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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.043

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютерних та інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмне забезпечення брокера хмарних послуг різних провайдерів»

Виконав:
студент ІІ курсу, групи ІП-21мн
Мальцев Олександр Сергійович

Керівник:
доктор технічних наук, професор
Жаріков Едуард Вячеславович

Рецензент:
Доцент кафедри ІСТ, кандидат технічних наук
Шимкович Володимир Миколайович

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент Мальцев Олександр Сергійович

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення комп'ютерних та інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2024 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Мальцеву Олександру Сергійовичу

1. Тема дисертації «Програмне забезпечення брокера хмарних послуг різних провайдерів», науковий керівник дисертації Жаріков Едуард Вячеславович, д.т.н., професор кафедри інформатики та програмної інженерії, затверджені наказом по університету від «23» березня 2024 р. № 1445-с

2. Термін подання студентом дисертації «9» червня 2024 р.

3. Об'єкт дослідження – вибір сервісів провайдерів хмарних послуг та їх налаштування для заданого проєкту.

4. Предмет дослідження – підходи, технології та програмне забезпечення брокеру вибору сервісів провайдерів хмарних послуг.

5. Перелік завдань, які потрібно розробити – проаналізувати пропозиції послуг на ринку хмарних обчислень; проаналізувати алгоритми багатокритеріальної оптимізації; розробити методику вибору провайдера хмарних послуг та вибору налаштувань обраних сервісів; розробити програмний продукт, що реалізує брокер вибору сервісів провайдерів хмарних послуг; провести експериментальні дослідження, що доводять ефективність запропонованого рішення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 1 плакат

7. Орієнтовний перелік публікацій – одна публікація

8.Дата видачі завдання «5» лютого 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Отримання завдання	05.02.2023	
2	Огляд літератури	04.03.2023	
3	Аналіз пропозицій послуг на ринку хмарних обчислень	12.06.2023	
4	Аналіз алгоритмів багатокритеріальної оптимізації	24.09.2023	
5	Розробка методики вибору провайдера хмарних послуг та вибору налаштувань обраних сервісів	02.11.2023	
6	Розробка програмного продукту	15.01.2024	
7	Проведення експериментальних	17.03.2024	

	досліджень		
8	Оформлення пояснювальної записки	10.04.2024	
9	Подання дисертації на попередній захист	19.05.2024	
10	Подання дисертації на захист	09.06.2024	

Студент Віктор КРИШТАЛЬ

Науковий керівник Едуард ЖАРІКОВ

РЕФЕРАТ

Розмір пояснювальної записки – 55 аркушів, містить 14 ілюстрацій, 24 таблиць, 2 додатків, 31 посилань на джерела.

Актуальність теми. В наш час збільшується потреба у проведенні хмарних обчисленнь, адже вони значно спрощують інфраструктуру та зменшують затрати на утримання власних серверів та їх підтримку. Наразі існує багато компаній, які надають послуги хмарних обчисленнь, вони пропонують розробнику різноманіття можливих конфігурацій середи з різними характеристиками швидкодії та вартістю. Вибір хмарного провайдера та оптимальної конфігурації для конкретної задачі є критично важливим, адже саме вони визначають швидкодію обчислень та кількість витрачених коштів. Розробка брокеру хмарних послуг дозволить обрати провайдера та відповідну конфігурацію сервісів, тому дана тема є актуальною.

Мета дослідження. Зменшення витрат на використання хмарних послуг у спосіб розроблення програмного забезпечення брокера хмарних послуг, що дозволяє вибрати провайдера хмарних послуг для заданого проєкту або задачі за встановленими вимогами та надає власний інтерфейс взаємодії з користувачем для замовленної послуги.

Об’єкт дослідження: вибір сервісів провайдерів хмарних послуг та їх налаштування для заданого проєкту.

Предмет дослідження: підходи, технології та програмне забезпечення брокеру вибору сервісів провайдерів хмарних послуг.

Для реалізації поставленої мети **сформульовані такі завдання**:

- Проаналізувати пропозиції послуг на ринку хмарних обчислень.
- Проаналізувати алгоритми багатокритеріальної оптимізації.
- Розробити методику вибору провайдера хмарних послуг та вибору налаштувань обраних сервісів;

- Розробити програмний продукт, що реалізує брокер вибору сервісів провайдерів хмарних послуг.
- Провести експериментальні дослідження, що доводять ефективність запропонованого рішення.

Наукова новизна результатів магістерської дисертації полягає у розробленні методики вибору визначеного сервісу, який може надаватися різними провайдерами хмарних послуг, та його налаштувань, що дозволяє більш ефективно виконувати хмарні обчислення.

Практичне значення отриманих результатів полягає в тому, що реалізований програмний продукт дозволяє користувачу замовляти хмарну функцію у провайдерів з вибором конфігурації, що найбільше підходить його потребам. Зв’язок з науковими програмами, планами, темами.

Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на VI Міжнародній науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024» – м. Київ.

Публікації. Наукові положення дисертації опубліковані в:

1) Мальцев О.С.. Програмне забезпечення брокеру хмарних послуг // Матеріали науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024» – м. Київ: КПІ ім. Ігоря Сікорського, 21-23 травня 2024 р.

Ключові слова: БРОКЕР, ХМАРНІ ПОСЛУГИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

The explanatory note comprises 55 pages and contains 14 illustrations, 24 tables, 2 appendices, and 31 references.

Relevance of the topic. Currently, the demand for cloud computing is increasing as it significantly simplifies infrastructure and reduces the costs associated with maintaining and supporting own servers. There are numerous companies offering cloud computing services, providing developers with a variety of possible configurations of environments with different performance characteristics and costs. The choice of a cloud provider and the optimal configuration for a specific task is critically important as it determines the performance of computations and the amount of money spent. A cloud services broker helps to choose the provider and optimal configuration, making this topic relevant.

The aim of the study. The target of the study is to develop software that selects a cloud services provider for the chosen service based on a range of parameters and provides a user interface for the ordered service.

The object of research: selection of a cloud provider and its configuration for computing a cloud function.

The subject of research: software for brokering cloud services from various providers.

To achieve the stated goal, the **following tasks** were formulated:

- Analyze the cloud computing market.
- Analyze multi-criteria optimization algorithms.
- Develop a methodology for selecting a cloud services provider and the configuration of the cloud computing environment.
- Develop a software product that implements the described methodology in practice.
- Conduct experimental studies to demonstrate the effectiveness of the proposed solution.

The scientific novelty of the results of the master's dessertation is a methodology for selecting a cloud services provider and the configuration of the cloud computing environment has been proposed, allowing for more efficient cloud computations.

The practical value of the obtained results lies in the implemented software product, which enables users to order a cloud function from providers with a configuration that best suits their needs.

Relationship with working with scientific programs, plans, topics. Work was carried out at the Department of Informatics and Software Engineering of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

Approbation. The scientific positions of the dissertation were tested at the VI International Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies SoftTech-2024" – Kyiv.

Publications. The scientific positions of the dissertation are published in:

1) Maltsev O.S. Software for Cloud Services Brokerage // Proceedings of the Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies SoftTech-2024" – Kyiv: KPI named after Igor Sikorsky, May 21-23, 2024.

Keywords: BROKER, CLOUD SERVICES, SOFTWARE.

ЗМІСТ

ВСТУП

Останнім часом хмарні обчислення стали новою парадигмою для розміщення та надання послуг через Інтернет. Хмарні обчислення привертають увагу власників бізнесу, оскільки вони зменшують капітальні витрати та усувають потребу користувачів у заздалегідь запланованому використанні ресурсів, що дозволяє підприємствам розпочати з невеликої IT-інфраструктури та збільшувати ресурси лише у випадку збільшення попиту на послуги. Однак, незважаючи на те, що хмарні обчислення відкривають широкі можливості для IT-індустрії, розвиток технології хмарних обчислень наразі перебуває в своїх початках, і є багато проблем, які ще потрібно вирішити.

Хмарні обчислення надають великий спектр архітектурних рішень та конфігурацій сервісів, наприклад кількість ядер, обсяг пам'яті, кількість запитів, час виконання, кількість вузлів та інші. Продуктивність робочого навантаження (застосунку) та його введення при одній архітектурі може виконуватися до 20 разів довше або коштувати у 10 разів більше чим при застосуванні іншого варіанту архітектури. Готова гнучкість в хмарних послугах призвела до зміни парадигми. Порівняно з тим, як раніше додаток налаштовувався для визначеного кластера, в хмарі архітектурна конфігурація (рішення) налаштовується під робоче навантаження. Крім того, оскільки хмара має модель сплати за фактом використання, кожна конфігурація (розмір кластера, тип віртуальної машини та ін.) має витрати та час виконання. Таким чином, робоче навантаження може бути оптимізоване з точки зору найменших витрат або найкоротшого часу, що представляє собою різні конфігурації або архітектурні рішення. Вибір вірної хмарної конфігурації для додатка є ключовим для якості обслуговування та комерційної конкурентоспроможності. Наприклад, неоптимальна хмарна конфігурація може призвести до витрат, що в 12 разів перевищують витрати для того ж самого цільового рівня продуктивності. Вибір належної хмарної конфігурації є ще більш суттєвим для повторюваних завдань [5], [9], у яких подібні робочі навантаження виконуються систематично. Тим не менш, вибір

найкращої хмарної конфігурації, наприклад, найдешевшої або найшвидшої, ускладнений через складність одночасного досягнення високої точності, низької накладеної вартості та адаптивності для різних додатків та робочих навантажень. Тому задача вибору хмарного провайдеру для заданого сервісу та вибору його конфігурації є актуальною.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Хмарні обчислення

Хмарні обчислення є моделлю, яка надає можливість всюди, зручно та за вимогою отримувати доступ через мережу до налаштувань та спільного пулу конфігуруючих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, додатків та послуг), які можуть бути швидко надані та звільнені з мінімальними зусиллями управління або взаємодії із постачальником послуг [13]. Ця модель хмари складається з п'яти основних характеристик, трьох моделей надання та чотирьох моделей розгортання.

Визначення NIST виокремлює п'ять основних характеристик хмарних обчислень:

- Самообслуговування на вимогу,
- доступність хмарних сервісів через мережу з персональних комп’ютерів, ноутбуків та мобільних телефонів,
- пул ресурсів, з якого виділяється частка для окремих користувачів та додатків,
- швидка еластичність виділення ресурсів, які можуть бути швидко перерозподілені за необхідності,
- вимірюване обслуговування та моніторинг.

При наявності усіх цих характеристик створюється інфраструктура хмарних обчислень, яка містить як фізичний, так і віртуальний шари. Фізичний шар складається з апаратних ресурсів, які підтримують хмарні послуги (тобто сервери, сховище даних та компоненти мережі). Віртуальний або програмно-визначений шар включає програмне забезпечення, розгорнуте через фізичний шар, тим самим виражаючи основні характеристики хмари згідно із визначенням NIST.

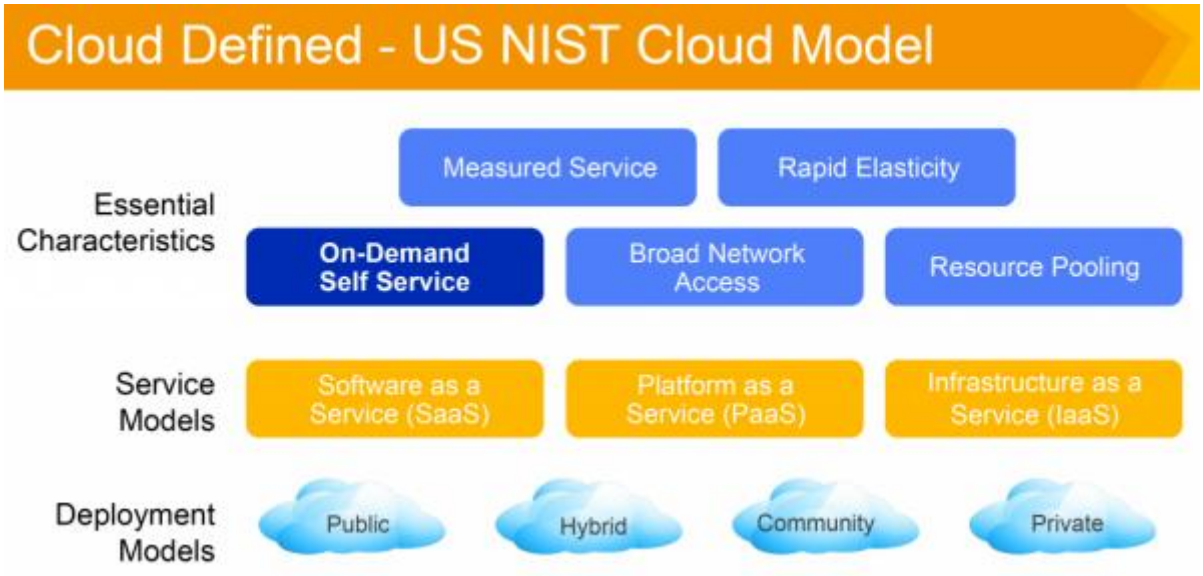


Рис. 1.1. Складові хмари за визначенням NIST

Самообслуговування за вимогою. Користувач може у односторонньому порядку надавати обчислювальні можливості, такі як час сервера та мережеве сховище, як необхідно автоматично без потреби у взаємодії з кожним постачальником послуг.

Широкий доступ до мережі. Можливості доступні через мережу та використовуються за допомогою стандартних механізмів, які сприяють використанню різнорідними платформами клієнтів (тонкими або товстими), такими як мобільні телефони, планшети, ноутбуки та робочі станції.

Пул ресурсів. Обчислювальні ресурси постачальника об'єднуються для обслуговування кількох користувачів за допомогою моделі багатокористувацької системи з різними фізичними та віртуальними ресурсами, які динамічно призначаються та перерозподіляються відповідно до запиту користувача. Існує відчуття незалежності від місця, оскільки зазвичай користувач не має контролю або знань щодо точного місцезнаходження наданих ресурсів, але може вказати місце на вищому рівні абстракції (наприклад, країна, штат або датацентр). Приклади ресурсів включають сховища, обробку, пам'ять та мережеву пропускну здатність.

Швидка еластичність. Можливості можуть бути еластично надані та звільнені, у деяких випадках автоматично, щоб масштабуватися швидко залежно від попиту. Для користувача доступні можливості для надання часто здається нескінченними та можуть бути використані у будь-якій кількості в будь-який час.

Вимірювання послуги. Хмарні системи автоматично контролюють та оптимізують використання ресурсів, використовуючи можливість вимірювання на певному рівні абстракції, відповідно до типу послуги (наприклад, сховище, обробка, пропускна здатність мережі та активні облікові записи користувачів). Використання ресурсів може бути контрольоване, моніторинговане та звітуване, що забезпечує прозорість як для постачальника, так і для споживача використаної послуги.

За даними Amazon [1], хмарні технології призводять до семи змін у розробці, побудові та експлуатації додатків:

- Хмара спрощує розподілені архітектури.
- Хмарні технології дозволяють користувачам скористатися перевагами безпеки спільних систем.
- Хмара дозволяє підприємствам переходити від масштабування за архітектурою до масштабування за командою.
- Хмара дозволяє користувачам частіше експериментувати.
- Хмарні технології дозволяють використовувати великі обсяги даних без потреби у великих серверах.

1.2 Сервісні моделі хмарних технологій

Інфраструктура як послуга (IaaS). Можливість, що надається споживачеві, полягає в наданні серверних та безсерверних обчислень, сховищ, мереж та інших фундаментальних обчислювальних ресурсів, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати

операційні системи та додатки. Споживач не керує базовою інфраструктурою хмари, але має контроль над операційними системами, сховищами та розгортаними додатками. І, можливо, обмежений контроль окремих компонентів мережі.

Основні переваги:

- Менші витрати, оскільки всі витрати на обслуговування несе постачальник.
- Дані знаходяться в центрі обробки даних постачальника, де вони захищені фізичними та цифровими системами безпеки.
- Просте масштабування.

Інфраструктура як послуга є базою ресурсів для послуг компанії. Постачальники Інфраструктури як послуги пропонують менше послуг, ніж у інших моделях. Залежно від потреб компанії це може бути перевагою. Якщо у компанії є власна стратегія щодо розробки IT-рішень, ця модель надасть їй найбільшу свободу для впровадження цієї стратегії. Інфраструктура як послуга використовується для розгортання окремих додатків, а також всього набору IT-інструментів компанії: ERP-систем, термінальних серверів, віддалених офісів, баз даних і т. д.

Платформа як послуга (PaaS). Можливість, що надається споживачеві, полягає в розгортанні на інфраструктурі хмари створених або придбаних споживачем додатків, створених за допомогою мов програмування, бібліотек, сервісів та інструментів, підтриманих постачальником. Споживач не керує або не контролює базову інфраструктуру хмари, включаючи мережу, сервери, операційні системи або сховища, але має контроль над розгортаними додатками та можливо налаштуваннями конфігурації середовища для хостингу додатків.

Основні переваги Платформи як послуги:

- Постачальники PaaS відповідають за будь-які оновлення та виправлення програмного забезпечення.

- Клієнт не повинен інвестувати у апаратне забезпечення або програмне забезпечення - постачальник пропонує повну платформу для оренди
- гнучкість під час розгортання: інфраструктура повністю контролюється постачальником.

PaaS схожа на IaaS за своїм призначенням, однак клієнти PaaS не можуть масштабувати інфраструктуру. Це означає, що користувач не може вимкнути зайві ресурси або додати нові. Відмінність між IaaS та PaaS полягає в тому, що Платформа як послуга надає обчислювальну платформу та набір рішень, але користувач не може налаштувати конфігурацію. Стандартизовані набори розробки є вагомою причиною вибору PaaS. Це може значно підвищити ефективність команди, яка працює над одним проектом.

Приклади послуг згідно моделі PaaS: Google App Engine, VMWare Cloud Foundry, IBM Bluemix та інші.

Програмне забезпечення як послуга (SaaS). Можливість, що надається споживачеві, полягає в користуванні додатками постачальника, які працюють у інфраструктурі хмари. Додатки доступні з різних пристроїв клієнтів через тонкий клієнтський інтерфейс, такий як веб-браузер (наприклад, веб-пошта), або програмний інтерфейс. Споживач не керує базову інфраструктуру хмари, включаючи мережу, сервери, операційні системи, сховища, або навіть окремі можливості програм, за винятком можливих обмежених налаштувань конфігурації додатка для користувача.

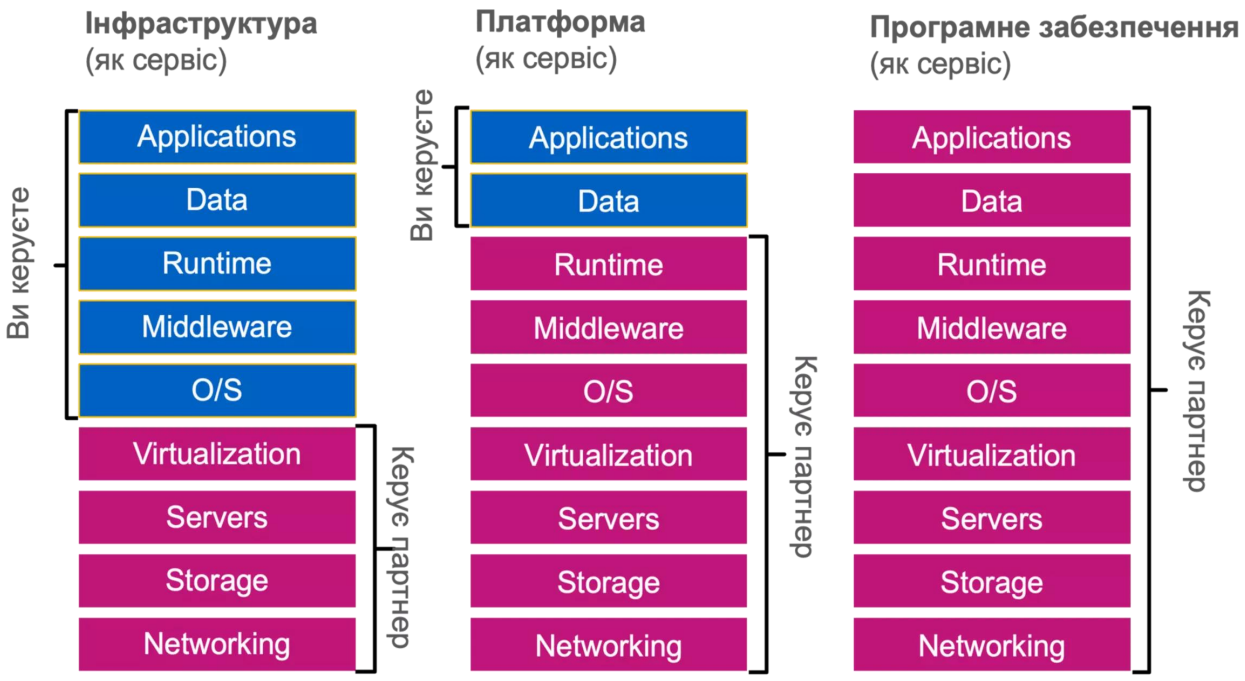


Рис 1.2. Сервісні моделі хмарних технологій.

Функція як сервіс (FaaS). Модель обчислювання без серверів. Це модель, яка дозволяє користувачам виконувати свій код, не маючи потреби в налаштуванні, управлінні або масштабуванні серверів.

FaaS також відомий як "платіть-за-використання", оскільки користувач платить лише за те, що використовує. Ця модель хмарних обчислень була розроблена спочатку PiCloud, яка випустила першу версію своєї пропозиції в 2010 році після того, як Amazon випустила AWS Lambda в 2014 році.

