Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем»
за освітньою програмою: «12 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ20130»

	(підпис студента)	/Вячеслав ПЕСОЦЬКИЙ/ (Ім'я ПРІЗВИЩЕ)
Керівник:	(підпис)	/Владислав СКАЛОЗУБ/ (посада, Ім'я ПРІЗВИЩЕ)
Нормоконтролер:	(підпис)	/Світлана ВОЛКОВА / (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____ (підпис)

Дніпро – 2023

Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Спеціальність: «Інженерія програмного забезпечення»

«ДО ЗАХИСТУ»
Завідувач кафедри
_____ Вадим ГОРЯЧКІН
(підпис) (ПІБ)
202_ р. _____ «_____»

ЗАВДАННЯ

до дипломної роботи на здобуття ОС «бакалавр»

студента групи _____ Песоцький Вячеслав Андрійович
номер групи (ПІБ)

1. Тема роботи: «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем»
затверджена наказом по університету від «___»_____202_ р. № ___ ст.

2. Термін подання студентом роботи «20» червня 2023 р.

3. Вихідні дані до дипломної роботи Приклади постановок завдань багатокритеріального аналізу та оптимізації складних систем, прототипи складних систем і сфера їх застосування, особливості завдань і спеціалізовані математичні моделі багатокритеріального аналізу, основні алгоритми багатокритеріального аналізу, область компромісних рішень її властивості, програмні засоби реалізації методів багатокритеріального аналізу та оптимізації складних систем, поняття рівномірної апроксимації областей аналізу параметрів систем.

4. Зміст пояснювальної записки (перелік питань до розробки):

4.1 Аналітична частина: огляд досліджень та розробок щодо можливостей реалізації завдань вибору областей компромісних рішень, процедури застосування даних прототипів для формування математичних моделей багатокритеріального вибору, поняття та властивості ЛП-послідовностей, структура завдань та процедури формування моделей для застосування паралельних алгоритмів багатокритеріального аналізу, визначення можливостей удосконалення методів формування моделей компромісу на основі прототипів та ЛП-послідовностей.

4.2 Основна частина: розробка удосконаленої моделі та процедури, призначених для визначення оцінок коефіцієнтів важливості показників

векторних моделей оптимізації складних багато параметричних систем на основі використання даних прототипів, розробка удосконаленої моделі та нової процедура, призначеної для забезпечення паралельного зондування областей компромісно-оптимальних рішень (ефективність за Парето) наборами скалярних моделей оптимізації; розробка спеціалізованої евристичної процедури для забезпечення паралельного зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування; розробка програмного забезпечення щодо реалізації запропонованих процедур багатокритеріальної оптимізації та дослідження числових результатів аналізу моделей, проведення досліджень для підтвердження достовірності та ефективності програмних рішень.

5. Перелік демонстраційного матеріалу: презентація результатів розробки програмного забезпечення, результати чисельних експериментальних досліджень алгоритмів і процедур, програмні засоби для реалізації алгоритмів і процедур багатокритеріального аналізу та оптимізації складних систем, відео-демонстрація функціонування програмного комплексу.

КАЛЕНДАРНИЙ ПЛАН

№ пор	Зміст роботи (розділу)	Термін виконання розділів роботи	Примітка
1	Вступ		
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами		
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження		
4	Постановка задачі, технічне завдання		
5	Техніко-економічні показники		
6	Розробка інструментальних засобів дослідження		
7	Виконання досліджень		
8	Оформлення результатів дипломної роботи		
9	Подання дипломної роботи до кафедри		
10	Захист дипломної роботи на засіданні екзаменаційної комісії		

Дата видачі завдання «___»_____ 202_ р.

Керівник дипломної роботи _____ Владислав СКАЛОЗУБ
(підпис) (ПІБ)

Завдання прийняв до виконання _____

РЕФЕРАТ

Об’єктом дослідження та розробки дипломної роботи є математичні моделі і паралельні алгоритми, призначені для вирішення завдань багатокритеріального аналізу та оптимізації складних систем шляхом визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації складних багато параметричних систем на основі використання даних прототипів створюваних об’єктів.

Мета роботи полягала в удосконаленні моделі та процедури із визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів; в удосконаленні моделі та створенні нової процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей; у розробленні спеціалізованої процедури яка забезпечує паралельне зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування; у розробленні програмне забезпечення для процедур багатокритеріальної оптимізації та дослідженні числових результатів аналізу запропонованих моделей.

