
        const { targetUsername, token } = req.body;
        const tokenStatus = await checkToken(token);
        const username = tokenStatus.username.login;
        if (tokenStatus.success && isGoodAccessLevel('deleteUser', getAccessLevel(username))) {
            const result = deleteUser (targetUsername);
            if (result === true) {
                res.json({ result: 'Deleted user: ' + targetUsername, error: null });
            } else {
                res.status(400).json({ result: null, error: result });
```

```
        }
      } else {
        res.status(403).json({ result: null, error: 'Access denied' });
      }
    } catch (error) {
      res.status(400).json({ error: error.toString() });
    }
});

router.post('/workstations', (req, res) => {
    res.json(workstations.map (a => ({
      name: a.name,
      'gpus-shared': a.gpus.shared.map (
            gpu => `id: ${gpu.id}, memory: ${gpu.memory}, processes: ${gpu.processes.length}, maxProcesses: ${gpu.maxProcesses}`
      ),
      'gpus-personal': a.gpus.personal.map (
            gpu => `id: ${gpu.id}, memory: ${gpu.memory}, processes: ${gpu.processes.length}, maxProcesses: ${gpu.maxProcesses}`
      ),
       cpus: a.cpus.map (cpu => `id: ${cpu.id}, processes: ${cpu.processes.length}, maxProcesses: ${cpu.maxProcesses}`),
    })));
});

router.post('/help', async (req, res) => {
    try {
      const { token } = req.body;
      const tokenStatus = await checkToken(token);
      const username = tokenStatus.username.login;
                  if (tokenStatus.success && isGoodAccessLevel('clearFinishedTasks', getAccessLevel(username))) {
        res.json({
          result: `
            login: login (username, password)
            logout: logout ()
                  addTask task: addTask (workstaton, gpus, minimalGpuMemory, name, script, workdingDir, saveScreen = true, logScreen = true)
            show queue: showQueue (self = true)
            set password: setPassword (password)
            createUser: createUser (username, password, accessLevel = 1)
            set access level: setAccessLevel (username, newAccessLevel)
            show current workstations: workstations ()
            show finished tasks: showFinishedTasks ()
            clear finished tasks: clearFinishedTasks ()
            repeat task: repeatTask (taskId)
```

```
                clear all your tasks: clearAllTasks ()`,
            error: null
        });
    } else {
        res.status(403).json({ result: null, error: 'Access denied' });
    }
} catch (error) {
    res.status(400).json({ error: error.toString() });
}
});

export default router;

import fs from 'fs';
import path from 'path';
import chalk from 'chalk';

const testDir = path.resolve('./tests');

const runTestFile = async (filePath) => {
    console.log(chalk.blue(`\n\nRunning test file: ${filePath}`));
    const testModule = await import(filePath);
    await testModule.default();
};

const runAllTests = async () => {
    const testFiles = fs.readdirSync(testDir).filter(file => {
        return file.endsWith('.js');
    });

    for (const file of testFiles) {
        await runTestFile(path.join(testDir, file));
    }
    console.log('All tests completed.');
};

runAllTests().catch(error => {
    console.error('Error running tests:', error);
});

import fetch from 'node-fetch';
import assert from 'assert';
import chalk from 'chalk';
import database from '../database.js';

const runHelpTests = async () => {
```

```
    try {
        console.log('Running failure test for /help endpoint (unexpected data)');
        const response = await fetch('http://127.0.0.1:2604/help', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ token: 'invalid_token' })
        });

        const data = await response.json();
        assert.strictEqual(data.error, null);
        console.log(chalk.green('Failure test passed unexpectedly!'));
    } catch (error) {
        console.error(chalk.red('Failure test correctly failed for /help endpoint:', error.message));
    }

    try {
        console.log('Running success test for /help endpoint');
        const response = await fetch('http://127.0.0.1:2604/help', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ token: database.users[0].token })
        });

        const data = await response.json();

        if (data.error) {
            console.error(chalk.red('Error executing code:', data.error));
        } else {
            console.log(chalk.green('Execution result:', data.result));
            console.log(chalk.green('Test passed for /help endpoint'));
        }
    } catch (error) {
        console.error(chalk.red('Request failed:', error));
    }
};

export default runHelpTests;
```