За даними [6] FaaS є одним з найпотужніших інструментів на ринку і стрімко зростає. Фактично, його використання збільшилося на 25% з 2017 року. Це означає, що FaaS і надалі буде важливою частиною галузі на протязі наступних років. Існує кілька причин, чому FaaS стає все більш популярним і широко використовуваним:

- Ця модель надає розробникам можливість зосередитися на своєму коді, не турбуючись про інфраструктуру.

- Вона дозволяє використовувати будь-яку мову програмування для створення програмного забезпечення, і немає обмежень у масштабуванні.
- FaaS дозволяє розробникам тестувати та розгортати нові функції всього за декілька секунд, не турбуючись про сервери та обслуговування інфраструктури.

В результаті FaaS виявився надзвичайно цінним технологічним рішенням як для розробників, так і для бізнесу. Його використовують великі компанії - такі як Google, Netflix, Amazon та Salesforce, а також багато менших стартапів, яким потрібен безпечний спосіб хостингу їх додатків, і він допомагає їм всім заощаджувати час і гроші у процесі.

FaaS приховує всю інфраструктуру та точки інтеграції для користувача та надає середовище виконання для запуску коду. Це схоже на PaaS, але з іним акцентом: замість надання всіх інфраструктурних та точок інтеграції для користувача, FaaS керує всією інфраструктурою та точками інтеграції за нього, щоб йому не довелося турбуватися про налаштування серверів чи створення цілої хмарної інфраструктури з нуля.

FaaS не обмежується лише виконанням коду, його також можна використовувати як альтернативний спосіб розробки додатків взагалі. За допомогою технології FaaS розробники можуть створювати свої власні користувацькі функції. Ці функції потім розгортаються як "безсерверний" код, який запускається в хмарних службах. Це означає, що розробники мають засіб для створення додатків, не турбуючись про створення власних серверів або управління інфраструктурою. З технологією FaaS розробники можуть зосередитися на написанні коду і легко розгортати його, що означає менше часу, витраченого на налаштування серверів і більше часу, витраченого на вирішення бізнес-завдань.

Незважаючи на багаточисельні переваги, модель FaaS має також і недоліки. Найважливішим з них є те, що розробники повинні бути дуже

кваліфіковані у вибраній мові, щоб використовувати її ефективно. Ще одним недоліком FaaS є те, що він може бути дорогим у випадку, якщо користувач використовує його надто багато. Кожен виклик коштує гроші, і це швидко накопичується, якщо користувач запускає багато функцій у своєму додатку - FaaS може не дуже добре працювати з підприємницькими додатками взагалі (або з будь-якими додатками, які потребують підтримки високого навантаження протягом тривалих періодів).

1.3 Моделі розгортання додатків

Приватний хмарний сервіс. Інфраструктура хмари надається для виключного використання однією організацією, яка складається з кількох споживачів (наприклад, бізнес-підрозділів). Вона може бути власністю, керована та операційною діяльністю організації, третьої сторони або комбінацією їх, і може існувати на власних або зовнішніх площадках.

Спільна хмара. Інфраструктура хмари надається для виключного використання конкретною спільнотою споживачів з організацій, які мають спільні інтереси (наприклад, місію, вимоги до безпеки, політику та врахування вимог щодо відповідності). Вона може бути власністю, керована та операційною діяльністю однієї або кількох організацій у спільноті, третьої сторони або комбінацією їх, і може існувати на власних або зовнішніх площадках.

Публічний хмарний сервіс. Інфраструктура хмари надається для відкритого використання загальною громадськістю. Вона може бути власністю, керована та операційною діяльністю бізнесу, навчального закладу або урядової організації, або комбінацією їх. Вона існує на площадках хмарного провайдера.

Гібридний хмарний сервіс. Інфраструктура хмари є складовою двох чи більше відмінних хмарних інфраструктур (приватних, спільних або публічних), які залишаються унікальними сутностями, але пов'язані за допомогою стандартизованих чи власних технологій, які дозволяють переносити дані та

програми (наприклад, розширення хмар для балансування навантаження між хмарами).

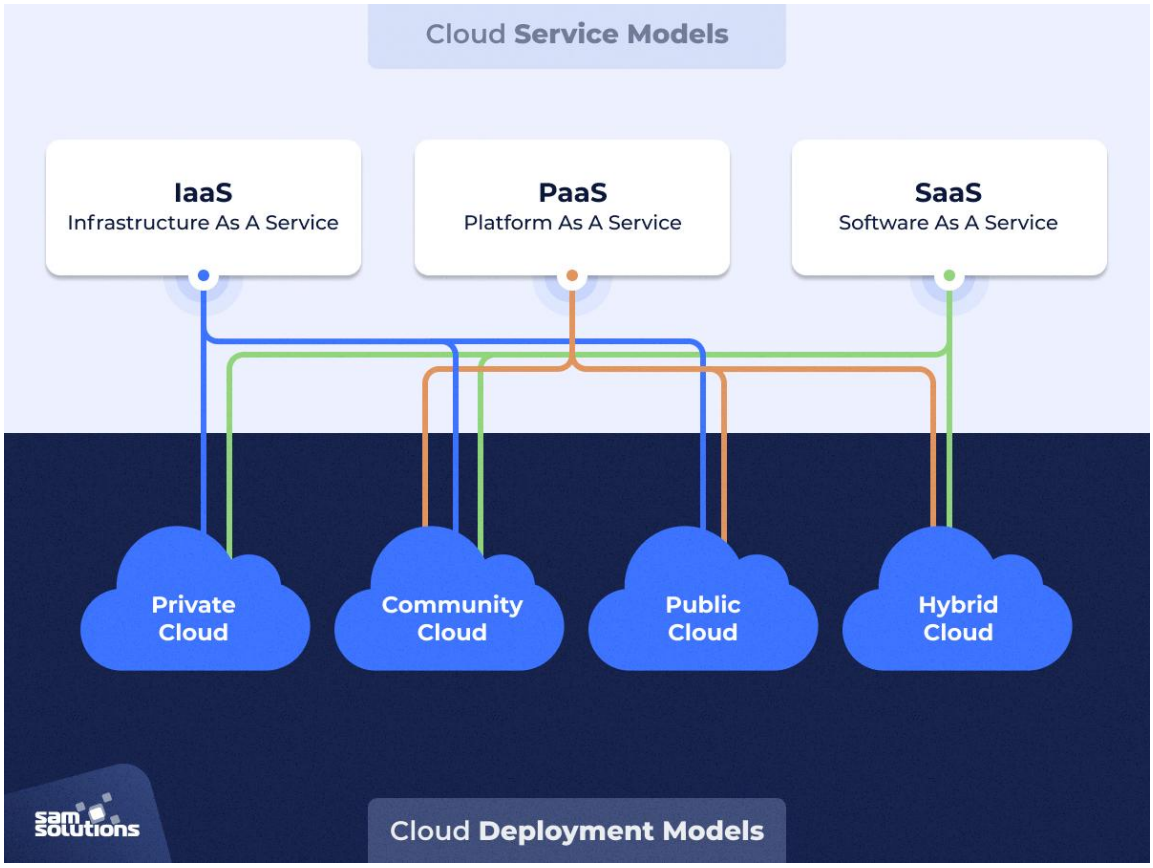


Рис. 1.3. Моделі розгортання додатків у хмарі

1.4 Провайдери хмарних послуг

Найбільшими постачальниками хмарних послуг є: Amazon Web Services, Google Cloud Platform та Microsoft Azure.

Як зазначено у [8], AWS займає лідуючу позицію ринку із 34%, далі йде Microsoft із своїм хмарним рішенням - 22%, Google займає третю позицію і займає 11% ринку хмарних технологій. Наглядно це можна побачити на рис. 1.4.

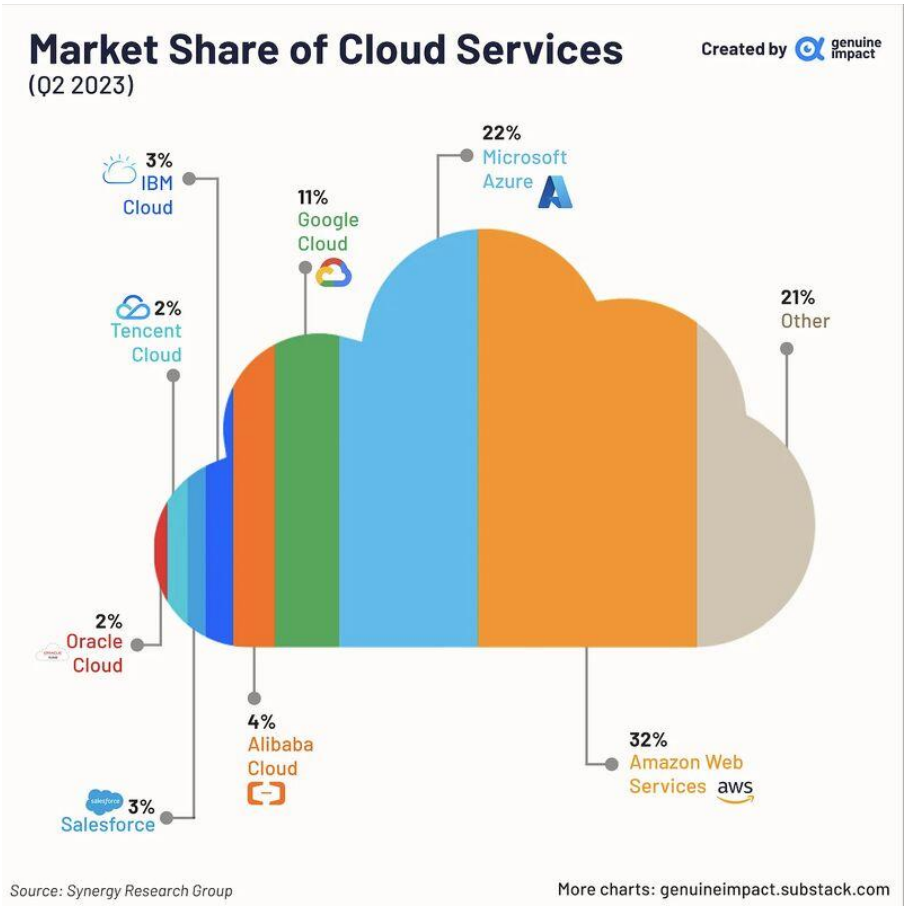


Рис. 1.4. Розподіл ринку провайдерів хмарних послуг

Amazon Web Services

Amazon Web Services (AWS) - набір хмарних сервісів, які надають обчислення, зберігання та інші функціональні можливості на основі хмарних технологій, що дозволяють організаціям та індивідуумам розгортати застосунки та послуги за запитом і за розумною ціною. Пропозиції Amazon Web Services [1] доступні через HTTP, використовуючи протоколи REST і SOAP.

Найбільш використовуваним сервісом AWS є Amazon Elastic Compute Cloud (Amazon EC2), що дозволяє користувачам хмари запускати та керувати екземплярами серверів у центрах обробки даних за допомогою API або доступних інструментів та утиліт.

Екземпляри EC2 - це віртуальні машини, які працюють на основі віртуалізаційного рушія Xen. Після створення та запуску екземпляра користувачі можуть завантажувати програмне забезпечення та вносити зміни до нього. Після завершення змін їх можна упакувати як новий образ машини. Тоді

можна запускати ідентичну копію в будь-який час. Користувачі мають практично повний контроль над всім програмним стеком на екземплярах EC2, які для них виглядають як апаратне забезпечення. З іншого боку, ця функція робить автоматичне масштабування ресурсів важким завданням для Amazon.

EC2 надає можливість розміщувати екземпляри в кількох місцях. Місця EC2 складаються з Регіонів та Доступних Зон. Регіони складаються з однієї або декількох Доступних Зон, розташованих географічно. Доступні Зони - це відокремлені місця, які спеціально інженеруються для ізоляції від відмов у інших Доступних Зонах і забезпечують недороге, маловитратне мережеве з'єднання з іншими Доступними Зонами в тому ж Регіоні.

Зображення машин EC2 зберігаються в і вилучаються з Amazon Simple Storage Service (Amazon S3). S3 зберігає дані як об'єкти, які групуються в бакети. Кожен об'єкт містить від 1 байта до 5 гігабайтів даних. Імена об'єктів в основному є URI шляхами. Бакети повинні бути явно створені перед тим, як їх можна буде використовувати. Бакет може зберігатися в одному з декількох Регіонів. Користувачі можуть вибрати Регіон для оптимізації затримки, мінімізації витрат або відповідно до вимог регулювальних органів.

Amazon Virtual Private Cloud (VPC) - це безпечний і непомітний місток між існуючою IT-інфраструктурою компанії та хмарою AWS. Amazon VPC дозволяє підприємствам підключити їхню існуючу інфраструктуру до набору ізольованих обчислювальних ресурсів AWS через віртуальну приватну мережу (VPN), а також розширити їхні можливості управління, такі як служби безпеки, брандмауери та системи виявлення вторгнень, для включення їхніх ресурсів AWS. Для користувачів хмар Amazon CloudWatch є корисним інструментом управління, який збирає сирові дані з партнерських сервісів AWS, таких як Amazon EC2, а потім обробляє інформацію в читабельні, майже в реальному часі метрики. Метрики щодо EC2 включають, наприклад, використання ЦП, мережеві байти вхід/вихід, операції зчитування/запису на диск і т.д.

Microsoft Windows Azure

Платформа Windows Azure від компанії Microsoft [3] складається з трьох компонентів, кожен з яких надає конкретний набір сервісів користувачам хмарних рішень. Windows Azure забезпечує середовище на основі Windows для запуску додатків та зберігання даних на серверах у центрах обробки даних; SQL Azure надає послуги з обробки даних в хмарі на основі SQL Server; а .NET Services пропонують розподілені інфраструктурні послуги для хмарних та локальних додатків. Платформу Windows Azure можна використовувати як додатки, що працюють у хмарі, так і додатки, що працюють на локальних системах. Крім того, Windows Azure підтримує додатки, побудовані на базі .NET Framework та інших мов, які підтримуються в системах Windows, таких як C, Visual Basic, C++ та інші. Windows Azure підтримує загальні програми, а не лише один клас обчислень. Розробники можуть створювати веб-додатки за допомогою технологій, таких як ASP.NET і Windows Communication Foundation (WCF), додатки, які працюють як незалежні фонові процеси, або додатки, які комбінують обидва підходи. Windows Azure дозволяє зберігати дані у блобах, таблицях та чергах, до яких можна отримати доступ у стилі REST через HTTP або HTTPS. Компоненти SQL Azure - це SQL Azure Database та Huron Data Sync. SQL Azure Database побудований на основі Microsoft SQL Server і забезпечує систему управління базами даних (СУБД) в хмарі. Дані можна отримати, використовуючи ADO.NET та інші інтерфейси доступу до даних у Windows. Користувачі також можуть використовувати програмне забезпечення на місці роботи для роботи з цією інформацією у хмарі. Huron Data Sync синхронізує реляційні дані між різними локальними СУБД.

Google Cloud Platform

Google Cloud Platform (GCP) - це платформа для традиційних веб-застосунків у дата-центрах, керованих Google [2]. Наразі підтримуються мови програмування Python і Java. Веб-фреймворки, що працюють на платформі Google App Engine, включають Django, CherryPy, Pylons та web2py, а також

спеціальний фреймворк для веб-застосунків, написаний Google, схожий на JSP або ASP.NET. Google відповідає за розгортання коду на кластер, моніторинг, аварійне відновлення та запуск екземплярів застосунків за необхідності. Поточні API підтримують такі функції, як зберігання та отримання даних з нереляційної бази даних BigTable, виконання HTTP-запитів та кешування. Розробники мають доступ лише для читання до файлової системи на App Engine.

1.5 Постановка задачі

Як видно із проведеного аналізу, від вибору провайдера хмарних послуг та конфігурації конкретного сервісу залежить вартість використання конкретної послуги, продуктивність додатку та швидкість холодного старту. Тому вибір правильного провайдера послуг та параметрів сервісу є ключовим при розробці кінцевого продукту підприємства.

У дисертації розроблено програмне забезпечення, що виступає посередником між клієнтом та провайдерами хмарних послуг при виборі хмарних послуг.

Основні задачі, які вирішує додаток:

- спростити вибір провайдера хмарних послуг;
- спростити конфігурацію сервісу;
- зменшити кількість людино-годин та коштів затрачених на аналіз ринку провайдерів хмарних послуг та хмарних технологій вцілому;
- зменшити затрати призведені вибором невірної конфігурації хмарного провайдера та конфігурації обраного сервісу;

Розроблене програмне забезпечення надає зручний інтерфейс користувача для замовлення хмарного сервісу. Програмне рішення буде підтримувати два найрозповсюдженіших провайдери хмарних сервісів: Amazon Web Services та Google Cloud Platform та їх сервіси згідно моделі PaaS: AWS Lambda та Google Cloud Function.

Висновки до розділу

У розділі проведено аналіз предметної області. Дано визначення поняттю хмарних послуг. Розглянуті різні сервісні моделі надання хмарних послуг, в тому числі - хмарні функції. Визначено основні характеристики хмарних обчислень. Перелічено основні моделі розгортання додатків у хмарі, їх переваги та недоліки. Проведено аналіз поточного ринку надання послуг з хмарних обчислень, визначені провайдери хмарних послуг.

Представлена постановка задачі, доведено актуальність теми дослідження.

2 ПОБУДОВА БРОКЕРА ХМАРНИХ ПОСЛУГ

2.1 Аналіз методів багатокритеріальної оптимізації

Багатокритеріальний аналіз прийняття рішень (MCDA) - це багатоетапний процес, який включає набір методів для структуризації та формалізації процесів прийняття рішень з прозорістю та послідовністю [22]. MCDA налічує багато методів для вирішення визначених проблем. Для використання цих методів важливо визначити проблему, альтернативи та критерії, які можуть бути різного типу: вартісні показники, індикатори впливу на навколишнє середовище, соціальні показники, енергоефективність, якість та інші специфічні критерії, що є відповідними для даної проблеми. Коли існує багато альтернатив для однієї проблеми, важливо знайти найбільш придатну альтернативу з кращими критеріями вартості, найменшим впливом на навколишнє середовище та високою енергоефективністю. Цього можна досягти за допомогою методу MCDA як інструменту для порівняння альтернатив. Існує багато методів, які можна використовувати для вирішення проблем, і їх можна впорядковувати за різними параметрами. Кожен метод MCDA має свій власний метод обчислення, за яким альтернативи розташовуються, і неможливо стверджувати, що використання конкретних методів з однаковими вхідними даними призведе до одного й того ж кінцевого результату. Методи можуть бути обрані залежно від типу результату [23]. Якщо результат потрібен у вигляді порівняння значень, можна використовувати методи AHP, MULTIMOORA, MAUT, метод зваженої суми, метод зваженого добутку та інші. З використанням AHP, TOPSIS, VIKOR та інших методів можна досягти визначеної мети та знайти найкращу альтернативу серед запропонованих варіантів. Методи PROMETHEE та ELECTRE ґрунтуються на парних порівняннях та оцінці відповідності для досягнення бажаної мети.

2.1.1 TOPSIS

TOPSIS - метод компенсаторної агрегації, який порівнює набір альтернатив, нормалізуючи бали для кожного критерію та обчислюючи геометричну відстань між кожною альтернативою та ідеальною альтернативою, яка є найкращим показником у кожному критерії. Ваги критеріїв у методі TOPSIS можуть бути розраховані за допомогою методу Ординального Підходу до Пріоритету, Аналітичного ієрархічного процесу і т.д. Однією з передумов TOPSIS є те, що критерії монотонно зростають або зменшуються. Нормалізація, як правило, є необхідною, оскільки параметри або критерії часто мають невідповідні розміри в багатокритеріальних задачах. [5][6] Компенсаторні методи, такі як TOPSIS, дозволяють здійснювати компроміси між критеріями, де слабкий результат за одним критерієм може бути компенсований хорошим результатом за іншим критерієм. Це надає більш реалістичну форму моделювання, ніж некомпенсаторні методи, які включають чи виключають альтернативні рішення на основі жорстких відсічень.

Алгоритм TOPSIS:

1. Створюється оціночна матриця з m альтернатив та n критеріїв, де перетин кожної альтернативи та критерію позначений як x_{ij}. Отримаємо матрицю $(x_{ij})_{mxn}$.
2. Матриця $(x_{ij})_{mxn}$ нормалізується, утворюючи матрицю $R = (r_{ij})_{mxn}$ за допомогою нормалізації:

   $r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=2}^{m} x_{kj}^2}}, i = 1,2,\dots,m, j = 1,2,\dots,n$ (1.1)
3. Обчислюється зважена нормалізована матриця рішень

   $t_{ij} = r_{ij} * w_j$, де $w_j = W_j / \sum_{k=1}^{n} W_k , j = 1,2,\dots,n$, (1.2) так щоб $\sum_{i=1}^{n} w_i = 1$ (1.3).
4. Визначається найгірший варіант (A_w) та найкращий варіант (A_b):

$(\mathrm{A_w}) = \{ < \max(t_{ij} | \, i = 1,2,\ldots,m) \mid j \in J_- > , < \min(t_{ij} | \, i = 1,2,\ldots,m) \mid j \in J_+ > \} \equiv \{t_{wj} | j = 1,2,\ldots,n\}$ (1.4)

$(\mathrm{A_b}) = \{ < \min(t_{ij} | \, i = 1,2,\ldots,m) \mid j \in J_- > , < \max(t_{ij} | \, i = 1,2,\ldots,m) \mid j \in J_+ > \} \equiv \{t_{bj} | j = 1,2,\ldots,n\}$, (1.5)

де

$$J_+ = \{j = 1,2,\ldots,n | j\} \text{ (1.6)}$$

$$J_- = \{j = 1,2,\ldots,n | j\} \text{ (1.7)}$$

5. Обчислюються L2 відстань між цільовою альтернативою і та найгіршою умовою $(\mathrm{A_w})$:

$$d_{iw} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{wj})^2}, i = 1,2,\ldots,m, \text{ (1.8)}$$

Та L2 відстань між цільовою альтернативою і та найкращою умовою умовою $(\mathrm{A_b})$:

$$d_{ib} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{bj})^2}, i = 1,2,\ldots,m, \text{ (1.9)}$$

6. Обчислюється схожість до найгіршої умови s_{iw}:

$$s_{iw} = d_{iw}/(d_{iw} + d_{ib}), 0 \le s_{iw} \le 1, i = 1,2,\ldots,m \text{ (1.10)}$$

7. Альтернативи впорядковуються відповідно до s_{iw}.