Для досягнення мети дослідження та розробок були застосовані методи системного аналізу, математичного моделювання, моделі і методи багатокритеріальної оптимізації, методи програмної інженерії, призначені для формування вимог, проектування та розробки програмного забезпечення.

В результаті виконання дипломної роботи були удосконалені моделі та процедури, призначені для визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації за рахунок даних прототипів створюваних об’єктів, удосконалена процедура формування області Парето, що забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей, а також розроблене програмне забезпечення щодо

багатокритеріальної оптимізації і дослідженні результатів аналізу запропонованих моделей, які підтвердили достовірність та ефективність отриманих рішень.

Розрахунково-пояснювальна записка складається зі вступу, 4 розділів, висновків, бібліографічного списку та додатків.

ЗМІСТ

ВСТУП

Актуальність роботи

При створенні чи удосконаленні складних, розподілених або багато функціональних систем широкого призначення, а також широкого кола таких технологій, в багатьох випадках мають місце невизначеність умов застосування, необхідність узгодження конкуруючих вимог, а також узгодження наборів показників, відзначається відсутність загальних моделей створюваних об’єктів (систем). Разом з тим розробникам відомі попередні подібні системи і технології, які є прототипами створюваних систем. Для цих прототипів вже існують дані, оцінки функціональних ознак та значення наборів цільових показників систем. Наявність вектору цільових показників що оптимізуються (ВО) являється типовою ситуацію при проєктуванні та оптимізації складних систем, що характеризує системну невизначеність цілі. Одним із головних методів, а також засобом моделювання та відображення завдань ВО являється формування області компромісно-оптимальних рішень, область Парето. Для формування таких областей, як правило, необхідно вирішувати значну кількість одноцільвих завдань багатопараметричної оптимізації. Таке завдання має велику обчислювальну складність. Тому удосконалення моделей та алгоритмів, які дозволяють підвищити точність побудови як моделі області Парето, так числову ефективність процедур її формування має значний теоретичний і практичний інтерес, відповідає сучасним розробкам цього напряму.

У зв’язку з вище зазначеним тема та об’єкт дослідження та розробки дипломної роботи, як є математичні моделі і паралельні алгоритми, призначені для вирішення завдань багатокритеріального аналізу та оптимізації складних систем шляхом визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації складних багато параметричних систем на

основі використання даних прототипів створюваних об’єктів, являються актуальними.

Мета роботи

Мета дипломної роботи полягала в удосконаленні математичних моделей і паралельних алгоритмів, які призначені для багатокритеріального аналізу (БКА) та оптимізації складних систем, а також розробки відповідних програмних засобів. Особливість удосконалених моделей і процедур полягає у вирішенні завдання щодо визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів. Наступне завдання БКА - удосконалення моделі та створення нової процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей. В роботі також необхідно було розробити спеціалізовану процедуру для забезпечення паралельного зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування, а також створити відповідне програмне забезпечення, призначене для процедур БКА та дослідженні числових результатів аналізу запропонованих моделей.

Для досягнення мети дипломної роботи були застосовані методи системного аналізу, математичного моделювання, моделі і методи багатокритеріальної оптимізації, методи програмної інженерії, призначені для формування вимог, проектування та розробки програмного забезпечення.

Експлуатаційне призначення.

Призначення результатів розробки обумовлене новими можливостями застосування удосконалених моделей і процедур формування паралельних алгоритмів та відповідних програмних засобів БЛА, що дозволяє підвищити

числову ефективність та точність апроксимації областей Парето за рахунок використання рівномірних $ЛП_{тау}$ послідовностей.

Створення спеціалізованих програмних засобів для БЛА щодо автоматизації завдань ефективного формування областей компромісно-оптимальних рішень на основі наборів аналогів об’єкту проектування, визначає експлуатаційне призначення розробки.

РОЗДІЛ 1. ЗБІР ТА АНАЛІЗ ВИМОГ

1.1 Аналіз наукової та технологічної новизни

Удосконалена модель та процедура, призначена для визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації складних багато параметричних систем на основі використання даних прототипів створюваних об’єктів, яка відрізняється структурою критерію скаляризації (maxmin) та алгоритмом і процедурою розрахунку, що дозволяє оцінювати коефіцієнтів важливості на основі багатьох прототипів і не залежить від кількості параметрів об’єкту проектування.

Удосконалена модель та створена нова процедура, призначена для забезпечення паралельного зондування областей компромісно-оптимальних рішень (ефективність за Парето) наборами скалярних моделей оптимізації[1], яка відрізняється використанням рівномірних $ЛП_{тау}$ послідовностей[6], що підвищує точність апроксимації областей Парето.

Розроблена спеціалізована евристична процедура, призначена для забезпечення паралельного зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування.

Розроблене програмне забезпечення щодо реалізації запропонованих процедур багатокритеріальної оптимізації та дослідження числових результатів аналізу моделей підтвердили достовірність та ефективність отриманих рішень.

1.2 Постановка задачі

Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем використовуються для ефективного аналізу та оптимізації складних систем, які мають багато критеріїв[2].

Задача полягає в розробці математичних моделей, які враховують взаємозв'язки між компонентами системи та їх вплив на критерії оптимізації. Ці моделі використовуються для розробки багатокритеріальних алгоритмів аналізу та оптимізації, які забезпечують пошук найкращих рішень за визначеними критеріями.

Однак, для ефективного вирішення таких задач потрібні великі обчислювальні ресурси. Тому розробляються паралельні алгоритми, які дозволяють ефективно використовувати ресурси обчислювальної системи та прискорювати процес аналізу та оптимізації складних систем.

Окрім того, розробляється методика оцінки ефективності та якості рішень, що будуть отримані з використанням розроблених моделей та алгоритмів. Дослідження, пов'язані з моделями і паралельними алгоритмами багатокритеріального аналізу та оптимізації складних систем[1], дозволяють забезпечити більш точний аналіз та ефективну оптимізацію різноманітних систем, що мають багато критеріїв.

Висновки за розділом 1

У розділі проведено огляд досліджень та розробок щодо можливостей реалізації завдань вибору областей компромісних рішень, процедури застосування даних прототипів для формування математичних моделей багатокритеріального вибору. Визначено поняття та властивості ЛП-послідовностей, визначені структура завдань та процедури формування моделей для застосування паралельних алгоритмів багатокритеріального аналізу. На підставі аналізу встановлені можливості щодо удосконалення методів формування моделей компромісу на основі прототипів та ЛП-послідовностей.

РОЗДІЛ 2. ПРОЕКТУВАННЯ

2.1 Задачі проекту

- Збір вихідних даних:
  - Вивчення літератури та існуючих досліджень з багатокритеріального аналізу та оптимізації складних систем.
  - Аналіз методів та моделей, що використовуються для моделювання складних систем та їх оптимізації.
  - Визначення основних критеріїв, які необхідно оптимізувати в рамках проекту.
- Розробка математичної моделі:
  - Розробка математичної моделі, яка враховуватиме взаємозв'язки між компонентами складної системи та їх вплив на критерії оптимізації.
  - Удосконалення моделі та процедури із визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів;
  - Удосконалення моделі та створенні нової процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей;
  - Розроблення спеціалізованої процедури яка забезпечує паралельне зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування;
  - Розробка програмних процедур багатокритеріальної оптимізації та дослідженні числових результатів аналізу запропонованих моделей.

    - Визначення параметрів та функціоналів, що характеризують кожен компонент системи.
- Розробка багатокритеріальних алгоритмів:
    - Розробка багатокритеріальних алгоритмів аналізу та оптимізації[4], які забезпечують пошук Парето оптимальних (компромісних) рішень за визначеними критеріями.
    - Врахування взаємозв'язків між компонентами вектору показників та їх вплив на багатокритеріальну модель проектування.
- Розробка паралельних алгоритмів:
    - Розробка паралельних алгоритмів, що забезпечують рівномірне зондування областей компромісів, а також ефективне використання ресурсів обчислювальної системи для прискорення процесу аналізу та оптимізації складних систем.
    - Врахування особливостей паралельного обчислення та розподіленої обробки даних.
- Валідація та експериментальне дослідження:
    - Валідація розробленої математичної моделі та багатокритеріальних алгоритмів на реальних або симульованих даних.
    - Проведення експериментальних досліджень для оцінки ефективності та якості рішень, отриманих з використанням розроблених моделей та алгоритмів.
- Аналіз результатів та висновки:
    - Аналіз отриманих результатів експериментів та порівняння їх з існуючими методами та моделями.
    - Формулювання висновків щодо ефективності та застосовуваності розроблених моделей і алгоритмів.
    - Виявлення можливостей подальшого розширення та вдосконалення досліджуваної теми.