2.1.2 AHP метод

Аналітичний процес ієрархій (AHP) є одним з найпопулярніших та широко застосовуваних методів багатокритеріального аналізу. У цій техніці процеси оцінки альтернатив та агрегування для вибору найбільш відповідних альтернатив інтегровані. Техніка використовується для ранжування множини альтернатив або вибору найкращого серед них. Вибір здійснюється щодо загальної мети, яка розкладається на набір критеріїв.

Застосування методології полягає у встановленні ваг важливості, що пов'язуються з критеріями при визначенні загальної мети. Це відбувається за допомогою порівняння критеріїв попарно. Розглянемо два критерії, C_j і C_k.

Відповідального за вибір запитують про висловлення оцінки щодо пари щодо відносної важливості C_j перед C_k у відношенні до мети. Порівняльне оцінювання зафіксоване на семантичній шкалі (однаково важливо, більш важливо або дуже важливо і т.д.) і перетворюється у числове ціле значення a_{jk} . Відносна важливість C_k перед C_j визначається як його обернене значення, тобто:

$$a_{kj} = 1/a_{jk}. \quad (1.11)$$

Далі утворюється матриця парних порівнянь A з використанням a_{jk} для всіх пар j та k. Зауважимо, що $a_{jj} = 1$. Загальна домовленість, що ваги критеріїв можна оцінити, знаходячи головний власний вектор w матриці A:

$$\mathrm{AW} = \lambda_{\max}\mathrm{w} \quad (1.12)$$

Після нормалізації вектор w стає вектором пріоритетів критеріїв відносно мети; $\lambda_{\max}$ - найбільше власне значення матриці A, а відповідний власний вектор w містить лише позитивні елементи. Методологія також включає в себе встановлені процедури перевірки послідовності оцінок, наданих приймачем рішень.

Використовуючи аналогічні процедури, обчислюються ваги альтернатив відносно кожного критерію. Потім загальні ваги альтернатив обчислюються за допомогою зваженої суми:

$$\text{Загальна вага альтернативи i} = \sum_{\mathrm{j}} \text{Вага альтернативи i відносно } \mathrm{C_j} * \text{Вага } \mathrm{C_j} \text{ відносно мети}$$

Популярність AHP випливає з його простоти, гнучкості, інтуїтивного звернення і здатності комбінувати кількісні та якісні критерії в одній і тій же рамці прийняття рішень. Незважаючи на його популярність, у літературі було відзначено кілька недоліків AHP, які обмежили його застосування. Однак було запропоновано кілька модифікацій оригінального AHP, таких як мультиплікативний AHP (MAHP), щоб подолати ці обмеження. Деякі з помітних недоліків AHP включають наступні обмеження:

- Ранговий реверс: Ранжування альтернатив, визначене оригінальним AHP, може бути змінене додаванням іншої альтернативи для розгляду. Наприклад, коли AHP використовується для проблеми вибору технологій, можливе, що ранги технологій розвертаються при додаванні нової технології до списку технологій. Один зі способів подолання цієї проблеми полягає в тому, щоб включити всі можливі технології та критерії на початку вправи AHP, і не додавати або не видаляти технології під час або після завершення вправи. Деякі варіанти, такі як MAHP, не страждають від такого типу рангового реверсу.
- Кількість порівнянь: AHP використовує зайві оцінки для перевірки послідовності, і це може експоненційно збільшувати кількість оцінок, які потрібно отримати від DMs. Наприклад, для порівняння восьми альтернатив за одним критерієм потрібно 28 оцінок. Якщо є n критеріїв, то загальна кількість оцінок для порівняння альтернатив на підставі всіх цих критеріїв становить 28n.

2.1.3 Алгоритм VIKOR

VIKOR [30] - метод оптимізації, де існує кілька передвизначених та визначених альтернатив (показників, варіантів і т. д.) та критеріїв (параметрів, вимог і т. д.) для задачі прийняття рішень, і його мета - знайти найкращу серед інших альтернатив, враховуючи внесок критеріїв. Припустимо, що у проблемі F існує альтернатива i щодо критерію j. Для оцінки альтернатив необхідні конкретні чи розмиті дані. У більшості випадків для визначення числових значень альтернатив потрібна експертна оцінка (суб'єктивний суд), яка зазвичай залежить від шкал. Також необхідні характеристики критеріїв, незалежно від того, чи це витрати чи вигода.

Першим кроком розраховується матриця вибору F де m та n представляють кількість альтернатив та критеріїв у матриці прийняття рішень відповідно, і в

подальшому, індекси і та j вказують на відповідну альтернативу або критерій у матриці.

$$F = \begin{pmatrix} f_{11} & \cdots & f_{1n} \\ \vdots & \ddots & \vdots \\ f_{m1} & \cdots & f_{mn} \end{pmatrix} (1.13)$$

Далі йде нормалізація матриці вибору - перегрупування даних шляхом отримання нормалізованої матриці прийняття рішень (R), де її елементи позначаються як R_{ij}. Для знаходження значень rij використовуються наступні формули у рівняннях:

$$r_{ij} = \frac{f_j^+ - f_{ij}}{f_j^+ - f_{ij}^-} \ (1.14)$$

$$\text{де } f_j^+ = \max_i f_{ij} \text{ і } f_j^- = \min_i f_{ij} \ (1.15)$$

Наступним кроком йде розрахунок зваженої нормалізованої матриці вибору. На цьому кроці ваги кожного критерію множаться на відповідні значення rij, щоб отримати зважену нормалізовану матрицю прийняття рішень, як в рівнянні:

$$\text{V} = \begin{bmatrix} v_{11} & \cdots & f_{1n} \\ \vdots & \ddots & \vdots \\ v_{m1} & \cdots & v_{mn} \end{bmatrix}, \text{ де } v_{ij} = r_{ij} * w_{ij} \ (1.16)$$

Метод VIKOR використовує Si як стратегію максимального значення корисності та Ri як індивідуальне значення сумління для визначення та ранжування альтернатив. Ці значення необхідні для подальшого кроку розрахунку значення Qi. Значення Si та Ri знаходяться на основі мангаттенської відстані та відстані Чебишева:

$$S_i = \sum_{j=1}^{n} w_j \frac{f_j^+ - f_{ij}}{f_j^+ - f_{ij}^-} = \sum_{j=1}^{n} v_{ij} \ (1.17)$$

$$R_i = \max_j w_j \frac{f_j^+ - f_{ij}}{f_j^+ - f_{ij}^-} = \max_j v_{ij} \ (1.18)$$

Далі за алгоритмом VIKOR здійснюється процес агрегації значень Si та Ri для розрахунку значень Qi при прийнятті рішень. У випадку можливої зміни

рейтингів альтернатив на основі значень Si та Ri, використовується узагальнюючий показник. Основним чином, краща альтернатива вважається альтернативою з мінімальним значенням Qi, де альтернативи ранжуються на основі інформації Qi.

$$Q_i = v\frac{S_i - S^+}{S^- - S^+} + (1 - v)\frac{R_i - R^+}{R^- - R^+} \quad (1.19)$$

$$S^+ = \min_i S_i, S^- = \max_i S_i, R^+ = \min_i R_i, R^- = \max_i R_i \quad (1.20)$$

Фінальним кроком алгоритму є аналіз результатів. На цьому етапі проводяться контрольні перевірки щодо відповідності отриманих результатів двом заданим умовам:

- Умова 1 стосується прийнятної переваги. Припустимо, що $Q(a_0)$ та $Q(a_{00})$ - це найкращі та другі за рейтингом альтернативи, а DQ дорівнює $1/(m-1)$. Якщо $Q(a_{00}) - Q(a_0) \geq DQ$, це означає, що умова 1 - прийнятна перевага - виконана.
- Умова 2 стосується прийнятної стабільності при прийнятті рішень. Альтернатива a_0 також повинна бути найвище ранжованою за допомогою S та/або R. Якщо обидві умови виконані і задовольняються, то ранжування альтернатив за значеннями Qi вважається вірним.

2.2 Архітектура модулю ранжування постачальників хмарних послуг

Для можливості ранжування постачальників хмарних послуг використано метод VIKOR [27], який обрано на основі узагальненої структури, розробленої Jankowski. [29]. В рамках цієї структури проаналізовано колекцію з 56 доступних методів MCDA, і була створена ієрархічна множина функцій методу та база правил. Це інноваційний інструмент для вибору підходу MCDA, найбільш відповідного для поставленої задачі прийняття рішень. Акронім VIKOR означає VlseKriterijumska Optimizacija I Kompromisno Resenje (багатокритеріальна оптимізація та компромісне рішення) і був розроблений

сербським дослідником у 1998 році [30]. VIKOR надає компромісне рішення [31], і в літературі алгоритм VIKOR високо визнаний не лише в дослідженнях операцій та проблемах прийняття рішень [31], але й в галузях великих даних, інформаційних технологій та управління знаннями.

У цьому моделюванні використовуються три різні параметри, додатково до ваг користувача: вартість за секунду виконання, продуктивність, швидкість холодного старту.

Рисунок 2.1 показує матрицю оцінки для запропонованого методу. Значення параметрів обчислюються за допомогою самостійних програмних інструментів, які були розроблені, як описано раніше. Останній стовпець показує ваги, встановлені користувачем. Вищезазначена матриця оцінки слугує введенням для алгоритму MCDA VIKOR, а вихід - рейтинг постачальників хмарних послуг. Алгоритм VIKOR визначає і досягає компромісного порядку (рейтингу) для альтернатив або варіантів та компромісного рішення на основі кількох передвизначених ваг критеріїв або параметрів в складних задачах прийняття рішень.

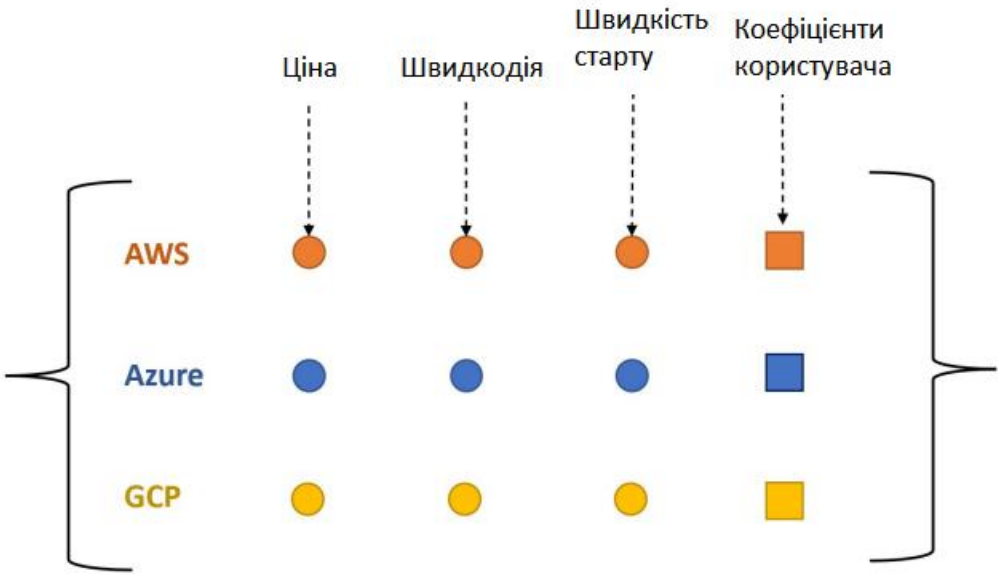


Рис 2.1. Матриця оцінки

2.3 Алгоритм роботи брокеру хмарних послуг

Алгоритм створення хмарної функції можна розбити на такі етапи:

- Користувач за допомогою веб клієнту створює запит на створення хмарної функції.
- Застосунок брокера завантажує виконуваний файл хмарної функції користувача на сервери підтримуваних системою хмарних провайдерів та створює хмарну функцію на всіх доступних конфігураціях для кожного з провайдерів у тестовому режимі.
- Брокер запускає створені хмарні функції з тестовими даними, наданими користувачем, деяку кількість разів, для отримання метрик про швидкодію функції користувача на всіх наявних конфігураціях. Після цього сервер агрегує отримані метрики, усереднюючи дані швидкодії.
- Брокер будує матрицю вибору, необхідну для ранжування конфігурацій хмарних провайдерів, на основі агрегованих метрик з попереднього кроку. За допомогою цієї матриці, брокер обирає найкращу конфігурацію за методом VIKOR.
- Застосунок брокера видаляє всі тестові хмарні функції, які були створені для збору метрик. Брокер створює фінальний варіант хмарної функції на обраній конфігурації обраного хмарного провайдеру.
- Брокер повертає інформацію про створену хмарну функцію до веб клієнту.

Брокер реалізований у вигляді серверу. Загальну схему роботи запропонованого алгоритму показано на рис. 2.2.

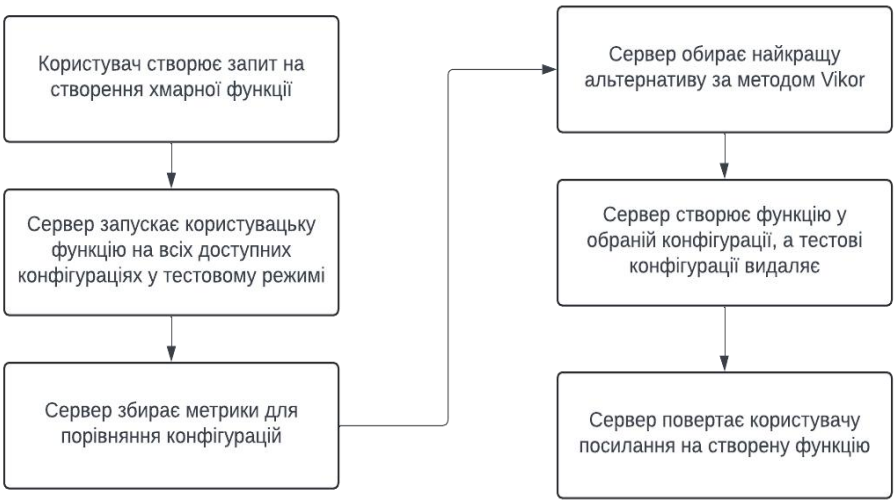


Рис 2.2. Алгоритм створення хмарної функції.

2.3 Архітектура програмного забезпечення брокеру хмарних послуг для обчислення хмарної функції за вказаними параметрами

Програмне забезпечення брокера складається з 4 основних частин:

а) Web додаток, що дозволяє користувачу завантажувати файл функції для виконання у хмарному середовищі, вводити параметри для вибору провайдера хмарних обчислень.

б) Серверний застосунок, що:

- обробляє запити користувачів.
- обирає за методом VIKOR провайдера для обчислення заданої користувачем функції в хмарному середовищі.
- замовляє послугу у обраного провайдера.
- відповідає за збереження та безпеку даних користувачів.

в) База даних, що зберігає користувачів, їх замовлені послуги.

г) Інтеграція з провайдерами хмарних послуг з використанням відповідного API (application programming interface).

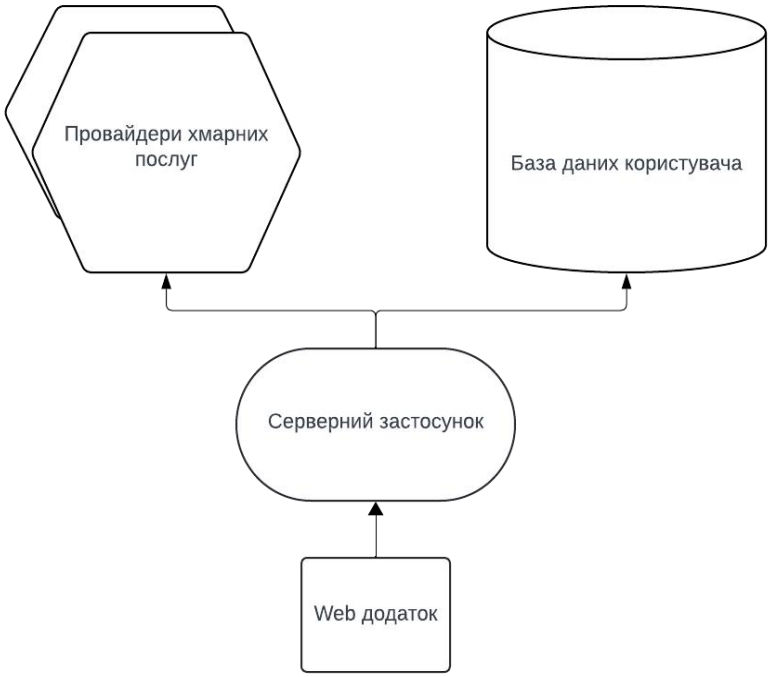


Рис 2.3. Діаграма архітектури програмного забезпечення

Серверний застосунок складається з декількох модулів:

а) Модуль авторизації та аутентіфікації користувача.

Цей модуль відповідає за можливість входу користувача до власного облікового запису, його створення та отримання доступу до замовлених хмарних функцій.

б) Модуль вибору провайдера хмарних послуг за алгоритмом VIKOR.

Цей модуль буде проводити ранжування постачальників хмарних послуг за описаним вище алгоритмом VIKOR. Також, для отримання значень холодного старту, цей модуль буде завантажувати та запускати хмарну функцію до кожного провайдера хмарних послуг, використовуючи описаний нижче модуль інтеграції з провайдером хмарних послуг.

в) Модуль проксіювання запитів користувача.

Цей модуль надає користувачам доступ до їх замовлених функцій за допомогою посилань-ідентифікаторів в середині серверного застосунку. Це потрібно щоб розділити замовлені послуги різних користувачів та надати користувачеві доступ лише до його замовлених послуг.

г) Модуль зберігання даних.

Цей модуль буде відповідати за зв’язок додатку із базою даних.

ґ) Модуль інтеграції з провайдерам хмарних послуг (для кожного підтримуваного хмарного провайдера).

Модуль, який дозволить завантажувати, модифікувати, видаляти та налаштовувати конфігурацію для наданої користувачем хмарної функції, а також, власне, запускати обчислення хмарної функції.

Також модулі інтеграції будуть надавати дані щодо можливих конфігурацій сервісів та їх ваг, які потрібні для ранжування за алгоритмом VIKOR. Таким чином, ранжування можливо буде робити не тільки за провайдерами хмарних послуг, а й за можливими конфігураціями, які можна замовити в конкретного провайдера.

На додачу, цей модуль буде дозволяти дізнатися інформацію про метрики використання та роботу конкретної користувацької хмарної функції, та розраховувати витрати коштів.

Для можливості легкої інтеграції з різними хмарними провайдерами буде створено інтерфейс, що буде ховати деталі реалізації взаємодії з API конкретного провайдера.

Висновки до розділу

У розділі було розроблено математичну модель брокеру для прийняття рішень на базі алгоритму багатокритеріальної оптимізації VIKOR.

Розглянуті різні алгоритми ранжування на основі математичних методів багатокритеріальної оптимізації. Порівняно їх переваги та недоліки. Обрані основні критерії вибору провайдеру хмарних послуг, сформовано методику ранжування провайдерів хмарних послуг та їх конфігурацій.

Запропоновано алгоритм збору метрик, потрібних для прийняття рішення щодо вибору оптимальної конфігурації для обчислення хмарної функції користувача.

Наведено архітектуру програмного забезпечення, що буде виступати брокером хмарних послуг. Сформовано вимоги до такого програмного забезпечення. Сформовано перелік модулів, що має містити розроблене програмне забезпечення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ БРОКЕРУ ХМАРНИХ ПОСЛУГ

3.1 Вибір технологій та мов програмування

Для розробки серверного додатку було обрано кросплатформну мову програмування Java.

Для спрощення розробки та зменшення необхідності написання шаблонного коду було використано екосистему фреймворку Spring Boot:

- Spring MVC для написання веб серверу за принципом REST посилань.
- Spring Data JPA для доступу до бази даних у вигляді об’єктної моделі Java та написання SQL запитів.
- Spring Security для можливості авторизації та аутентифікації користувача у системі, генерації та валідації Json Web токенів.

Для зберігання інформації про користувача, його наявні хмарні функції та їх конфігурацію було використано реляційну базу даних PostgreSQL.

Для інтеграції з хмарними провайдерами було використано надані провайдерами бібліотеки для мови програмування Java, що дозволяють отримати доступ до API керування сервісами хмарних обчислень.

Для розробки частини представлення було використано мову програмування TypeScript разом із бібліотекою React, для зручного відображення компонентів на сторінці браузеру.