Загалом, проект з моделей і паралельних алгоритмів багатокритеріального аналізу та оптимізації складних систем включає розробку моделей, алгоритмів та їх експериментальну перевірку з метою забезпечення ефективного аналізу та оптимізації складних систем з багатьма критеріями.

2.2 Функціональні вимоги

- Завантаження та імпорт даних:
  - Можливість завантаження даних зовнішніми користувачами з бази даних.
  - Механізм імпорту даних з баз даних.
- Управління проектами:
  - Створення та управління проектами для проведення багатокритеріального аналізу та оптимізації складних систем.
  - Додавання, видалення та редагування проектів.
  - Призначення доступу до проектів для різних користувачів.
- Введення та обробка даних:
  - Можливість введення та редагування даних, необхідних для аналізу та оптимізації.
  - Реалізація алгоритмів багатокритеріального аналізу та оптимізації для обробки даних.
  - Використання паралельних алгоритмів для прискорення обчислень.
- Візуалізація та звітність:
  - Генерація візуальних даних, зображень результатів аналізу та оптимізації.

    - Створення звітів з результатами для зручного представлення користувачам.

- Інтеграція:
    - API для інтеграції з іншими системами або сервісами.
- Управління користувачами:
    - Можливість управління користувачами (створення, видалення, редагування).
    - Налаштування рівнів доступу та прав користувачів.
- Пошук та фільтрація даних:
    - Функціонал пошуку і фільтрації даних, що дозволяє користувачам швидко знаходити необхідну інформацію в системі.

2.3 Вибір мови програмування

Для написання даного програмного продукту була використана одна мова програмування JavaScript. Програмний продукт складається з клієнт-серверної архітектури, а саме з сайту – це зовнішній інтерфейс програми, та API – серверної частини програми.

JavaScript - це мова програмування, яку використовують розробники для створення інтерактивних веб-сторінок[10]. Функції JavaScript можуть поліпшити зручність взаємодії користувача з веб-сайтом: від оновлення стрічки новин у соціальних мережах і до відображення анімації та інтерактивних карт. JavaScript є мовою програмування під час розроблення скриптів для виконання на стороні клієнта, що робить його однією з базових технологій у всесвітній мережі Інтернет. Наприклад, карусель зображення, меню, що випадає за кліком, і динамічно мінливі кольори елементів на веб-сторінці, які ви бачите під час перегляду сторінок в Інтернеті, виконані за допомогою JavaScript.

Для написання інтерфейсу користувача було обрано мову програмування JavaScript, та фреймворк React.

React - відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб сторінки, з якими стикаються в розробці односторінкових застосунків[19]. Розробляється Meta (Facebook) і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови вебзастосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

Для написання API було використано мову NodeJS, та фреймворк NestJS.

Node.js є відкритою платформою для розробки серверних додатків на JavaScript[16]. Вона побудована на двигуні V8, розробленому Google для браузера Chrome, що забезпечує високу продуктивність та швидкість виконання JavaScript-коду. Node.js відрізняється від традиційного використання JavaScript у веб-браузері, оскільки воно дозволяє виконувати JavaScript на сервері.

Основні характеристики Node.js:

- Не відбувається блокування операцій вводу/виводу (non-blocking I/O): Основна філософія Node.js полягає в тому, щоб уникати блокування

операцій вводу/виводу. Це дозволяє одночасно обробляти багато запитів, що забезпечує високу продуктивність та швидкість виконання додатків.

- Подієва модель програмування (event-driven programming): Node.js використовує подієву модель програмування, де додаток реагує на події та виконує відповідні обробники. Це дозволяє створювати швидкі та масштабовані додатки.
- Пакетний менеджер npm: Node.js постачається з пакетним менеджером npm (Node Package Manager), який є найбільшою колекцією безкоштовних, відкритих модулів JavaScript. npm дозволяє легко встановлювати, оновлювати та управляти залежностями проекту.
- Розширені можливості розробки веб-додатків: Node.js забезпечує багато вбудованих модулів, які спрощують розробку веб-додатків. Наприклад, модуль http дозволяє створювати HTTP-сервери, а модуль fs дозволяє взаємодіяти з файловою системою.
- Підтримка платформи: Node.js підтримує багатоопераційні системи, такі як Windows, macOS та різні дистрибутиви Linux. Це робить його універсальним і зручним для розробки на різних середовищах.
- Розширюваність: За допомогою Node.js можна розробляти не тільки веб-сервери, але і різні типи додатків, такі як мережеві програми, інструменти командного рядка, роботи з базами даних тощо. Крім того, за допомогою розширення Node.js API можна підключати сторонні бібліотеки, що дозволяє використовувати великий вибір інструментів та функціональності.

Node.js здобув значну популярність у розробників завдяки своїм перевагам, зокрема високій продуктивності, ефективному використанню ресурсів, широкій підтримці спільноти та багатому екосистемі пакетів. Він використовується багатьма великими компаніями для створення високонавантажених серверних додатків, мікросервісів та інших додатків, які вимагають швидкого та масштабованого розвитку.

NestJS є прогресивним, модульним та масштабованим фреймворком для розробки серверних додатків на Node.js[17]. Він базується на засадах архітектури Angular і використовує TypeScript для розробки додатків. NestJS пропонує елегантну синтаксичну модель, що полегшує створення складних серверних додатків.

Основні особливості та переваги NestJS:

- Архітектура модулів: NestJS пропонує модульну архітектуру, яка дозволяє розбити додаток на окремі функціональні модулі. Це сприяє збереженню чистоти коду, полегшує перевикористання та розширення функціональності.
- Провідні принципи SOLID: NestJS дотримується принципів SOLID (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), що сприяє створенню масштабованих, легко змінюваних та тестових додатків.
- Використання TypeScript: NestJS повністю підтримує TypeScript, що дозволяє розробникам використовувати сильну типізацію, робити рефакторинг легше та отримувати переваги інструментів, таких як автодоповнення коду та перевірка типів під час розробки.
- Декоратори та Метадані: NestJS використовує декоратори та метадані для опису різних аспектів додатка, таких як маршрутизація, обробники запитів, валідація даних та інші аспекти. Це дозволяє розробникам зосередитися на бізнес-логіці, зменшуючи необхідність в ручному налаштуванні та конфігурації.
- HTTP і WebSockets підтримка: NestJS надає вбудовану підтримку для обробки HTTP-запитів та використання WebSockets для реалізації багаторівневих та реактивних додатків.
- Middleware: За допомогою middleware, NestJS дозволяє виконувати додаткову обробку запитів, наприклад, автентифікацію,

авторизацію, логування тощо. Це забезпечує більш гнучкий контроль над потоком даних у додатку.

- Інтеграція з базами даних: NestJS підтримує інтеграцію з різними базами даних, включаючи SQL (наприклад, PostgreSQL, MySQL), NoSQL (наприклад, MongoDB) та інші. Завдяки використанню ORM (Object-Relational Mapping) та ODM (Object-Document Mapping) бібліотек, таких як TypeORM або Mongoose, легко працювати з базами даних у NestJS.
- Тестування: NestJS полегшує тестування додатків, надаючи вбудовану підтримку для модульних, інтеграційних та e2e (end-to-end) тестів. Це допомагає забезпечити якість коду та впевненість у функціональності додатку.

NestJS має активну спільноту розробників та широкий екосистему пакетів, що дозволяє легко розширювати функціональність та використовувати різні рішення в розробці додатків. Він є потужним фреймворком для будь-яких серверних додатків на Node.js, незалежно від їх розміру та складності.

При роботі з даними було використано реляційну базу даних PostgreSQL.

PostgreSQL - це потужна об'єктно-реляційна система керування базами даних (СКБД), яка надає розширені можливості для зберігання, керування та маніпулювання даними[18]. Особливість PostgreSQL полягає у його відкритості, стабільності та високій продуктивності, що робить його одним з найпопулярніших СКБД у світі. Ось деякі ключові аспекти та функціональність PostgreSQL:

- Реляційна модель: PostgreSQL використовує реляційну модель баз даних, що дозволяє організовувати дані у відношеннях (таблицях) зі структурованими зв'язками між ними. Це забезпечує консистентність, цілісність та надійність даних.