3.2 Уніфікований інтерфейс розробки хмарних функцій та його програмна реалізація мовою Java

Для розробки хмарної функції, кожний з провайдерів надає інструменти, які мають бути використані користувачем, який хоче скористатися хмарною послугою відповідного провайдера. Ці інструменти існують у вигляді бібліотек, які користувач підключає до свого проекту, до якого належить програмна

реалізація його хмарної функцій. Однак, хмарна функція, створена з використанням такої бібліотеки, може бути запущена лише на потужностях провайдеру хмарних послуг, який цю бібліотеку розробив. Такий підхід має ряд недоліків:

- Користувач має обирати серед конфігурацій, які надає один обраний хмарний провайдер.
- Користувач не може перенести хмарну функцію з одного провайдера на інший без внесення змін до програмної реалізації його хмарної функції.
- Інструменти розробки хмарної функції можуть з часом зазнавати змін, що змусить користувача вносити зміни до програмної реалізації його хмарної функції.

Для вирішення цих проблем було створено уніфікований інтерфейс розробки хмарних послуг. Основною ідеєю цього інтерфейсу є простота, що приховує складність реалізації обробки користувацьких запитів на обчислення хмарної функції з використанням наданих хмарним провайдером інструментів розробки.

Інтерфейс має назву *CloudFunction* та налічує в собі один абстрактний метод *compute*, що приймає набір вхідних даних у вигляді словника ключ-значення який і на виході повертає словник вихідних даних, що є результатом обчислення хмарної функції. Для того, щоб створити хмарну функцію з використанням даного інструменту розробки, користувач має імпортувати бібліотеку, що містить в собі інтерфейс *CloudFunction,* наявну для обраної мови програмування, до свого проекту і створити програмну реалізацію цього інтерфейсу.

У роботі створено програмну реалізацію запропонованого інтерфейсу у вигляді бібліотеки для мови програмування Java, з підтримкою двох найбільших хмарних провайдерів - Amazon Web Services та Google Cloud Platform.

Рис. 3.1 UML діаграма інтерфейсу розробки хмарних функцій.

Для використання цієї бібліотеки, користувач має імпортувати jar файл бібліотеки до свого проекту та створити клас, що буде реалізувати інтерфейс org.and1ss.CloudFunction. Після цього треба скомпілювати проект таким чином, щоб у classpath була наявна запропонована бібліотека. Хмарна функція, створена за описаною схемою може бути запущена як у хмарі AWS, так і GCP.

3.3 Розробка структури бази даних

Як було зазначено раніше, в якості сховища даних було обрано реляційну базу даних PostgreSQL. Це означає, що структура сховища має моделюватися у термінах Сутність - Зв’язок.

Основні сутності: users - сутність, що зберігає інформацію про користувача та cloud_functions - сутність, що зберігає інформацію про хмарну функцію, її конфігурацію та власника.

Детальну інформацію про ці сутності та їх поля наведено у таблицях 3.1, 3.2.

Табл. 3.1. Опис сутності users.

Колонка	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор користувача у системі
username	TEXT	Унікальне ім’я користувача
password	TEXT	Зашифрований пароль користувача

Табл 3.2. Опис сутності cloud_function.

Колонка	Тип даних	Опис
id	BIGSERIAL	Унікальний ідентифікатор хмарної функції у системі
name	TEXT	Назва хмарної функції у системі
cloud_provider_name	TEXT	Назва хмарної функції в системі хмарного провайдеру
cloud_provider	CloudProvider	Тип хмарного провайдеру, в системі якого була створена хмарна функція
runtime	Runtime	Тип середовища, який буде запускати обчислення хмарної функції
vcpu	FLOAT	Кількість віртуальних процесорних ядер на яких буде відбуватися обчислення хмарної функції

ram	FLOAT	Кількість оперативної пам’яті у мегабайтах, що доступно для використання під час обчислення хмарної функції
owner_id	BIGINT	Ідентифікатор користувача, що створив хмарну функцію. Являє собою зв’язок з сутністю користувача. Зовнішний ключ - посилання на таблицю users
created_at	TIMESTAMP	Час створення хмарної функції

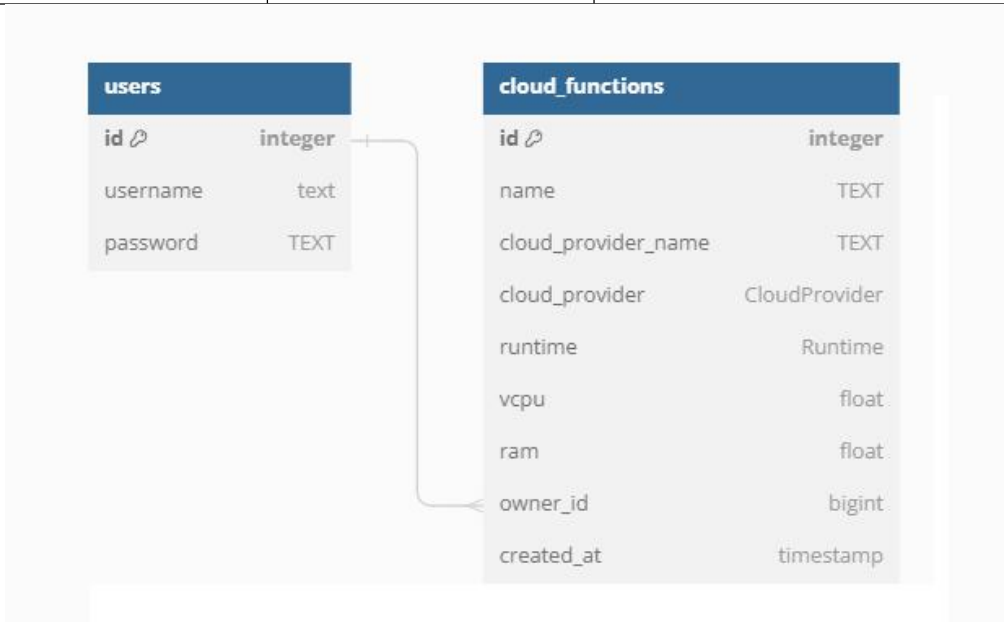


Рис. 3.2. Діаграма сутностей бази даних

3.4 Розробка серверної частини застосунку

Як описано раніше, для розробки серверної частини додатку обрано фреймворк Spring Boot. Spring Boot надає користувачу інструменти для зручної розробки серверної частини веб додатків. При цьому розробнику пропонується створювати свій застосунок з використанням уніфікованої багатошарової архітектури.

Багатошарова архітектура, також відома як "Цибулина" є шаблоном проектування, який був представлений Джеффрі Палермо. Вона зосереджена на ідеї організації коду у концентричні шари, кожен з яких має певну відповідальність. Це потрібно для того, щоб зберегти ядро доменної логіки в центрі, оточеному шарами, які представляють такі аспекти, як сервісні програми, інфраструктура та користувацький інтерфейс.

Рівні багатошарової архітектури:

- Доменний шар (найвнутрішній шар)
- Бізнесовий/Сервісний шар
- Шар представлення

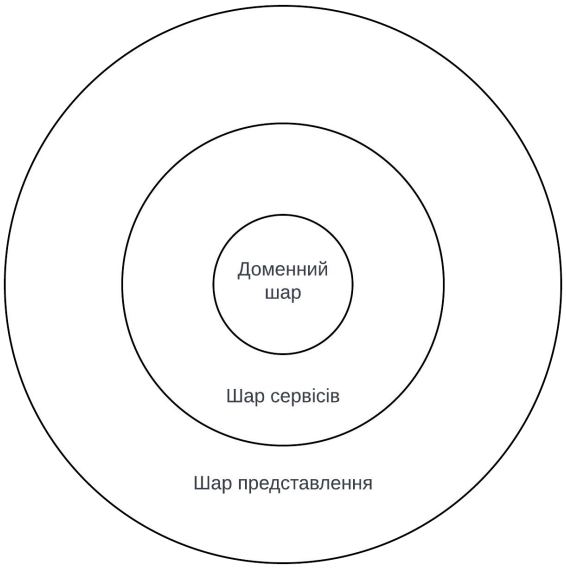


Рис. 3.3. Схематичне представлення багатошарової архітектури

Далі в цій роботі серверний застосунок буде побудовано з урахуванням цієї архітектури.

3.4.1 Розробка шару представлення

У Spring Boot додатку, що дотримується багатошарової архітектури, шар презентації обробляє HTTP-запити та відповіді, виконуючи роль інтерфейсу між клієнтом та основною логікою додатку. Цей шар використовує класи, анотовані @RestController, для прив'язки HTTP-запитів до методів. Він перевіряє дані запиту за допомогою анотацій, і керує помилками за допомогою спеціальних обробників виключень. Контролери делегують бізнес-логіку компонентам шару сервісів, які впроваджуються за допомогою механізму впровадження залежностей Spring. Шар презентації перетворює доменні моделі на об'єкти передачі даних (DTO) для відповідей клієнтів, забезпечуючи чіткий API контракт. Він формує HTTP-відповіді з відповідними статус-кодами та вмістом тіла. Така архітектура забезпечує обробку логіки запитів/відповідей шаром презентації, делегування бізнес-логіки шару сервісів, використання DTO для передачі даних, а також включає надійну перевірку та обробку помилок, роблячи додаток чистим, підтримуваним та тестованим.

У розробленому серверному застосунку присутні такі обробники запитів як AuthController і CloudFunctionController, які обслуговують 5 наявних REST посилань. Наявні REST посилання, їх призначення, вхідні та вихідні дані:

POST /api/v1/auth/registration - посилання, що дозволяє користувачу зареєструватись у системі для подальшої його ідентифікації.

Вхідні дані наведено у таблиці 3.3.

Табл. 3.3. Вхідні дані для посилання /api/v1/auth/registration.

Назва	Тип	Опис
username	String	Унікальне ім’я користувача у системі

password	String	Пароль доступу до системи

POST /api/v1/auth/login - посилання, що дозволяє користувачу отримати Json Web Token (JWT token) для ідентифікації користувача у системі.

Вхідні дані наведено у таблиці 3.4.

Табл. 3.4. Вхідні дані для посилання /api/v1/auth/registration.

Назва	Тип	Опис
username	String	Унікальне ім’я користувача у системі
password	String	Пароль доступу до системи

Вихідні дані: строка що містить JWT token.

GET /api/v1/cloud-function - посилання, що дозволяє користувачу отримати інформацію про створені ним хмарні функції.

Вихідні дані: список створених користувачем хмарних функцій та інформації про них у вигляді JSON об’єктів. Список полів цих JSON об’єктів, їх тип даних та їх опис наведено у таблиці 3.5.

Табл. 3.5. Опис полів JSON об’єктів хмарної функції.

Назва	Тип	Опис
name	String	Назва хмарної функції
cloudProvider	String	Назва провайдеру хмарних послуг в системі якого була створена хмарна функція
runtime	String	Середовище в якому відбувається обчислення хмарної функції
vCpu	double	Кількість віртуальних процесорних ядер доступних для використання при обчисленні хмарної функції

ram	double	Кількість оперативної пам’яті у мегабайтах, доступних для використання при обчисленні хмарної функції
createdAt	String	Час створення хмарної функції

POST /api/v1/cloud-function - посилання, що дозволяє користувачу створити хмарну функцію.

Вхідні дані наведено у таблиці 3.6.

Табл. 3.6. Вхідні дані посилання POST /api/v1/cloud-function

Назва	Тип	Опис
name	String	Назва хмарної функції
handler	String	Назва обробника запитів у виконуваному файлі функції користувача
runtime	String	Середовище в якому відбувається обчислення хмарної функції
testPayload	String	Вхідні дані, які будуть використані під час тестових запусків користувацької функції для збору метрик
weights	double[]	Ваги користувача у форматі: вага ціни за запуск хмарної функції, вага кількості віртуальних

		процесорів, вага кількості доступної оперативної пам’яті, вага часу холодного запуску, вага часу теплого запуску
executable	byte[]	Виконуваний файл користувацької функції

Вихідні дані: інформація про створену хмарну функцію у вигляді JSON об’єкту. Список полів цих JSON об’єктів, їх тип даних та їх опис наведено у таблиці 3.5.

POST /api/v1/cloud-function/{function-name} - посилання, що дозволяє користувачу запустити виконання створеної хмарної функції

Вхідні дані наведено у таблиці 3.7.

Табл. 3.7. Вхідні дані для посилання /api/v1/cloud-function/{function-name}

Назва	Тип	Опис
function-name	String	Назва хмарної функції
payload	String	Вхідні дані для виконання хмарної функції

Вихідні дані: Рядок результату виконання хмарної функції користувача.

3.4.2 Розробка сервісного рівня

Шар сервісів виступає посередником між шаром презентації та основним доменним шаром, інкапсулюючи бізнес-логіку додатку. Цей шар містить сервісні класи, анотовані як @Service, і здійснює бізнес-операції, координуючи роботу з доменними моделями та репозиторіями. Він забезпечує застосування бізнес-правил і стабільне управління станом системи. Сервіси отримують запити від контролерів, обробляють їх відповідно до бізнес-правил та взаємодіють з доменним шаром для виконання таких операцій, як валідація даних,

перетворення та збереження. Використовуючи впровадження залежностей, сервіси можуть використовувати репозиторії для доступу до даних та інші сервіси за потреби. Шар сервісів сприяє дотриманню принципу розподілу обов'язків, зберігаючи бізнес-логіку поза межами контролерів і роблячи її повторно використовуваною та тестованою.

Назва та основний функціонал класів - сервісів розроблюваного програмного забезпечення описано у таблиці 3.8.

Табл. 3.8. Опис класів сервісного шару програмного забезпечення.

Назва	Опис
CloudFunctionBrokerService	Сервіс, задача якого зв’язувати ішні сервіси. Надає можливість замовлення хмарної функції за запропонованим у даній роботі алгоритмом, запуску та видалення вже створенної функції
CloudFunctionBrokerRanker	Сервіс, що надає можливість ранжування різних конфігурацій брокерів хмарних функцій за значеннями метрик, зібраних під час тестового обчислення хмарної функції користувача
VikorMCDARanker	Сервіс, що реалізує математичне обчислення алгоритму багатокритеріальної оптимізації Vikor

3.4.3 Розробка доменного рівня

Доменний рівень є ядром додатку, де розміщена основна бізнес-логіка та правила, незалежні від зовнішніх аспектів, таких як доступ до даних або інтерфейс користувача. Цей шар включає доменні сутності, які представляють основні бізнес-об'єкти, та об'єкти-значення, що інкапсулюють значення зі специфічними правилами. Він також містить доменні сервіси, які обробляють

складні бізнес-операції, що не можуть бути природно призначені сутностям або об'єктам-значенням. Доменний шар забезпечує дотримання бізнес-правил та інваріантів, гарантуючи, що стан системи завжди є послідовним та коректним. Ізолюючи бізнес-логіку в доменному шарі, архітектура сприяє чіткому розподілу обов'язків, роблячи додаток більш підтримуваним та масштабованим. Крім того, цей шар взаємодіє з прикладним шаром через інтерфейси, забезпечуючи абстрагування будь-яких інфраструктурних або зовнішніх залежностей, таким чином зберігаючи чистоту бізнес-логіки.

Доменний шар розроблюваного додатку містить як доменні сутності, так і доменні сервіси, назви та опис яких наведено у таблицях 3.9 та 3.10.

Табл. 3.9. Доменні сервіси.

Назва	Опис
CloudBroker	Інтерфейс, що являє собою узагальнення брокерів хмарних функцій у системі конкретного провайдеру хмарних послуг. Надає інформацію про доступні конфігурації середи виконання хмарних функцій, наявні хмарні функції. Надає можливість створювати та видаляти хмарні функції. Надає можливість запуску обчислення наявної у системі хмарної функції.
AwsLambdaBroker	Сервіс, що реалізує інтеграцію із Amazon Web Services, є реалізацією доменного інтерфейсу CloudFunctionBroker. Надає можливість завантажувати виконуваний файл хмарної функції користувача до AWS, створювати Lambda функцію з

	вказанням різних конфігурацій середи виконання, запускати та видаляти наявні Lambda функції
GcpFunctionBroker	Сервіс інтеграції із Google Cloud Platform, є реалізацією доменного інтерфейсу CloudFunctionBroker. Надає можливість архівувати та завантажувати виконуваний файл хмарної функції до Google Cloud Storage, створювати Function за допомогою завантаженого архіву з вказанням різних конфігурацій середи виконання, запускати та видаляти наявні функції
UserService	Надає можливість виконувати реєстрацію нового користувача у системі, отримання Json Web Token для доступу до ресурсів користувачу за даними авторизації
CustomUserDetailsService	Допоміжний сервіс, що реалізує наданий Spring Security інтерфейс, необхідний для авторизації доступу користувача до ресурсів, ідентифікації користувача у системі за вхідним запитом
UserRepository	Сервіс репозиторій, що реалізує зв’язок з базою даних та надає доступ додатку до облікових даних користувачів
CloudFunctionRepository	Сервіс репозиторій, що реалізує зв’язок з базою даних та надає додатку доступ до даних про створені користувачами хмарні функції, їх хмарного провайдеру та

	інформації про конфігурацію

Табл. 3.10. Доменні класи

Назва	Опис
CloudFunction	Клас, що описує створену хмарну функцію користувача, є відображенням сутності cloud_functions у базі даних
CloudProvider	Клас, що описує провайдера хмарних послуг
User	Клас, що описує обліковий запис користувача, є відображенням сутності users у базі даних

Текст програми серверного застосунку наведено у додатках.

3.5 Розробка клієнтського застосунку

Веб-додатки, створені за допомогою React і TypeScript, зазвичай дотримуються сучасної компонентно-орієнтованої архітектури, яка підкреслює модульність, повторне використання та типобезпеку.

Компнонентно-орієнтована архітектура

Додатки React будуються за допомогою компонентів, які є багаторазовими та автономними частинами інтерфейсу користувача. Компоненти можуть бути функціональними або класовими, причому функціональні компоненти є більш поширеними у сучасній розробці на React через їх простоту та впровадження хуків. Кожен компонент відповідає за рендеринг частини інтерфейсу та управління власним станом і життєвим циклом. Компоненти можуть вкладатися один в одного, дозволяючи створювати складні інтерфейси користувача з простих, багаторазових блоків. Використовуючи TypeScript, ці компоненти виграють від статичної перевірки типів, що допомагає виявляти помилки на ранніх етапах розробки та забезпечує кращу документацію через анотації типів.

Структура додатку

Типовий додаток React і TypeScript структурований на кілька ключових частин:

- Компоненти, часто організовані у папки за функціоналом або призначенням.
- Управління станом: Стан може керуватися локально в межах компонентів або глобально за допомогою бібліотек управління станом, таких як Redux, MobX або React Context API.
- Маршрутизація: Для обробки навігації та маршрутизації в додатку використовуються такі бібліотеки, як React Router, що дозволяє створювати односторінкові додатки (SPA).
- Інтеграція з API: Для взаємодії з бекенд API використовуються сервіси або хуки. Інтерфейси або типи TypeScript визначають форму даних, отриманих з API, забезпечуючи типобезпеку.
- Стилі: Для стилізації компонентів використовуються CSS-in-JS рішення, такі як styled-components, або традиційні CSS/SCSS.

У роботі компоненти було побудовано з використанням розширення мови TypeScript, що дозволяє описувати HTML та стилі разом із кодом, що оперує над ними.

Перелік компонентів, наявних у веб клієнті розроблюваного застосунку, та їх функції описано у таблиці 3.11.

Табл. 3.11. Компоненти клієнтського застосунку

Назва	Опис
CloudFunctionListItem	Компонент, що відповідає за відображення елементу списку хмарних функцій. Відображає назву, провайдера та середу виконання хмарної функції користувача. Дає змогу запустити, видалити та відкрити вікно детальної інформації

	про відповідну функцію користувача
CloudFunctionList	Компонент, що відповідає за відображення списку компонентів типу CloudFunctionListItem, завантаження списку наявних хмарних функцій користувача з серверного застосунку. Також надає можливість відкрити модальне вікно створення нової хмарної функції
CreateCloudFunctionModal	Компонент, що відповідає за відображення модального вікна створення нової хмарної функції
CloudFunctionDetails	Компонент, що відображає детальну інформацію про конкретну хмарну функцію. Відображає назву, провайдера хмарних послуг, кількість віртуальних процесорних ядер, кількість доступної оперативної пам’яті, посилання, за яким можна запустити хмарну функцію. Надає можливість спробувати запустити хмарну функцію за допомогою графічного інтерфейсу
App	Головний компонент клієнтського застосунку, який відображає на екран всі інші компоненти

Текст програми клієнтського застосунку наведено у додатках.

В результаті маємо клієнтський застосунок із графічним інтерфейсом користувача, що допомагає зручно користуватися функціоналом серверного застосунку.

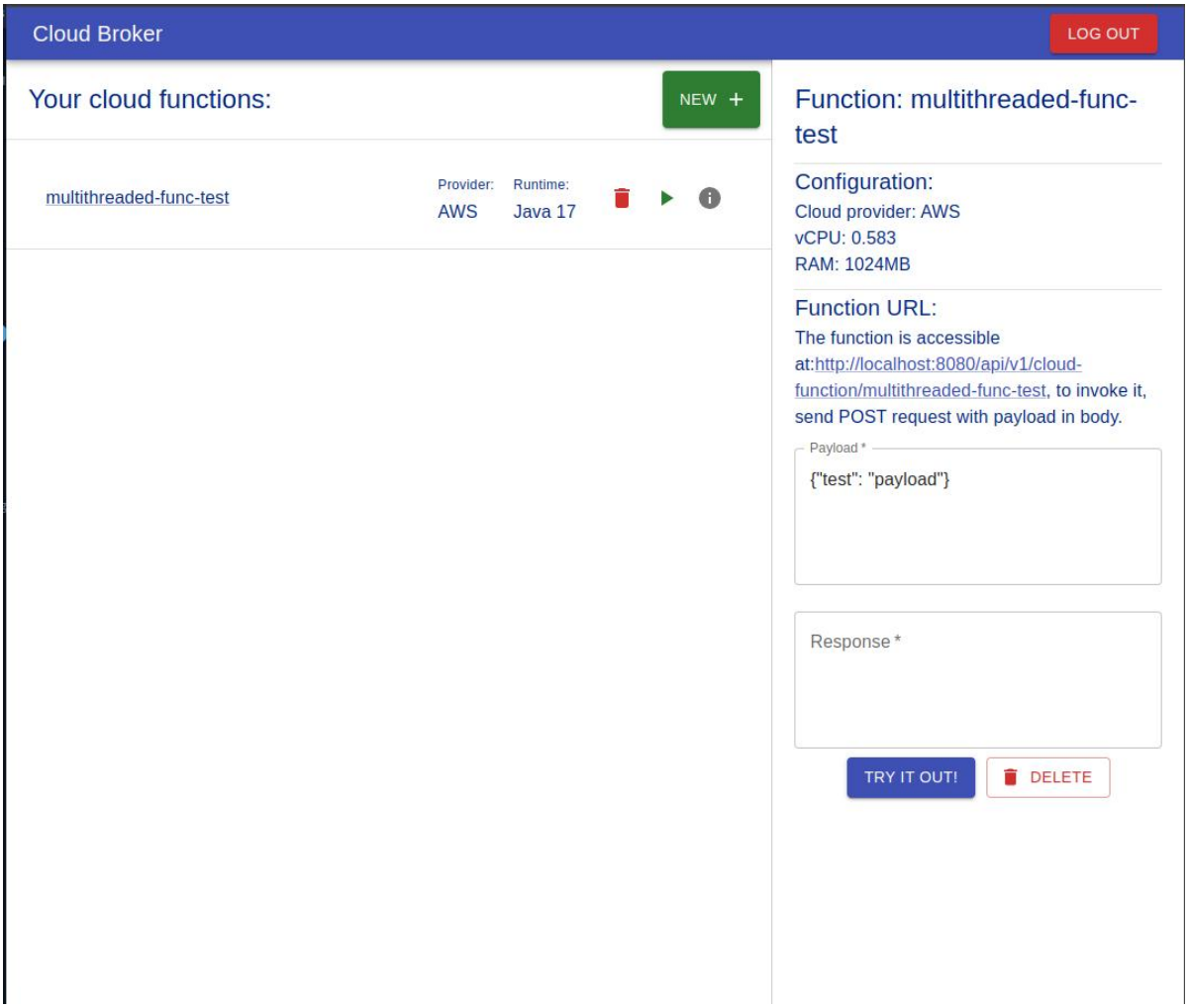


Рис. 3.4. Розроблений графічний інтерфейс користувача

Висновки до розділу

У розділі було описано вибір інструментів необхідних для створення прикладного програмного забезпечення брокеру хмарних послуг, а саме: мови програмування, фреймворки та бібліотеки.

Описано схему бази даних, її основні сутності та зв’язок між ними.

Описано багатошарову архітектуру серверного застосунку, описано реалізацію доменного, сервісного та презентаційного рівнів. Наведено основні класи, їх функціонал. Описано основні REST посилання, вхідні та вихідні дані.

Створено уніфікований інструмент для розробки хмарних функцій користувача.

Запропоновано реалізацію клієнтського веб застосунку із графічним

інтерфейсом користувача, для зручного управління функціоналом розробленого застосунку брокера.

4 РЕЗУЛЬТАТИ

4.1 Методика проведення експериментального дослідження

Для перевірки результатів роботи програмного забезпечення брокеру хмарних послуг було проведено декілька експериментів, в яких перевірялася можливість створення хмарної функції з різними конфігураціями в залежності від хмарної функції користувача, її профілю навантаженності, а також ваг користувача. Для цього було розроблено декілька хмарних функцій з використанням описаної у розділі 3 Java бібліотеки, що дозволило збільшити варіативність доступних для тестування конфігурацій запуску. Далі за допомогою розробленого програмного забезпечення було зібрано метрики ефективності для кожної підтримуваної конфігурації для кожної з розроблених експериментальних функцій, обрано провайдера хмарних послуг та конфігурацію для виконання хмарного обчислення. На останок, було проведено аналіз оптимальності обраних конфігурацій на основі отриманих метрик за рахунок порівняння метрик обраної програмним забезпеченням та найгіршої конфігурацій.

4.2 Метрики оцінки, використані у експерименті

Під час вибору провайдера та конфігурації хмарної функції, користувача цікавлять час виконання коду та вартість виконання обчислень. Тому оптимальна конфігурація для користувача - це та конфігурація, яка надає можливість якнайшвидшого обчислення за найменші кошти, з урахуванням вагів. Для пошуку оптимальної конфігурації було обрано метрики, що дозволяють оцінити:

- Кількість зекономлених коштів за один запуск обчислення хмарної функції у доларах США.
- Час, зекономлений на холодному запуску хмарних функції у мілісекундах.

– Час, зекономлений на теплому запуску хмарної функції у мілісекундах.

Такий набір метрик дозволяє порівнювати конкретні конфігурації хмарних функцій залежно від ваг користувача та робити висновки щодо доцільності їх вибору.

4.3 Проведення експериментів та збір необхідних метрик

4.3.1 Обчислення функції користувача з акцентом на завантаженність процессору

Для даного експерименту було створено функцію, яка навмисне завантажує процесор, щоб змоделювати ситуацію, в якій час обчислення функції користувача залежить від потужності процесору. Створена функція рекурсивно обчислює числа фібоначі у декілька потоків, щоб були задіяні всі віртуальні ядра процесору, які доступні у конкретній конфігурації середовища виконання, в якій відбувається запуск хмарної функції користувача. Лістинг коду такої функції, створеної мовою програмування Java, наведено на рис. 4.1.

```
9  public class Main implements CloudFunction {
10
11      public static int fibonacci(int n) {
12          return switch (n) {
13              case 0 -> 0;
14              case 1 -> 1;
15              default -> fibonacci(n - 1) + fibonacci(n - 2);
16          };
17      }
18
19      @Override
20      public Map<String, Object> compute(Map<String, Object> s) {
21          List<Thread> threads = new ArrayList<>();
22
23          for (int i = 0; i < 8; i++) {
24              Thread thread = new Thread(() -> System.out.println("Fib: " + fibonacci(40)));
25              threads.add(thread);
26              thread.start();
27          }
28
29          threads.forEach(thread -> {
30              try { thread.join(); } catch (InterruptedException ignored) {}
31          });
32
33          return Map.of(
34                  "Output", "Hello World!, its cloud function created with CloudFunctionAPI!",
35                  "Input", s);
36      }
37  }
```

Рис. 4.1. Лістинг коду модельованої функції

Далі цю функцію скомпільовано та запущено алгоритм збору метрик та вибору конфігурації за допомогою розробленого програмного забезпечення, з різними вагами.

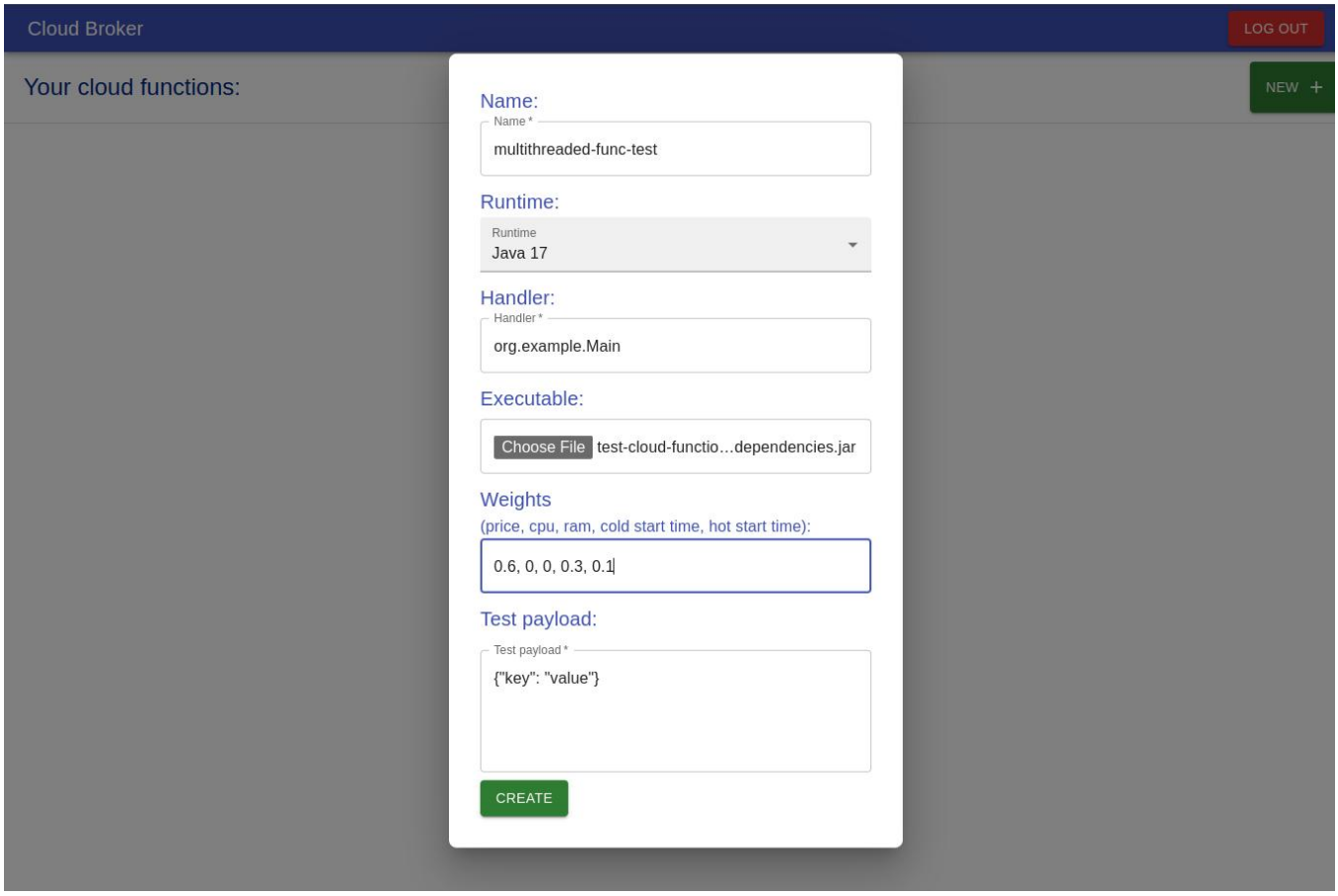


Рис. 4.2. Приклад створення експериментальної функції

В результаті роботи алгоритму, було зібрано метрики швидкодії та ціни, які можна побачити у таблиці 4.1.

Табл 4.1. Зібрані експериментальні метрики

Провайдер	cpu	RAM(MB)	холодний старт(мс)	теплий старт(мс)	ціна $
AWS	0.0072	128	27241	25122	3.28E-05
AWS	0.029	512	21445	20939	1.72E-04
AWS	0.583	1024	10891	10402	1.73E-04
AWS	0.868	1536	7527	6978	1.75E-04
AWS	1.157	2048	5849	5343	1.78E-04

GCP	0.083	128	9656	9295	4.43E-05
GCP	0.167	256	4859	3434	6.20E-05
GCP	0.333	512	4957	3413	1.24E-04
GCP	0.583	1024	4641	3406	2.27E-04
GCP	1	2048	3982	3269	3.91E-04
GCP	2	4096	3767	2014	6.96E-04
GCP	2	8192	3141	1748	7.96E-04
GCP	4	16384	1411	1025	1.50E-03
GCP	8	32768	1153	820	2.90E-03

В результаті роботи алгоритму, залежно від ваг користувача, було обрано конфігурації з метриками, наведеними у таблицях 4.2, 4.3:

Табл. 4.2

Ваги	0	0	0.1	0.2	0.7
Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
GCP	0.083	128	9656	9295	4.43E-05

Табл. 4.3

Ваги	0	0	0	0	1
Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	0.072	128	27241	25122	3.28E-05

4.3.2 Обчислення функції з акцентом на підсистему вводу виводу

Для даного експерименту створено функцію, яка робить HTTP запит, щоб змоделювати ситуацію, в якій час обчислення функції користувача залежить від швидкості підсистеми введення-виведення, а не від доступної потужності процесора. Лістинг коду такої функції, створеної мовою програмування Java, наведено на рис. 4.3.

```java
public class Request implements CloudFunction {

    @Override
    public Map<String, Object> compute(Map<String, Object> map) {
        HttpURLConnection connection = null;

        try {
            var serverAddress = new URL("https://linkedin.com/");
            connection = (HttpURLConnection) serverAddress.openConnection();
            connection.setRequestMethod("GET");
            connection.setDoOutput(true);
            connection.setReadTimeout(10000);
            connection.connect();

            var wr = new OutputStreamWriter(connection.getOutputStream());
            wr.write("");
            wr.flush();

            var rd = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            var sb = new StringBuilder();
            String line;
            while ((line = rd.readLine()) != null)
                sb.append(line).append('\n');

            return Map.of("response", sb.toString());
        } catch (Throwable t) {
            return Map.of("error", t.getCause());
        } finally {
            connection.disconnect();
        }
    }
}
```

Рис. 4.3. Лістинг коду модельованої функції

В результаті роботи алгоритму, було зібрано метрики швидкодії та ціни, які можна побачити у таблиці 4.4.

Табл 4.4. Зібрані експериментальні метрики

Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	0.0072	128	3685	923	8.37E-06
AWS	0.029	512	3537	902	7.48E-06
AWS	0.583	1024	2683	586	9.70E-06
AWS	0.868	1536	2082	487	1.21E-05
AWS	1.157	2048	1726	511	1.70E-05

GCP	0.083	128	3079	1075	2.54E-05
GCP	0.167	256	2884	909	5.04E-05
GCP	0.333	512	3195	1030	1.02E-04
GCP	0.583	1024	2995	916	1.79E-04
GCP	1	2048	3074	917	3.16E-04
GCP	2	4096	2866	848	6.26E-04
GCP	2	8192	3055	885	7.34E-04
GCP	4	16384	2717	843	1.47E-03
GCP	8	32768	2751	823	2.90E-03

В результаті роботи алгоритму, залежно від ваг користувача, було обрано конфігурації з метриками, наведеними у таблицях 4.5, 4.6:

Табл. 4.5

Ваги	0	0	0.1	0.2	0.7
Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	1.157	2048	1726	511	1.70E-05

Табл. 4.6

Ваги	0	0	0	0	1
Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	0.029	512	3537	902	7.48E-06

4.4 Аналіз отриманих результатів

У першому експерименті найгіршою за часом виявилась конфігурація з метриками:

Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	0.0072	128	27241	25122	3.28E-05

Найгіршою за вартістю виявилась конфігурація з метриками:

Провайдер	cpu	RAM(MB)	холодний	теплий	ціна $

			старт	старт	
GCP	8	32768	2751	823	2.90E-03

Виграш у часі виконання, та/або затрачених коштах обраної конфігурації відносно найгіршої, в залежності від ваг користувача наведено у табл 4.7.

Табл. 4.7. Вигода, отримана в результаті першого експерименту

Ваги	0	0	0.1	0.2	0.7
Грошова вигода, USD					0.0028557
Вигода в часі виконання (теплий старт), мс					15827
Вигода в часі виконання (холодний старт), мс					17585
Ваги	0	0	0	0	1
Грошова вигода, USD					0.002867244
Вигода в часі виконання (теплий старт), мс					0
Вигода в часі виконання (холодний старт), мс					0

У другому експерименті найгіршою за часом виявилась конфігурація з метриками:

Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
AWS	0.0072	128	3685	923	8.37E-06

Найгіршою за вартістю виявилась конфігурація з метриками:

Провайдер	cpu	RAM(MB)	холодний старт	теплий старт	ціна $
GCP	8	32768	1153	820	2.90E-03

Виграш у часі виконання, та/або затрачених коштах обраної конфігурації відносно найгіршої, в залежності від ваг користувача наведено у табл 4.8.

Табл. 4.8. Вигода, отримана в результаті другого експерименту

Ваги	0	0	0.1	0.2	0.7
Грошова вигода, USD					0.00288299
Вигода в часі виконання (теплий старт), мс					564
Вигода в часі виконання (холодний старт), мс					1959
Ваги	0	0	0	0	1
Грошова вигода, USD					0.00289252
Вигода в часі виконання (теплий старт), мс					173
Вигода в часі виконання (холодний старт), мс					148

Як можна бачити, обрані алгоритмом конфігурації дійсно мають перевагу у тих аспектах, які важливі користувачу найбільше, алгоритм знаходить найкращі значення потрібних метрик залежно від ваг користувача. Так, наприклад, у експериментах, в яких акцент було зроблено на ціну, найбільшої оптимізації зазнавала метрика ціни, а вигоди в часі могло взагалі не бути, або міг бути програш.

Висновки до розділу

У даному розділі було обрано методику оцінки вибраної конфігурації провайдера хмарних послуг, описано метрики оцінки.

Наведено роботу програмного забезбечення та його результати на експериментальних даних.

Було проведено аналіз отриманих експериментальних даних.

ВИСНОВКИ

У роботі проведено аналіз підходів надання хмарних обчислень. Він показав наявні проблеми та недоліки, їх важливість для користувача. Основною проблемою хмарних обчислень є вибір оптимального провайдеру хмарних послуг та конфігурації потрібного сервісу. Не вірно обрана конфігурація може призвести до колосальних збитків при використанні хмарного сервісу, тому дана задача є актуальною.

Для вирішення задачі вибору запропоновано методику, що дозволяє зменшити витрати, призведені необгрунтованим вибором провайдера хмарних послуг та конфігурації сервісу. Для цього обрано метод багатокритеріальної оптимізації VIKOR, описано критерії вибору та матрицю вибору, метод збору метрик для створення матриці вибору. Описано застосування обраного методу при реалізації брокера хмарних послуг.

Наведено архітектуру програмного забезпечення, що реалізує брокер вибору провайдера для запуску хмарних функцій, та основні вимоги до нього. Наведено опис розробки серверного додатку, бази даних та клієнтського застосунку з графічним інтерфейсом користувача. Наведено приклади роботи розробленого програмного забезпечення.

Для оцінки результатів роботи запропонованого рішення розроблено модель тестування розробленого додатку та оцінки вартості обраної конфігурації сервісу запуску хмарної функції на AWS та GCP. Представлено методику проведення експериментальних досліджень. Проведено ряд експериментів, в яких протестовано роботу програмного продукту, що реалізує описаний у роботі алгоритм. У результаті отримано метрики швидкодії та ціни запуску коду користувача, а також вигоду користувача призведену використанням запропонованого алгоритму.

Експерименти показали, що запропонований алгоритм та його програмна реалізація допомагає зменшити витрати, призведені невірно обраним постачальником хмарних послуг та невірною конфігурацією сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amazon web services [Електронний ресурс] // AWS. – 2024. – Режим доступу до ресурсу: https://aws.amazon.com.
2. Google cloud [Електронний ресурс] // GCP. – 2024. – Режим доступу до ресурсу: https://cloud.google.com.
3. Windows azure [Електронний ресурс] // Azure. – 2024. – Режим доступу до ресурсу: https://www.microsoft.com/azure.
4. Xensource inc [Електронний ресурс] // Xen. - 2024. – Режим доступу до ресурсу: https://www.xensource.com.
5. Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. // In ACM SIGARCH Computer Architecture News, volume 42 – С. 127–144.
6. Brittany King. FaaS explained. [Електронний ресурс] // Digitalocean. – 2024. – Режим доступу до ресурсу: https://www.digitalocean.com/blog/what-is-faas-function-as-a-service-explained.
7. Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware, 2012 – С. 37-48.
8. Rakibul Hassan. Cloud Computing: Literature Review – С. 5.
9. Andrew D Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. In Proceedings of the 7th ACM european conference on Computer Systems, 2012 – С. 99–112.
10. Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. Lowlevel augmented bayesian optimization for finding the best cloud vm.
11. Botong Huang, Matthias Boehm, Yuanyuan Tian, Berthold Reinwald, Shirish Tatikonda, and Frederick R Reiss. Resource elasticity for largescale machine

learning. // In Proceedings of the International Conference on Management of Data, ACM, 2015 – C. 137-152.

12. Virajith Jalaparti, Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Bridging the tenant-provider gap in cloud services. // In Proceedings of the Third ACM Symposium on Cloud Computing, ACM, 2012 – C. 10.
13. Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011)
14. Dejan Novakovi c´, Nedeljko Vasic´, Stanko Novakovic´, Dejan Kostic´, and Ricardo Bianchini. Deepdive: Transparently identifying and man- aging performance interference in virtualized environments. // In Presented as part of the Annual Technical Conference, USENIX, 2013 – C. 219–230.
15. Juwei Shi, Jia Zou, Jiaheng Lu, Zhao Cao, Shiqiang Li, and Chen Wang. Mrtuner: a toolkit to enable holistic optimization for mapreduce jobs. // Proceedings of the VLDB Endowment, 2014.
16. Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The impact of memory subsystem resource sharing on datacenter applications. // In ACM SIGARCH Computer Architecture News, ACM, 2011 – C. 283–294.
17. Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: efficient performance prediction for largescale advanced analytics. // In 13th USENIX Symposium on Networked Systems Design and Implementation, 2016 – C. 363–378.
18. R Clint Whaley, Antoine Petitet, and Jack J Dongarra. Automated empirical optimizations of software and the atlas project. Parallel computing, 2001.
19. Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best vm across multiple public clouds: a data-driven performance modeling approach. // In Proceedings of the Symposium on Cloud Computing, ACM, 2017 – C. 452–465.
20. Makrani, Hosein Mohammadi. Evaluation of software-based faulttolerant techniques on embedded OS's components. // Proceedings of the International Conference on Dependability, 2014.

21. Makrani, Hosein Mohammadi. Energy-aware and Machine Learningbased Resource Provisioning of In-Memory Analytics on Cloud, 2018.
22. Ishizaka A., Nemery P. Multi-Criteria Decision Analysis, 2013.
23. Malik, Maria, Dean M. Tullsen, and Houman Homayoun. Co-Locating and concurrent fine-tuning MapReduce applications on microservers for energy efficiency. // International Symposium on Workload Characterization (IISWC), IEEE, 2017.
24. Topsis MCDA method [Електронний ресурс] // Wikipedia. – 2024. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/TOPSIS.
25. Sayadi, Hossein. Power conversion efficiency-aware mapping of multithreaded applications on heterogeneous architectures: A comprehensive parameter tuning. // Asia and South Pacific Design Automation Conference, IEEE, 2018.
26. Makrani, Hosein Mohammadi. Understanding the role of memory subsystem on performance and energy-efficiency of Hadoop applications. // Eighth International Green and Sustainable Computing Conference (IGSC), IEEE, 2017.
27. VIKOR MCDA method. [Електронний ресурс] // Wikipedia. – 2024. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/VIKOR_method.
28. Ishizaka, A., Nemery, P. Multi-Criteria Decision Analysis: Methods and Software, 2013.
29. Watróbski, J., Jankowski, J., Ziemba, P.; Karczmarczyk, A., Zioło, M. Generalised framework for multi-criteria method selection, 2019 – С. 107-124.
30. Opricovi´c, S. Multicriteria Optimization of Civil Engineering Systems. Ph.D. Thesis, Faculty of Civil Engineering, University of Belgrade, Beograd, Serbia, 1998.
31. Opricovic, S., Tzeng, G.H. Compromise solution by MCDM methods: A comparative analysis of VIKOR and TOPSIS, 2004 – С. 156, 445–455.

ДОДАТКИ

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

CreateFunctionRequestDto.java

```
package org.and1ss.cloudbroker.api.dto;

import lombok.Data;
import org.and1ss.cloudbroker.service.brokers.Runtime;

@Data
public class CreateFunctionRequestDto {

    private String name;

    private String handler;

    private Runtime runtime;

    private String testPayload;

    private String weights;
}
```

AuthController.java

```
package org.and1ss.cloudbroker.api;

import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.service.user.dto.LoginRequestDto;
import org.and1ss.cloudbroker.service.user.dto.LoginResponseDto;
```

```
import org.and1ss.cloudbroker.service.user.dto.RegistrationRequestDto;
import org.and1ss.cloudbroker.domain.User;
import org.and1ss.cloudbroker.security.JwtUtil;
import org.and1ss.cloudbroker.service.user.UserService;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/v1/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final UserService userService;
    private final JwtUtil jwtUtil;

    @PostMapping("/login")
    public LoginResponseDto login(@RequestBody LoginRequestDto loginRequestDto)
{
        final UsernamePasswordAuthenticationToken authenticationToken =
```

```
            new
UsernamePasswordAuthenticationToken(loginRequestDto.getUsername(),
loginRequestDto.getPassword());
        final Authentication authentication =
authenticationManager.authenticate(authenticationToken);
        final String username = authentication.getName();
        final User user = new User().setUsername(username);

        return LoginResponseDto.of(username, jwtUtil.createToken(user));
    }

    @PostMapping("/registration")
    @ResponseStatus(HttpStatus.CREATED)
    public void register(@RequestBody RegistrationRequestDto registrationDto) {
        userService.register(registrationDto);
    }
}
```

CloudFunctionBrokerController.java

```
package org.and1ss.cloudbroker.api;

import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.and1ss.cloudbroker.api.dto.CreateFunctionRequestDto;
import org.and1ss.cloudbroker.domain.CloudFunction;
import org.and1ss.cloudbroker.service.CloudFunctionBrokerService;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInformation;
import org.and1ss.cloudbroker.service.brokers.Runtime;
```

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import java.util.Arrays;
import java.util.List;

@CrossOrigin
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/cloud-function")
public class CloudFunctionBrokerController {

    private final CloudFunctionBrokerService cloudFunctionBrokerService;

    @Value("${host}")
    private String host;

    @GetMapping
```

```
    public List<CloudFunction> listAllCloudFunctions() {
        return cloudFunctionBrokerService.listAllFunctions();
    }

    @PostMapping
    @SneakyThrows
    @ResponseStatus(HttpStatus.CREATED)
    public CloudFunction createCloudFunction(
            @RequestPart("createRequest") CreateFunctionRequestDto requestDto,
            @RequestPart("file") MultipartFile multipartFile) {

        final CloudFunctionInformation cloudFunctionInformation = new
CloudFunctionInformation(
                multipartFile.getBytes(), requestDto.getHandler(), Runtime.JAVA17);

        final double[] weights = Arrays.stream(requestDto.getWeights().split(","))
                .map(String::trim)
                .map(Double::parseDouble)
                .mapToDouble(Double::doubleValue)
                .toArray();

        return cloudFunctionBrokerService.createFunction(
                requestDto.getName(),
                cloudFunctionInformation,
                requestDto.getTestPayload(),
                weights);
    }

    @PostMapping("/{function-name}")
```

```
    public String invokeFunction(@PathVariable(name = "function-name") String
name, @RequestBody String payload) {
        return cloudFunctionBrokerService.invokeFunction(name, payload);
    }

    @DeleteMapping("/{function-name}")
    public void deleteFunction(@PathVariable(name = "function-name") String name)
{
        cloudFunctionBrokerService.deleteFunction(name);
    }
}
```

AwsConfiguration.java

```
package org.and1ss.cloudbroker.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;

@Configuration
public class AwsConfiguration {

    @Bean
    public LambdaClient lambdaClient() {
        return LambdaClient.builder()
                .region(Region.US_EAST_1)
                .build();
```

```
    }
}
```

RestTemplateConfiguration.java

```
package org.and1ss.cloudbroker.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class RestTemplateConfiguration {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

SecurityConfiguration.java

```
package org.and1ss.cloudbroker.configuration;

import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.security.JwtAuthorizationFilter;
import org.and1ss.cloudbroker.service.user.CustomUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.authentication.builders.Authentication
ManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigu
rer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {


    private final CustomUserDetailsService userDetailsService;
    private final JwtAuthorizationFilter jwtAuthorizationFilter;
```

```
    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http,
                                       PasswordEncoder passwordEncoder) throws
Exception {

        final AuthenticationManagerBuilder authenticationManagerBuilder =
                http.getSharedObject(AuthenticationManagerBuilder.class);

        authenticationManagerBuilder
                .userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder);

        return authenticationManagerBuilder.build();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
        return http.csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(
                        auth -> auth
                                .requestMatchers("/api/v1/auth/**").permitAll()
                                .requestMatchers("/swagger-ui/**").permitAll()
                                .requestMatchers("/v3/api-docs/**").permitAll()
                                .requestMatchers("/api/**").authenticated()
                                .anyRequest().permitAll())
                .sessionManagement(
                        sessionManager ->
sessionManager.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
```

```
            .addFilterBefore(jwtAuthorizationFilter,
UsernamePasswordAuthenticationFilter.class)
            .cors(Customizer.withDefaults())
            .build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

CloudFunction.java

```
package org.and1ss.cloudbroker.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import lombok.Data;
import lombok.experimental.Accessors;
```

```
import org.and1ss.cloudbroker.service.brokers.Runtime;
import org.hibernate.annotations.CreationTimestamp;

import java.time.Instant;

@Data
@Entity
@Table(name = "cloud_functions")
@Accessors(chain = true)
public class CloudFunction {

    @Id
    @JsonIgnore
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(name = "cloud_provider_name")
    private String cloudProviderName;

    @Column(name = "cloud_provider")
    @Enumerated(EnumType.STRING)
    private CloudProvider cloudProvider;

    @Column(name = "vcpu")
    private double vCpu;

    private double ram;
```

```
    @Enumerated(EnumType.STRING)
    private Runtime runtime;

    @JsonIgnore
    @ManyToOne
    @JoinColumn(name = "owner_id", referencedColumnName = "id")
    private User owner;

    @CreationTimestamp
    @Column(name = "created_at")
    private Instant createdAt;
}
```

CloudProvider.java

```
package org.and1ss.cloudbroker.domain;

public enum CloudProvider {

    AWS, GCP, AZURE
}
```

User.java

```
package org.and1ss.cloudbroker.domain;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
```

```
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Entity
@Table(name = "users")
@Accessors(chain = true)
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    private String password;
}
```

CloudFunctionRepository.java

```
package org.and1ss.cloudbroker.repository;

import org.and1ss.cloudbroker.domain.CloudFunction;
import org.and1ss.cloudbroker.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
import java.util.List;
import java.util.Optional;

public interface CloudFunctionRepository extends JpaRepository<CloudFunction,
Long> {

   Optional<CloudFunction> findByOwnerAndName(User owner, String name);

   List<CloudFunction> findAllByOwner(User owner);
}
```

UserRepository.java

```
package org.and1ss.cloudbroker.repository;

import org.and1ss.cloudbroker.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {

   Optional<User> findByUsername(String username);
}
```

JwtAuthorizationFilter.java

```
package org.and1ss.cloudbroker.security;
```

```
import com.fasterxml.jackson.databind.ObjectMapper;
import io.jsonwebtoken.Claims;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Map;
import java.util.Optional;

@Component
@RequiredArgsConstructor
public class JwtAuthorizationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final ObjectMapper mapper;

    @Override
```

```
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain)
            throws IOException, ServletException {

        final Optional<String> accessToken = jwtUtil.resolveToken(request);
        if (accessToken.isEmpty()) {
            filterChain.doFilter(request, response);
            return;
        }

        try {
            jwtUtil.resolveClaims(request)
                    .filter(jwtUtil::validateClaims)
                    .map(Claims::getSubject)
                    .map(username -> new
UsernamePasswordAuthenticationToken(username, "", new ArrayList<>()))
                    .ifPresent(authentication ->
SecurityContextHolder.getContext().setAuthentication(authentication));
        } catch (Exception e) {
            response.setStatus(HttpStatus.FORBIDDEN.value());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);

            mapper.writeValue(
                    response.getWriter(),
                    Map.of("message: ", e.getMessage(), "timestamp: ",
LocalDateTime.now()));
        }

        filterChain.doFilter(request, response);
```

```
    }
}
```

AwsConfiguration.java

```
package org.and1ss.cloudbroker.service.brokers.aws;

import lombok.Data;

@Data
public class AwsConfiguration {

    private int ramSize; // in MB

    private double vCPU;

    private double price; // in USD per 1 ms of compute time
}
```

AwsConfigurationProperties.java

```
package org.and1ss.cloudbroker.service.brokers.aws;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import java.util.List;

@Data
```

```
@Component
@ConfigurationProperties(prefix = "broker.aws")
public class AwsConfigurationProperties {

    private String createFunctionRole;

    private String cloudFunctionApiHandler;

    private String cloudFunctionApiHandlerEnvironmentVariable;

    private List<AwsConfiguration> availableConfigurations;
}
```

AwsLambdaBroker.java

```
package org.and1ss.cloudbroker.service.brokers.aws;

import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.domain.CloudProvider;
import org.and1ss.cloudbroker.service.brokers.CloudBrokerConfiguration;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionBroker;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInformation;
import org.and1ss.cloudbroker.service.brokers.Runtime;
import org.springframework.stereotype.Service;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.AddPermissionRequest;
import software.amazon.awssdk.services.lambda.model.CreateFunctionRequest;
```

```
import
software.amazon.awssdk.services.lambda.model.CreateFunctionUrlConfigRequest;
import software.amazon.awssdk.services.lambda.model.DeleteFunctionRequest;
import software.amazon.awssdk.services.lambda.model.FunctionCode;
import software.amazon.awssdk.services.lambda.model.FunctionUrlAuthType;
import
software.amazon.awssdk.services.lambda.model.GetFunctionConfigurationRequest;
import
software.amazon.awssdk.services.lambda.model.GetFunctionConfigurationResponse;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.waiters.LambdaWaiter;

import java.util.List;
import java.util.Map;

@Service
@RequiredArgsConstructor
public class AwsLambdaBroker implements CloudFunctionBroker {

    private final AwsConfigurationProperties properties;
    private final LambdaClient lambdaClient;

    @Override
    public void createFunction(String name,
                               CloudFunctionInformation functionInformation,
                               CloudBrokerConfiguration configuration) {

        try {
```

```
        final LambdaWaiter waiter = lambdaClient.waiter();
        final FunctionCode code = FunctionCode.builder()
                .zipFile(SdkBytes.fromByteArray(functionInformation.executable()))
                .build();

        final CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
                .functionName(name)
                .description("Created by the Cloud Broker")
                .code(code)
                .handler(properties.getCloudFunctionApiHandler())
                .memorySize(configuration.getRamSize())
                .timeout(900)
                .environment(env -> env.variables(

Map.of(properties.getCloudFunctionApiHandlerEnvironmentVariable(),
                                functionInformation.handler())))
                .runtime(toAwsRuntime(functionInformation.runtime()))
                .role(properties.getCreateFunctionRole())
                .build();

        lambdaClient.createFunction(functionRequest);
        final GetFunctionConfigurationRequest getFunctionConfigurationRequest =
                GetFunctionConfigurationRequest.builder()
                        .functionName(name)
                        .build();

        final WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
                waiter.waitUntilFunctionActive(getFunctionConfigurationRequest);
```

```
            waiterResponse.matched().response().ifPresent(System.out::println);
        } catch (LambdaException e) {
            throw new RuntimeException(e);
        }

        final CreateFunctionUrlConfigRequest functionUrlConfigRequest =
CreateFunctionUrlConfigRequest.builder()
                .functionName(name)
                .authType(FunctionUrlAuthType.NONE)
                .build();

        lambdaClient.createFunctionUrlConfig(functionUrlConfigRequest);

        final AddPermissionRequest permissionRequest =
AddPermissionRequest.builder()
                .functionName(name)
                .functionUrlAuthType("NONE")
                .statementId("FunctionURLAllowPublicAccess")
                .principal("*")
                .action("lambda:InvokeFunctionUrl")
                .build();

        lambdaClient.addPermission(permissionRequest);
    }

    @Override
    public void deleteFunction(String name) {
```

```
        final DeleteFunctionRequest deleteFunctionRequest =
DeleteFunctionRequest.builder()
                .functionName(name)
                .build();

        lambdaClient.deleteFunction(deleteFunctionRequest);
    }

    @Override
    public String invokeFunction(String name, String payload) {
        final InvokeRequest invokeRequest = InvokeRequest.builder()
                .functionName(name)
                .payload(SdkBytes.fromUtf8String(payload))
                .build();

        return lambdaClient.invoke(invokeRequest).payload().asUtf8String();
    }

    @Override
    public List<CloudBrokerConfiguration> getAvailableConfigurations() {
        return properties.getAvailableConfigurations().stream()
                .map(configuration -> CloudBrokerConfiguration.of(
                        invocationTime -> invocationTime * configuration.getPrice(),
                        configuration.getVCPU(),
                        configuration.getRamSize()))
                .toList();
    }

    @Override
```

```
    public CloudProvider getCloudProvider() {
        return CloudProvider.AWS;
    }

    private static software.amazon.awssdk.services.lambda.model.Runtime
toAwsRuntime(Runtime runtime) {
        return software.amazon.awssdk.services.lambda.model.Runtime.JAVA17;
    }
}
```

GcpConfiguration.java

```
package org.and1ss.cloudbroker.service.brokers.gcp;

import lombok.Data;

@Data
public class GcpConfiguration {

    private int ramSize; // in MB

    private double vCPU;

    private double price; // in USD per 100 ms of compute time
}
```

GcpConfigurationProperties.java

```
package org.and1ss.cloudbroker.service.brokers.gcp;
```

```
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import java.util.List;

@Data
@Component
@ConfigurationProperties(prefix = "broker.gcp")
public class GcpConfigurationProperties {

    private String projectName;

    private String locationName;

    private String cloudFunctionApiEntrypoint;

    private String cloudFunctionApiHandlerEnvironmentVariable;

    private List<GcpConfiguration> availableConfigurations;
}
```

GcpFunctionBroker.java

```
package org.and1ss.cloudbroker.service.brokers.gcp;

import com.github.benmanes.caffeine.cache.Caffeine;
import com.google.cloud.functions.v2.BuildConfig;
```

```
import com.google.cloud.functions.v2.CreateFunctionRequest;
import com.google.cloud.functions.v2.DeleteFunctionRequest;
import com.google.cloud.functions.v2.Function;
import com.google.cloud.functions.v2.FunctionName;
import com.google.cloud.functions.v2.FunctionServiceClient;
import com.google.cloud.functions.v2.GenerateUploadUrlRequest;
import com.google.cloud.functions.v2.GenerateUploadUrlResponse;
import com.google.cloud.functions.v2.LocationName;
import com.google.cloud.functions.v2.ServiceConfig;
import com.google.cloud.functions.v2.Source;
import com.google.cloud.run.v2.ServiceName;
import com.google.cloud.run.v2.ServicesClient;
import com.google.iam.v1.Binding;
import com.google.iam.v1.GetIamPolicyRequest;
import com.google.iam.v1.Policy;
import com.google.iam.v1.SetIamPolicyRequest;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.and1ss.cloudbroker.domain.CloudProvider;
import org.and1ss.cloudbroker.service.brokers.CloudBrokerConfiguration;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionBroker;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInformation;
import org.and1ss.cloudbroker.service.brokers.Runtime;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
```

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

@Service
@RequiredArgsConstructor
public class GcpFunctionBroker implements CloudFunctionBroker {

    final GcpConfigurationProperties properties;
    final RestTemplate restTemplate;

    final Map<String, String> urlCache = Caffeine.newBuilder()
            .expireAfterAccess(1, TimeUnit.DAYS)
            .maximumSize(10000)
            .<String, String>build()
            .asMap();

    @Override
    @SneakyThrows
    public void createFunction(String name,
                               CloudFunctionInformation functionInformation,
                               CloudBrokerConfiguration configuration) {
```

```
    try (FunctionServiceClient functionServiceClient =
FunctionServiceClient.create()) {
        final GenerateUploadUrlRequest generateUploadUrlRequest =
GenerateUploadUrlRequest.newBuilder()
            .setParent(LocationName.of(properties.getProjectName(),
properties.getLocationName()).toString())
            .build();

        final GenerateUploadUrlResponse generateUploadUrlResponse =
            functionServiceClient.generateUploadUrl(generateUploadUrlRequest);

        uploadZipToGoogleCloud(functionInformation.executable(),
generateUploadUrlResponse.getUploadUrl());

        final Source source = Source.newBuilder()
            .setStorageSource(generateUploadUrlResponse.getStorageSource())
            .build();

        final ServiceConfig serviceConfig = ServiceConfig.newBuilder()
            .putEnvironmentVariables(
                properties.getCloudFunctionApiHandlerEnvironmentVariable(),
functionInformation.handler())
            .setAvailableCpu(String.valueOf(configuration.getCpuCount()))
            .setAvailableMemory(configuration.getRamSize() + "Mi")
            .build();

        final BuildConfig buildConfig = BuildConfig.newBuilder()
            .setRuntime(toGcpRuntime(functionInformation.runtime()))
            .setEntryPoint(properties.getCloudFunctionApiEntrypoint())
```

```
                .setSource(source)
                .build();

        final Function function = Function.newBuilder()
                .setName(FunctionName.of(
                        properties.getProjectName(), properties.getLocationName(),
name).toString())
                .setBuildConfig(buildConfig)
                .setServiceConfig(serviceConfig)
                .build();

        final CreateFunctionRequest request = CreateFunctionRequest.newBuilder()
                .setParent(LocationName.of(properties.getProjectName(),
properties.getLocationName()).toString())
                .setFunction(function)
                .setFunctionId(name)
                .build();

        functionServiceClient.createFunctionAsync(request).get();

        updateIamPolicyForFunction(name);

        final String functionUrl = functionServiceClient.getFunction(
                        FunctionName.of(properties.getProjectName(),
properties.getLocationName(), name))
                .getUrl();
        urlCache.put(name, functionUrl);
    }
  }
```

```
private void uploadZipToGoogleCloud(byte[] executable, String uploadUrl) {
    final byte[] zipBytes;
    try {
        zipBytes = zipBytes("executable.jar", executable);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    final HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.set("Content-Type", "application/zip");

    restTemplate.put(uploadUrl, new HttpEntity<>(zipBytes, httpHeaders));
}

public static byte[] zipBytes(String filename, byte[] input) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ZipOutputStream zos = new ZipOutputStream(baos);
    ZipEntry entry = new ZipEntry(filename);
    entry.setSize(input.length);
    zos.putNextEntry(entry);
    zos.write(input);
    zos.closeEntry();
    zos.close();
    return baos.toByteArray();
}

private void updateIamPolicyForFunction(String functionName) {
```

```
    try (ServicesClient servicesClient = ServicesClient.create()) {
      final ServiceName serviceName = ServiceName.of(
          properties.getProjectName(), properties.getLocationName(),
functionName);

      final Policy policy = servicesClient.getIamPolicy(
          GetIamPolicyRequest.newBuilder()
              .setResource(serviceName.toString())
              .build());

      final Policy updatedPolicy = policy.toBuilder()
          .addBindings(Binding.newBuilder()
              .addMembers("allUsers")
              .setRole("roles/run.invoker")
              .build())
          .build();

      final SetIamPolicyRequest request = SetIamPolicyRequest.newBuilder()
          .setResource(serviceName.toString())
          .setPolicy(updatedPolicy)
          .build();

      servicesClient.setIamPolicy(request);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }

  @Override
```

```
    @SneakyThrows
    public void deleteFunction(String name) {
        try (FunctionServiceClient functionServiceClient =
FunctionServiceClient.create()) {
            urlCache.remove(name);
            final DeleteFunctionRequest deleteFunctionRequest =
DeleteFunctionRequest.newBuilder()
                    .setName(FunctionName.of(
                            properties.getProjectName(), properties.getLocationName(),
name).toString())
                    .build();

            functionServiceClient.deleteFunctionAsync(deleteFunctionRequest).get();
        }
    }

    @Override
    public String invokeFunction(String name, String payload) {
        String functionUrl = urlCache.get(name);
        if (functionUrl == null) {
            try (FunctionServiceClient functionServiceClient =
FunctionServiceClient.create()) {
                final Function function = functionServiceClient.getFunction(
                        FunctionName.of(properties.getProjectName(),
properties.getLocationName(), name));
                functionUrl = function.getUrl();
                urlCache.put(name, functionUrl);
            } catch (Exception e) {
                throw new RuntimeException(e);
```

```
        }
    }

    final HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.setContentType(MediaType.APPLICATION_JSON);

    final ResponseEntity<String> response = restTemplate.postForEntity(
            functionUrl, new HttpEntity<>(payload, httpHeaders), String.class);

    if (!response.getStatusCode().is2xxSuccessful()) {
        throw new RuntimeException(response.getBody());
    }

    return response.getBody();
}

@Override
public List<CloudBrokerConfiguration> getAvailableConfigurations() {
    return properties.getAvailableConfigurations().stream()
            .map(configuration -> CloudBrokerConfiguration.of(
                    invocationTime ->
calculatePricePerInvocation(configuration.getPrice(), invocationTime),
                    configuration.getVCPU(),
                    configuration.getRamSize()))
            .toList();
}

private static double calculatePricePerInvocation(double pricePer100Millis, long
invocationTime) {
```

```
        final long roundedInvocationTime = invocationTime % 100 == 0
                ? invocationTime / 100
                : invocationTime / 100 + 100;

        return roundedInvocationTime * pricePer100Millis;
    }

    @Override
    public CloudProvider getCloudProvider() {
        return CloudProvider.GCP;
    }

    private static String toGcpRuntime(Runtime runtime) {
        return "java17";
    }
}
```

CloudBrokerConfiguration.java

```
package org.and1ss.cloudbroker.service.brokers;

import lombok.Value;

import java.util.function.Function;

@Value(staticConstructor = "of")
public class CloudBrokerConfiguration {

    Function<Long, Double> pricePerInvocation; // function to compute price for given
invocation time
```

```
    double cpuCount;

    int ramSize; // in MB
}
```

CloudBrokerRegistry.java

```
package org.and1ss.cloudbroker.service.brokers;

import org.and1ss.cloudbroker.domain.CloudProvider;
import org.apache.commons.lang3.tuple.Pair;
import org.springframework.stereotype.Service;

import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.stream.Collectors;

@Service
public class CloudBrokerRegistry {

    private final Set<CloudFunctionBroker> brokers;
    private final Map<CloudProvider, CloudFunctionBroker> providerToBroker;

    public CloudBrokerRegistry(Set<CloudFunctionBroker> brokers) {
        this.brokers = Objects.requireNonNull(brokers);
        this.providerToBroker = brokers.stream()
                .map(broker -> Pair.of(broker.getCloudProvider(), broker))
```

```
            .collect(Collectors.toMap(Pair::getKey, Pair::getValue));
    }

    public CloudFunctionBroker getCloudBroker(CloudProvider cloudProvider) {
        return providerToBroker.get(cloudProvider);
    }

    public Set<CloudFunctionBroker> getCloudBrokers() {
        return brokers;
    }
}
```

CloudFunctionBroker.java

```
package org.and1ss.cloudbroker.service.brokers;

import org.and1ss.cloudbroker.domain.CloudProvider;

import java.util.List;

public interface CloudFunctionBroker {

    void createFunction(String name,
                        CloudFunctionInformation functionInformation,
                        CloudBrokerConfiguration configuration);

    void deleteFunction(String name);

    String invokeFunction(String name, String payload);
```

```
    List<CloudBrokerConfiguration> getAvailableConfigurations();

    CloudProvider getCloudProvider();
}
```

CloudFunctionInformation.java

```
package org.and1ss.cloudbroker.service.brokers;

public record CloudFunctionInformation(byte[] executable, String handler, Runtime
runtime) {
}
```

CloudFunctionInvocationMetrics.java

```
package org.and1ss.cloudbroker.service.brokers;

import lombok.Value;

@Value(staticConstructor = "of")
public class CloudFunctionInvocationMetrics {

    long coldStartInvocationTime; // in millis

    long hotStartInvocationTime; // in millis

    double pricePerInvocation; // in usd

    CloudBrokerConfiguration cloudBrokerConfiguration;
```

```
    CloudFunctionBroker broker;
}
```

Runtime.java

```
package org.and1ss.cloudbroker.service.brokers;

import com.fasterxml.jackson.annotation.JsonCreator;

import java.util.Arrays;

public enum Runtime {

    JAVA17("Java 17");

    public final String name;

    Runtime(String name) {
        this.name = name;
    }

    @JsonCreator
    public static Runtime from(String value) {
        return Arrays.stream(values())
                .filter(runtime -> runtime.name.equals(value))
                .findFirst()
                .orElseThrow(() -> new IllegalArgumentException("Can't parse " + value +
" to runtime"));
    }
```

```
}
```

BestWorstItem.java

```
package org.and1ss.cloudbroker.service.ranker;

public record BestWorstItem(double best, double worst) {
}
```

DecisionMatrix.java

```
package org.and1ss.cloudbroker.service.ranker;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class DecisionMatrix {

    private final double[][] items;
    private final boolean[] isBeneficial;

    public DecisionMatrix(double[][] items, boolean[] isBeneficial) {
        if (items[0].length != isBeneficial.length) {
            throw new IllegalArgumentException("Invalid beneficial vector");
        }

        this.items = Arrays.stream(items).map(double[]::clone).toArray(double[][]::new);
        this.isBeneficial = isBeneficial.clone();
```

```
}

public double at(int i, int j) {
    return items[i][j];
}

public int rows() {
    return items.length;
}

public int cols() {
    return items[0].length;
}

public DecisionMatrix normalizePerColumn() {
    final int rows = items.length;
    final int cols = items[0].length;

    double[][] normalizedMatrix = new double[rows][cols];

    for (int j = 0; j < cols; j++) {
        double sum = 0;
        for (int i = 0; i < rows; i++) {
            sum += items[i][j] * items[i][j];
        }

        final double sumSqrt = Math.sqrt(sum);

        for (int i = 0; i < rows; i++) {
```

```
            normalizedMatrix[i][j] = items[i][j] / sumSqrt;
        }
    }

    return new DecisionMatrix(normalizedMatrix, isBeneficial.clone());
}

public List<BestWorstItem> bestWorstItemsPerColumn() {
    final int rows = items.length;
    final int cols = items[0].length;

    final List<BestWorstItem> bestWorstItems = new ArrayList<>(cols);
    for (int j = 0; j < cols; j++) {
        double best;
        double worst;

        if (isBeneficial[j]) {
            best = Double.NEGATIVE_INFINITY;
            worst = Double.POSITIVE_INFINITY;

            for (int i = 0; i < rows; i++) {
                best = Math.max(best, items[i][j]);
                worst = Math.min(worst, items[i][j]);
            }
        } else {
            best = Double.POSITIVE_INFINITY;
            worst = Double.NEGATIVE_INFINITY;

            for (int i = 0; i < rows; i++) {
```

```
                best = Math.min(best, items[i][j]);
                worst = Math.max(worst, items[i][j]);
            }
        }
        bestWorstItems.add(new BestWorstItem(best, worst));
    }

    return bestWorstItems;
  }
}
```

VikorMCDARanker.java

```
package org.and1ss.cloudbroker.service.ranker;

import org.springframework.data.util.Pair;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.IntStream;

public class VikorMCDARanker {

  /*
    final double[][] dec = {
            {3, 6, 4, 20, 2, 30000},
            {4, 4, 6, 15, 2.2, 32000},
            {6, 5, 9, 18, 3, 32100},
```

```
            {5, 6, 3, 23, 2.8, 28000},
            {4, 8, 7, 30, 1.5, 29000},
            {8, 3, 6, 35, 1.9, 27000},
            {7, 2, 5, 33, 1.7, 28500},
            {3, 8, 3, 34, 1.6, 30500},
            {8, 4, 8, 40, 2.5, 33000},
            {9, 3, 7, 34, 2, 29800}
        };

        final boolean[] isBeneficial = {true, true, true, false, false, false};
        final double[] weights = {0.1, 0.2, 0.1, 0.2, 0.1, 0.3};

        data used for test and check with this video:
        https://www.youtube.com/watch?v=9wE-
UWbrVes&ab_channel=Entrepreneuriat%2CRecherchesetConseils
     */
    public static List<Pair<Double, Integer>> rank(DecisionMatrix decisionMatrix,
double[] weights) {
        if (decisionMatrix.cols() != weights.length) {
            throw new IllegalArgumentException(
                    "Weights vector dimension does not match number of columns in
decision matrix");
        }

        if (Math.abs(Arrays.stream(weights).sum() - 1) > 0.05) {
            throw new IllegalArgumentException("Weights sum must be equal to 1");
        }
```

```
        final DecisionMatrix normalizedDecisionMatrix =
decisionMatrix.normalizePerColumn();
        final int rows = decisionMatrix.rows();

        final double[] profitability = findProfitability(normalizedDecisionMatrix,
weights); // S
        final double[] regret = findRegret(normalizedDecisionMatrix, weights); // R

        final double maxProfitability = Arrays.stream(profitability).max().getAsDouble();
// maxS
        final double minProfitability = Arrays.stream(profitability).min().getAsDouble();
// minS
        final double maxRegret = Arrays.stream(regret).max().getAsDouble(); // maxR
        final double minRegret = Arrays.stream(regret).min().getAsDouble(); // minR

        final double[] vikorIndexes = new double[rows]; // Q
        for (int i = 0; i < rows; i++) {
            vikorIndexes[i] = 0.5 * (profitability[i] - minProfitability) / (maxProfitability -
minProfitability)
                    + 0.5 * (regret[i] - minRegret) / (maxRegret - minRegret);
        }

        return IntStream.range(0, rows)
                .mapToObj(index -> Pair.of(vikorIndexes[index], index))
                .sorted(Comparator.comparingDouble(Pair::getFirst))
                .toList();
    }
```

```
    public static double[] findProfitability(DecisionMatrix decisionMatrix, double[]
weights) {
        final List<BestWorstItem> bestWorstItems =
decisionMatrix.bestWorstItemsPerColumn();
        final int cols = decisionMatrix.cols();
        final int rows = decisionMatrix.rows();

        final double[] profitability = new double[rows];
        for (int i = 0; i < rows; i++) {
            double sum = 0;

            for (int j = 0; j < cols; j++) {
                final double best = bestWorstItems.get(j).best();
                final double worst = bestWorstItems.get(j).worst();

                sum += weights[j] * (best - decisionMatrix.at(i, j)) / (best - worst);
            }

            profitability[i] = sum;
        }

        return profitability;
    }

    public static double[] findRegret(DecisionMatrix decisionMatrix, double[] weights)
{
        final List<BestWorstItem> bestWorstItems =
decisionMatrix.bestWorstItemsPerColumn();
        final int cols = decisionMatrix.cols();
```

```
        final int rows = decisionMatrix.rows();

        final double[] profitability = new double[rows];
        for (int i = 0; i < rows; i++) {
            double max = Double.NEGATIVE_INFINITY;

            for (int j = 0; j < cols; j++) {
                final double best = bestWorstItems.get(j).best();
                final double worst = bestWorstItems.get(j).worst();

                max = Math.max(max, weights[j] * (best - decisionMatrix.at(i, j)) / (best -
worst));
            }

            profitability[i] = max;
        }

        return profitability;
    }
}
```

CustomUserDetailsService.java

```
package org.and1ss.cloudbroker.service.user;

import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.domain.User;
import org.and1ss.cloudbroker.repository.UserRepository;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {



    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        final User user = userRepository.findByUsername(username).orElseThrow();
        return org.springframework.security.core.userdetails.User.builder()
                .username(username)
                .password(user.getPassword())
                .roles(List.of("USER").toArray(new String[0]))
                .build();
    }
}
```

UserService.java

```
package org.and1ss.cloudbroker.service.user;
```

```
import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.domain.User;
import org.and1ss.cloudbroker.repository.UserRepository;
import org.and1ss.cloudbroker.service.user.dto.RegistrationRequestDto;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public void register(RegistrationRequestDto registrationDto) {
        final User user = new User();
        user.setUsername(registrationDto.getUsername());
        user.setPassword(passwordEncoder.encode(registrationDto.getPassword()));
        // TODO: Add roles to user

        userRepository.save(user);
    }

    public User getCurrentUser() {
        final Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
```

```
        if (!(authentication instanceof AnonymousAuthenticationToken)) {
            return
userRepository.findByUsername(authentication.getName()).orElseThrow();
        } else {
            throw new IllegalStateException("Trying to retrieve current user when no
authenticated user exist");
        }
    }
}
```

CloudFunctionBrokerRanker.java

```
package org.and1ss.cloudbroker.service;

import org.and1ss.cloudbroker.service.brokers.CloudBrokerConfiguration;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionBroker;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInvocationMetrics;
import org.and1ss.cloudbroker.service.ranker.DecisionMatrix;
import org.and1ss.cloudbroker.service.ranker.VikorMCDARanker;
import org.springframework.data.util.Pair;

import java.util.List;

public class CloudFunctionBrokerRanker {

    private static final boolean[] isCriteriaBeneficial = {
            false, // price per invocation
            true, // cpu count
            true, // ram size
```

```
        false, // cold start invocation time
        false, // hot start invocation time
    };

    public static Pair<CloudFunctionBroker, CloudBrokerConfiguration>
findBestConfiguration(
        List<CloudFunctionInvocationMetrics> invocationMetrics,
        double[] weights) {

        final double[][] criteriaItems = invocationMetrics.stream()
            .map(metric -> new double[]{
                metric.getPricePerInvocation(),
                metric.getCloudBrokerConfiguration().getCpuCount(),
                metric.getCloudBrokerConfiguration().getRamSize(),
                metric.getColdStartInvocationTime(),
                metric.getHotStartInvocationTime()})
            .toArray(double[][]::new);

        final DecisionMatrix decisionMatrix = new DecisionMatrix(criteriaItems,
isCriteriaBeneficial);
        final int bestIndex = VikorMCDARanker.rank(decisionMatrix,
weights).get(0).getSecond();
        final CloudFunctionInvocationMetrics bestChoice =
invocationMetrics.get(bestIndex);
        return Pair.of(bestChoice.getBroker(),
bestChoice.getCloudBrokerConfiguration());
    }
}
```

CloudFunctionBrokerService.java

```
package org.and1ss.cloudbroker.service;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.and1ss.cloudbroker.domain.CloudFunction;
import org.and1ss.cloudbroker.domain.CloudProvider;
import org.and1ss.cloudbroker.domain.User;
import org.and1ss.cloudbroker.repository.CloudFunctionRepository;
import org.and1ss.cloudbroker.service.brokers.CloudBrokerConfiguration;
import org.and1ss.cloudbroker.service.brokers.CloudBrokerRegistry;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionBroker;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInformation;
import org.and1ss.cloudbroker.service.brokers.CloudFunctionInvocationMetrics;
import org.and1ss.cloudbroker.service.user.UserService;
import org.springframework.data.util.Pair;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Objects;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Service
```

```
@Transactional
@RequiredArgsConstructor
public class CloudFunctionBrokerService {

    private final CloudBrokerRegistry cloudBrokerRegistry;
    private final UserService userService;
    private final CloudFunctionRepository cloudFunctionRepository;
    private final ExecutorService executorService =
Executors.newFixedThreadPool(100);

    public CloudFunction createFunction(String name,
                                        CloudFunctionInformation functionInformation,
                                        String testPayload,
                                        double[] weights) {

        final User user = userService.getCurrentUser();
        final String randomizedName = name + "-" + UUID.randomUUID();
        final String cloudProviderFunctionName = randomizedName.substring(0,
Math.min(randomizedName.length(), 64))
                .replace(' ', '-')
                .toLowerCase();

        final List<CloudFunctionInvocationMetrics> testInvocationMetrics =
                gatherTestInvocationsMetrics(functionInformation, testPayload);

        final String metricsReport = testInvocationMetrics.stream()
                .map(metrics -> "Provider: " + metrics.getBroker().getCloudProvider()
                        + ", config(vcpu: " +
metrics.getCloudBrokerConfiguration().getCpuCount()
```

```
                + ", ram: " + metrics.getCloudBrokerConfiguration().getRamSize()
                + "mb), cold start: " + metrics.getColdStartInvocationTime()
                + "ms, hot start: " + metrics.getHotStartInvocationTime()
                + ", est price: " + metrics.getPricePerInvocation() + "usd")
        .collect(Collectors.joining("\n"));

    System.out.println("--------------METRICS REPORT--------------");
    System.out.println(metricsReport);
    System.out.println("----------------------------------------");

    final Pair<CloudFunctionBroker, CloudBrokerConfiguration> bestChoise =
        CloudFunctionBrokerRanker.findBestConfiguration(testInvocationMetrics,
weights);

    final CloudFunctionBroker bestBroker = bestChoise.getFirst();
    final CloudBrokerConfiguration bestConfiguration = bestChoise.getSecond();

    bestBroker.createFunction(cloudProviderFunctionName, functionInformation,
bestConfiguration);

    try {
      final CloudFunction cloudFunction = new CloudFunction()
            .setName(name)
            .setCloudProviderName(cloudProviderFunctionName)
            .setCloudProvider(bestBroker.getCloudProvider())
            .setVCpu(bestConfiguration.getCpuCount())
            .setRam(bestConfiguration.getRamSize())
            .setRuntime(functionInformation.runtime())
            .setOwner(user);
```

```
            cloudFunctionRepository.save(cloudFunction);

            return cloudFunction;
        } catch (Exception e) {
            bestBroker.deleteFunction(cloudProviderFunctionName);
            throw e;
        }
    }

    private List<CloudFunctionInvocationMetrics> gatherTestInvocationsMetrics(
            CloudFunctionInformation functionInformation,
            String testPayload) {

        final List<Future<CloudFunctionInvocationMetrics>>
testInvocationMetricsFutures =
                cloudBrokerRegistry.getCloudBrokers().stream()
                        .flatMap(broker -> gatherTestInvocationsMetrics(broker,
functionInformation, testPayload))
                        .toList();

        return testInvocationMetricsFutures.stream()
                .map(future -> {
                    try {
                        return future.get();
                    } catch (InterruptedException | ExecutionException e) {
                        return null;
                    }
                })
                .filter(Objects::nonNull)
```

```
            .toList();
    }

    private Stream<Future<CloudFunctionInvocationMetrics>>
gatherTestInvocationsMetrics(
            CloudFunctionBroker broker,
            CloudFunctionInformation functionInformation,
            String testPayload) {

        return broker.getAvailableConfigurations().stream()
                .map(configuration ->
                        executorService.submit(() ->
                                gatherTestInvocationMetrics(
                                        broker, functionInformation, configuration, testPayload)));
    }

    private CloudFunctionInvocationMetrics
gatherTestInvocationMetrics(CloudFunctionBroker broker,
                                                    CloudFunctionInformation
functionInformation,
                                                    CloudBrokerConfiguration configuration,
                                                    String testPayload) {

        final String testName = "test-" + UUID.randomUUID();
        final String trimmedTestName = testName.substring(0,
Math.min(testName.length(), 64));

        System.out.println("Testing configuration: " + configuration);
        broker.createFunction(trimmedTestName, functionInformation, configuration);
```

```
        final long coldStartStartTime = System.currentTimeMillis();
        broker.invokeFunction(trimmedTestName, testPayload);
        final long coldStartInvocationTime = System.currentTimeMillis() -
coldStartStartTime;

        final long hotStartStartTime = System.currentTimeMillis();
        try {
            broker.invokeFunction(trimmedTestName, testPayload);
        } catch (final Throwable e) {
            broker.deleteFunction(trimmedTestName);
            throw new RuntimeException(e);
        }

        final long hotStartInvocationTime = System.currentTimeMillis() -
hotStartStartTime;

        System.out.println("Finished testing configuration: " + configuration);
        broker.deleteFunction(trimmedTestName);

        return CloudFunctionInvocationMetrics.of(
                coldStartInvocationTime,
                hotStartInvocationTime,
                configuration.getPricePerInvocation().apply(hotStartInvocationTime),
                configuration,
                broker);
    }
```

```
    private void deleteFunctionInCloudProvider(CloudProvider cloudProvider, String
cloudProviderFunctionName) {

cloudBrokerRegistry.getCloudBroker(cloudProvider).deleteFunction(cloudProviderFu
nctionName);
    }

    public void deleteFunction(String name) {
        final User user = userService.getCurrentUser();
        final CloudFunction cloudFunction =
cloudFunctionRepository.findByOwnerAndName(user, name).orElseThrow();
        deleteFunctionInCloudProvider(cloudFunction.getCloudProvider(),
cloudFunction.getCloudProviderName());
        cloudFunctionRepository.delete(cloudFunction);
    }

    public String invokeFunction(String name, String payload) {
        final User user = userService.getCurrentUser();
        final CloudFunction cloudFunction =
cloudFunctionRepository.findByOwnerAndName(user, name).orElseThrow();
        return cloudBrokerRegistry.getCloudBroker(cloudFunction.getCloudProvider())
                .invokeFunction(cloudFunction.getCloudProviderName(), payload);
    }

    public List<CloudFunction> listAllFunctions() {
        return cloudFunctionRepository.findAllByOwner(userService.getCurrentUser());
    }
}
```

CloudBrokerApplication.java

```
package org.and1ss.cloudbroker;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CloudBrokerApplication {

    public static void main(String[] args) {
        SpringApplication.run(CloudBrokerApplication.class, args);
    }

}
```

App.tsx

```
import './App.css'
import {createTheme, ThemeProvider} from "@mui/material";
import CloudFunctionList from "./CloudFunctionList.tsx";
import CustomAppBar from "./CustomAppBar.tsx";
import SplitPane from "react-split-pane";
import CloudFunctionDetails from "./CloudFunctionDetails.tsx";
import {CloudFunction} from "./CloudFunction.tsx";
import {useState} from "react";

const theme = createTheme({
    palette: {
        background: {
            default: '#fff',
            paper: '#fff'
```

```
    },
    primary: {
      light: '#757ce8',
      main: '#3f50b5',
      dark: '#002884',
      contrastText: '#fff',
    }
  },
});

function App() {
  const [selectedCloudFunction, setSelectedCloudFunction] =
useState<CloudFunction | null>(null)

  return (
    <ThemeProvider theme={theme}>
      <CustomAppBar/>
      <SplitPane
        allowResize={true}
        split="vertical"
        minSize={100}
        maxSize={-100}
        defaultSize={selectedCloudFunction !== null ? "65%" : "100%"}>
        <CloudFunctionList
          selectedCloudFunction={selectedCloudFunction}
          setSelectedCloudFunction={setSelectedCloudFunction}/>

        {
          selectedCloudFunction !== null
```

```
                ? <CloudFunctionDetails cloudFunction={selectedCloudFunction}/>
                : <div></div>
            }
        </SplitPane>
    </ThemeProvider>
  );
}

export default App
```

CloudFunction.tsx

```
export interface CloudFunction {
  name: string,
  cloudProvider: string,
  runtime: string,
  vcpu: number,
  ram: number
}
```

CloudFunctionDetails.tsx

```
import {Box, Button, Divider, Link, TextField, Typography} from "@mui/material";
import {CloudFunction} from "./CloudFunction.tsx";
import {useState} from "react";

interface CloudFunctionDetailsProps {
  cloudFunction: CloudFunction
}
```

```
const CloudFunctionDetails = (props: CloudFunctionDetailsProps) => {
    const cloudFunction = props.cloudFunction
    const url = "http://localhost:8080/api/v1/cloud-function/" + cloudFunction.name;

    const [tryFunctionPayload, setTryFunctionPayload] = useState('{"test": "payload"}')
    const [tryFunctionResult, setTryFunctionResult] = useState("");

    const tryFunction = () => {
        fetch(url, {
            method: "POST",
            headers: {
                "Authorization": "Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJuYW1lIjoiYWRtaW4iLCJle
HAiOjYwMTcxNjIwMzU1Mn0.Uo0aVASgqc15D70QU1oydqtTOBFCB_IC_QV4b
MpYQ4o"
            },
            body: tryFunctionPayload
        })
            .then(response => response.text())
            .then(body => setTryFunctionResult(body));
    }

    return (
        <Box
            padding="20px"
            height="100%"
            alignItems="left"
            justifyContent="left"
```

```
        bgcolor="white">
        <Box>
          <Typography variant="h5" color="primary.dark" align="left">
            Function: {cloudFunction.name}
          </Typography>

          <Divider sx={{"margin-top": 10}}/>

          <Typography variant="h6" color="primary.dark"
align="left">Configuration:</Typography>
          <Typography variant="body1" color="primary.dark" align="left">
            Cloud provider: {cloudFunction.cloudProvider}
          </Typography>
          <Typography variant="body1" color="primary.dark" align="left">vCPU:
{cloudFunction.vcpu}</Typography>
          <Typography variant="body1" color="primary.dark" align="left">RAM:
{cloudFunction.ram}MB</Typography>

          <Divider sx={{"margin-top": 10}}/>

          <Typography variant="h6" color="primary.dark" align="left">Function
URL:</Typography>
          <Typography variant="body1" color="primary.dark" align="left">The
function is accessible at:
            <Link href={url}>{url}</Link>, to invoke it, send POST request with
            payload in body.
          </Typography>
          <TextField
            required
```

```
                margin="normal"
                id="outlined-required"
                label="Payload"
                multiline
                fullWidth
                rows={4}
                value={tryFunctionPayload}
                onChange={(event) => setTryFunctionPayload(event.target.value)}
            />
            <TextField
                required
                inputProps={{readOnly: true}}
                margin="normal"
                id="outlined-required"
                label="Response"
                multiline
                fullWidth
                rows={4}
                defaultValue=''
                value={tryFunctionResult}
            />
            <Button variant="contained" onClick={tryFunction}>Try it out!</Button>
        </Box>
    </Box>
  );
}

export default CloudFunctionDetails;
```

CloudFunctionList.tsx

```
import {Box, Button, Divider, List, Typography} from "@mui/material";
import CloudFunctionListItem from "./CloudFunctionListItem.tsx";
import {CloudFunction} from "./CloudFunction.tsx";
import {Add} from "@mui/icons-material";
import {useEffect, useState} from "react";
import CreateCloudFunctionModal from "./CreateCloudFunctionModal.tsx";

interface CloudFunctionListProps {
    selectedCloudFunction: CloudFunction | null
    setSelectedCloudFunction: (cloudFunction: CloudFunction | null) => void
}

const fetchCloudFunctions = async (): Promise<CloudFunction[]> => {
    const response = await fetch("http://localhost:8080/api/v1/cloud-function", {
        method: "GET",
        headers: {
            "Authorization": "Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJuYW1lIjoiYWRtaW4iLCJle
HAiOjYwMTcxNjIwMzU1Mn0.Uo0aVASgqc15D70QU1oydqtTOBFCB_IC_QV4b
MpYQ4o"
        }
    })

    return (await response.json() as CloudFunction[])
}

const deleteCloudFunction = async (cloudFunction: CloudFunction): Promise<void>
=> {
```

```
    return await fetch("http://localhost:8080/api/v1/cloud-function/" +
cloudFunction.name, {
        method: "DELETE",
        headers: {
            "Authorization": "Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJuYW1lIjoiYWRtaW4iLCJle
HAiOjYwMTcxNjIwMzU1Mn0.Uo0aVASgqc15D70QU1oydqtTOBFCB_IC_QV4b
MpYQ4o"
        }
    }).then(_ => {
    })
}

const CloudFunctionList = (props: CloudFunctionListProps) => {
    const [cloudFunctions, setCloudFunctions] = useState<CloudFunction[]>([])
    const [createModalOpened, setCreateModalOpened] = useState(false)

    useEffect(() => {
        fetchCloudFunctions()
            .then(functions => setCloudFunctions(functions))
    }, [])

    const onFunctionCreated = (cloudFunction: CloudFunction) => {
        setCloudFunctions([...cloudFunctions, cloudFunction])
    }

    const onDeleteClick = (cloudFunction: CloudFunction) => {
        deleteCloudFunction(cloudFunction)
```

```
        .then(() => setCloudFunctions(cloudFunctions.filter(fun => fun !==
cloudFunction)))

      if (props.selectedCloudFunction === cloudFunction) {
        props.setSelectedCloudFunction(null)
      }
    }

    return (
      <Box>
        <CreateCloudFunctionModal
          opened={createModalOpened}
          setOpened={setCreateModalOpened}
          onFunctionCreated={onFunctionCreated}/>

        <Box display="flex" padding="10px">
          <Typography variant="h5" color="primary.dark" flexGrow="1" align="left"
padding="10px">
            Your cloud functions:
          </Typography>
          <Button
            variant="contained"
            color="success"
            endIcon={<Add/>}
            onClick={() => setCreateModalOpened(true)}>
            New
          </Button>
        </Box>
        <Divider/>
```

```
        <List>
            {
                cloudFunctions.map((cloudFunction, _) => (
                    <CloudFunctionListItem
                        key={cloudFunction.name}
                        cloudFunction={cloudFunction}
                        onDeleteClick={() => onDeleteClick(cloudFunction)}
                        onDetailsClick={() =>
props.setSelectedCloudFunction(cloudFunction)}
                    />))
            }
        </List>
    </Box>

  )
};

export default CloudFunctionList;
```

CloudFunctionListItem.tsx

```
import {Box, IconButton, Link, ListItem, Tooltip, Typography} from '@mui/material';
import {styled} from '@mui/system';
import {Delete, Info, PlayArrow} from "@mui/icons-material";
import {CloudFunction} from "./CloudFunction.tsx";

const ListItemContainer = styled(Box)(({}) => ({
  display: 'flex',
  alignItems: 'center',
```

```
    justifyContent: 'space-between',
    borderRadius: '10px',
    padding: '10px 15px',
    margin: 5,
    width: '100%'
}));

interface CloudFunctionListItemProps {
    cloudFunction: CloudFunction,
    onDeleteClick: () => void,
    onDetailsClick: () => void
}

const CloudFunctionListItem = (props: CloudFunctionListItemProps) => {
    return (
        <ListItem dense={true} alignItems="center" divider={true}>
            <ListItemContainer>
                <Link onClick={props.onDetailsClick} sx={{"min-width": '300px'}}>
                    <Typography variant="body1"
color="primary.dark">{props.cloudFunction.name}</Typography>
                </Link>
                <Box sx={{marginLeft: '50px'}}>
                    <Typography variant="caption"
color="primary.dark">Provider:</Typography>
                    <Typography variant="subtitle1"
color="primary.dark">{props.cloudFunction.cloudProvider}</Typography>
                </Box>
                <Box sx={{marginLeft: '10px'}}>
```

```
                <Typography variant="caption"
color="primary.dark">Runtime:</Typography>
                <Typography variant="subtitle1"
color="primary.dark">{props.cloudFunction.runtime}</Typography>
            </Box>
            <Box>
                <Tooltip title="Delete" sx={{marginLeft: '15px'}}>
                    <IconButton color="error" onClick={props.onDeleteClick}>
                        <Delete/>
                    </IconButton>
                </Tooltip>
                <Tooltip title="Run">
                    <IconButton color="success" onClick={props.onDetailsClick}>
                        <PlayArrow/>
                    </IconButton>
                </Tooltip>
                <Tooltip title="Details" onClick={props.onDetailsClick}>
                    <IconButton>
                        <Info/>
                    </IconButton>
                </Tooltip>
            </Box>
        </ListItemContainer>
    </ListItem>
  )
}

export default CloudFunctionListItem;
```

CreateCloudFunctionModal.tsx

```
import {
    Backdrop,
    Box,
    Button,
    CircularProgress,
    Fade,
    FormControl,
    InputLabel,
    MenuItem,
    Modal,
    Select,
    TextField,
    Typography
} from "@mui/material";
import {useState} from "react";
import {CloudFunction} from "./CloudFunction.tsx";

interface CreateCloudFunctionModalProps {
    opened: boolean,
    setOpened: (opened: boolean) => void,
    onFunctionCreated: (cloudFunction: CloudFunction) => void
}

const style = {
    position: 'absolute' as 'absolute',
    top: '50%',
    left: '50%',
```

```
    transform: 'translate(-50%, -50%)',
    width: 400,
    bgcolor: 'background.paper',
    borderRadius: 2,
    boxShadow: 24,
    p: 4,
};

const availableRuntimes: string[] = [
    "Java 17"
]

interface CreateFunctionRequest {
    name: string,
    handler: string,
    runtime: string,
    testPayload: string,
    weights: string
}

const makeCreateFunctionRequest = async (request: CreateFunctionRequest,
executable: File): Promise<CloudFunction> => {
    const sum = request.weights.split(",")
        .map(weight => Number(weight.trim().replace(/\s+/g, "")))
        .reduce((sum, num) => sum + num, 0)

    if (Math.abs(sum - 1) > 0.05)
        return Promise.reject("Sum of weights should be 1");
```

```
    const requestData = new FormData();

    requestData.append(
        "createRequest", new Blob([JSON.stringify(request)], {type:
"application/json"}));
    requestData.append("file", executable);


    const response = await fetch("http://localhost:8080/api/v1/cloud-function", {
        method: "POST",
        headers: {
            "Authorization": "Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInVzZXJuYW1lIjoiYWRtaW4iLCJle
HAiOjYwMTcxNjIwMzU1Mn0.Uo0aVASgqc15D70QU1oydqtTOBFCB_IC_QV4b
MpYQ4o"
        },
        body: requestData
    });

    return (await response.json()) as CloudFunction
}

const CreateCloudFunctionModal = (props: CreateCloudFunctionModalProps) => {
    const [functionName, setFunctionName] = useState<string | null>(null)
    const [runtime, setRuntime] = useState(availableRuntimes[0])
    const [handler, setHandler] = useState<string | null>(null)
    const [testPayload, setTestPayload] = useState<string>('{"key": "value"}')
    const [weights, setWeights] = useState<string>("0, 0, 0, 0, 1")
```

```
const [selectedFile, setSelectedFile] = useState(undefined)
const [loading, setLoading] = useState(false)

const onCreateClick = () => {
    setLoading(true)
    makeCreateFunctionRequest({
        name: functionName!!,
        runtime: runtime,
        handler: handler!!,
        testPayload: testPayload,
        weights: weights
    }, selectedFile!!)
        .then(cloudFunction => {
            props.onFunctionCreated(cloudFunction)
            props.setOpened(false)
        })
}

return (
    <Modal
        aria-labelledby="spring-modal-title"
        aria-describedby="spring-modal-description"
        open={props.opened}
        onClose={() => props.setOpened(false)}
        closeAfterTransition
        slots={{backdrop: Backdrop}}
        slotProps={{
            backdrop: {
                TransitionComponent: Fade,
```

```
            },
          }}
        >
          <Fade in={props.opened}>
            <Box sx={style}>

              <Typography variant="h6" color="primary.main"
marginBottom="5px">Name:</Typography>
              <TextField
                required
                label="Name"
                fullWidth={true}
                onChange={(event) => setFunctionName(event.target.value)}/>

              <Typography variant="h6" color="primary.main"
marginTop="10px">Runtime:</Typography>
              <FormControl variant="filled" fullWidth={true}>
                <InputLabel id="demo-simple-select-filled-
label">Runtime</InputLabel>
                <Select
                  labelId="demo-simple-select-filled-label"
                  id="demo-simple-select-filled"
                  value={runtime}
                  onChange={setRuntime}
                >
                  {
                    availableRuntimes.map(
                      runtime => <MenuItem key={runtime}
value={runtime}>{runtime}</MenuItem>)
```

```
            }
          </Select>
        </FormControl>

        <Typography variant="h6" color="primary.main" marginTop="10px"
              marginBottom="5px">Handler:</Typography>
        <TextField
          required
          label="Handler"
          fullWidth={true}
          onChange={(event) => setHandler(event.target.value)}/>

        <Typography variant="h6" color="primary.main" marginTop="10px"
              marginBottom="5px">Executable:</Typography>
        <TextField
          type="file"
          fullWidth={true}
          onChange={(event) => {
            setSelectedFile(event.target.files[0])
            console.log(event.target.files[0])
          }}
        />

        <Typography variant="h6" color="primary.main"
marginTop="10px">Weights</Typography>
        <Typography variant="body1" color="primary.main"> (price, cpu, ram,
cold start time, hot start
          time):</Typography>
        <TextField
```

```
                required
                label=""
                fullWidth={true}
                defaultValue={weights}
                onChange={(event) => setWeights(event.target.value)}/>

            <Typography variant="h6" color="primary.main"
marginTop="10px">Test payload:</Typography>
            <TextField
                required
                id="outlined-required"
                label="Test payload"
                multiline
                fullWidth
                rows={4}
                contentEditable={false}
                margin="normal"
                defaultValue='{"key": "value"}'
                onChange={(event) => setTestPayload(event.target.value)}
            />
            <Button
                variant="contained"
                color="success"
                onClick={onCreateClick}
                fullWidth={true}
                disabled={loading}>
                Create
            </Button>
```

```
                {
                    loading
                        ? <Box width="50%" marginLeft="45%" marginTop="10px">
                            <CircularProgress/>
                        </Box>
                        : <></>
                }

            </Box>
        </Fade>
    </Modal>
  )
};

export default CreateCloudFunctionModal;
```

CustomAppBar.tsx

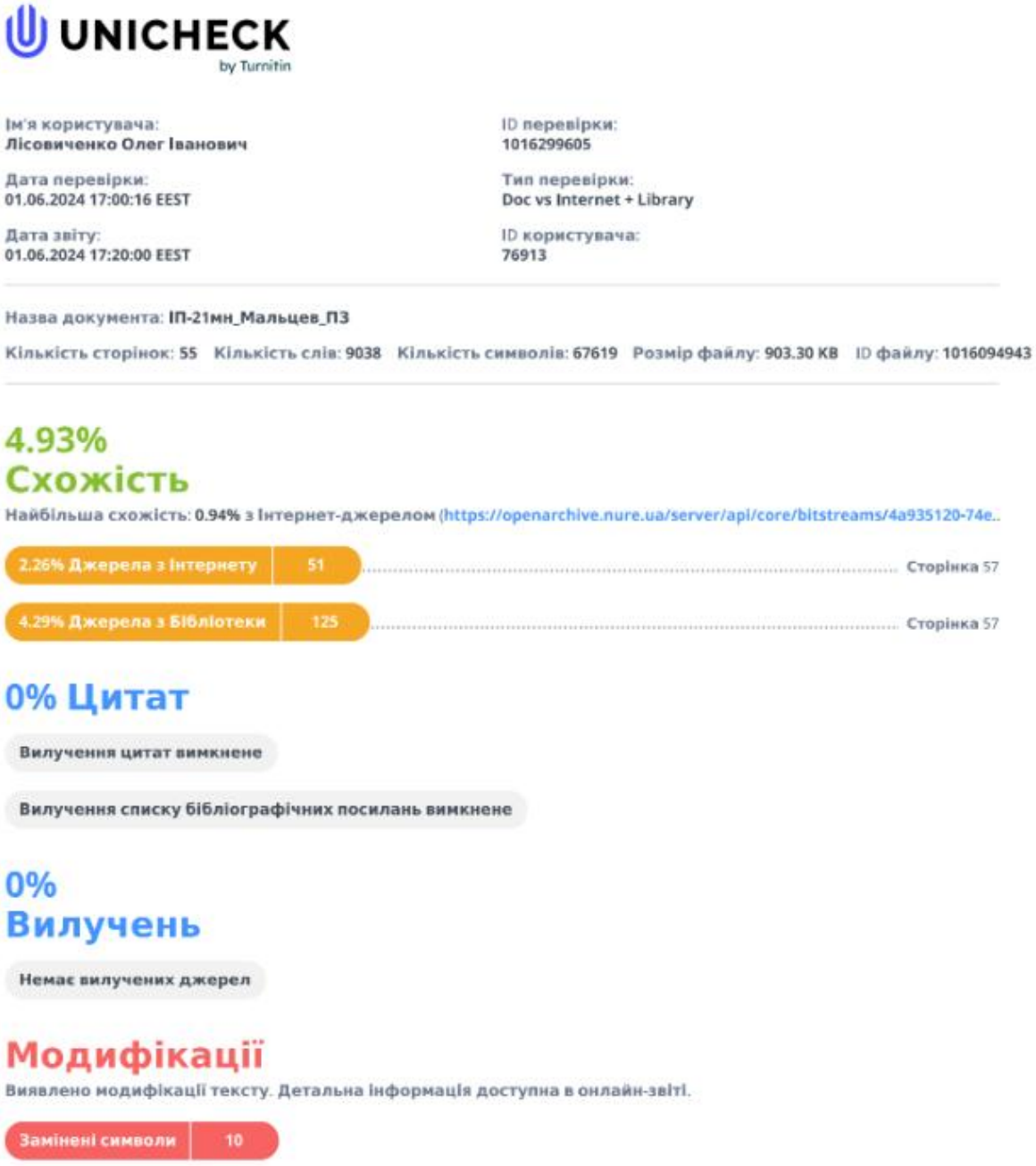
```
import {AppBar, Box, Button, Toolbar, Typography} from "@mui/material";

const CustomAppBar = () => {
  return (
    <Box sx={{flexGrow: 1}}>
      <AppBar position="static">
        <Toolbar variant="dense">
          <Typography variant="h6" color="inherit" component="div"
flexGrow="1" align="left">
            Cloud Broker
          </Typography>
```

```
                <Button variant="contained" color="error">LOG OUT</Button>
            </Toolbar>
        </AppBar>
    </Box>
  );
}

export default CustomAppBar;
```

Main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

ДОДАТОК Б РЕЗУЛЬТАТ ПЕРЕВІРКИ РОБОТИ НА СПІВПАДІННЯ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016299605

Дата перевірки:
01.06.2024 17:00:16 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2024 17:20:00 EEST

ID користувача:
76913

Назва документа: ІП-21мн_Мальцев_ПЗ

Кількість сторінок: 55 Кількість слів: 9038 Кількість символів: 67619 Розмір файлу: 903.30 KB ID файлу: 1016094943

4.93%
Схожість

Найбільша схожість: 0.94% з Інтернет-джерелом (https://openarchive.nure.ua/server/api/core/bitstreams/4a935120-74e..

2.26% Джерела з Інтернету 51 ..... Сторінка 57

4.29% Джерела з Бібліотеки 125 ..... Сторінка 57

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 10