- Розширені типи даних: PostgreSQL має багатий набір вбудованих типів даних, включаючи числа, рядки, дата/час, булеві значення, JSON, XML, географічні дані та багато інших. Крім того, ви можете визначати та використовувати власні типи даних та функції.
- Мови програмування та розширення: PostgreSQL підтримує різні мови програмування, такі як SQL, PL/pgSQL, Python, JavaScript, Perl, Ruby та інші. Це дозволяє розробникам реалізовувати бізнес-логіку безпосередньо на рівні бази даних. Крім того, ви можете створювати власні розширення та функції, що розширюють можливості бази даних.
- Транзакційна безпека: PostgreSQL гарантує транзакційну безпеку, що означає, що ви можете виконувати групу операцій як єдину атомарну операцію. У разі виникнення помилки, транзакція може бути відкатана (rollback), що дозволяє зберегти цілісність даних.
- Реплікація та висока доступність: PostgreSQL підтримує різні методи реплікації для забезпечення резервного копіювання, балансування навантаження та високої доступності. Ви можете налаштувати майстер-систему та декілька підчинених систем для автоматичного розподілу навантаження та забезпечення надійності системи.
- Індексування та оптимізація запитів: PostgreSQL надає широкі можливості для створення індексів, що покращують продуктивність запитів. Він також підтримує різні методи оптимізації запитів, такі як використання попередньо скомпільованих запитів та статистики для покращення продуктивності.
- Безпека та ролева модель: PostgreSQL має потужну ролеву модель, що дозволяє налаштувати доступ до об'єктів бази даних на рівні користувачів та ролей. Ви можете встановлювати права доступу до таблиць, схем, функцій та інших об'єктів, забезпечуючи безпеку даних.
- Розширюваність: PostgreSQL є високорозширюваною системою, яка дозволяє розробникам створювати власні розширення, функції та типи

даних. Ви можете розширити можливості бази даних, додавши власну логіку та функціональність.

PostgreSQL є вільним та відкритим програмним забезпеченням з активною спільнотою розробників, яка забезпечує підтримку, оновлення та розвиток системи. Він є популярним вибором для різноманітних застосувань, від невеликих проектів до великих підприємственних систем.

2.4 Властивості рівномірних $ЛП_{тау}$ послідовностей

Серед відомих нині розподільних послідовностей найкращі характеристики рівномірності як при $N \to \infty$, так і при малих значеннях N - у так званих $ЛП_{тау}$ послідовностей[6]. Точки однієї з таких послідовностей P0, P1, K, Pr, K легко обчислюються за допомогою невеликої таблиці допоміжних напрямних точок. Так, таблиця, що містить 20 напрямних точок, дає змогу обчислити 220, тобто понад мільйон, точок $ЛП_{тау}$ послідовностей. Походження назви досить курйозне. Спершу вдалося побудувати сітки, що складалися з кінцевого числа розподільних точок у K^n , і отримали назву $П_{тау}$ сіток. Потім було побудовано нескінченні послідовності таких точок, будь-яка двійкова (тобто така, що містить по 2k точок) ділянка яких є $П_{тау}$ сітка.

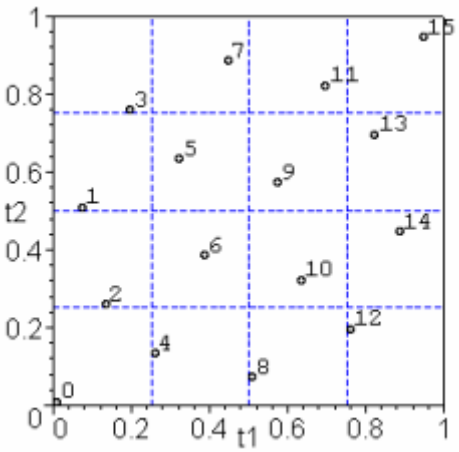

Рис. 2.1 $ЛП_{тау}$ сітка

Звідси термін - $ЛП_{тау}$ послідовностей. Там же наводиться досить простий арифметичний алгоритм:

$$x^i \in \mathbf{K}^n \subset \mathrm{R}^n,\ i = 0, 1, 2, \ldots$$

$$x_j^i = \sum_{k=1}^{m} 2^{-k+1} \left\{ \frac{1}{2} \sum_{l=k}^{m} \left[2\left\{ i\, 2^{-l} \right\} \right] \left[2\left\{ r_j^l\, 2^{k-1-l} \right\} \right] \right\}$$

Тут m = 1+ [ln i / ln 2], причому [z] - ціла частина, {z} - дробова частина числа z, r_j^l - заздалегідь обчислені числа (чисельники «напрямних чисел»). При цьому j визначає розмірність, 2^l - l - число точок. Числа r_j^l відомі для 20 $1 \le j \le 51$, $1 \le l \le 20$, що дає змогу обчислити понад мільйон квазівипадкових точок розмірності до 51.

Для всіх $ЛП_{тау}$ послідовностей і для всіх N справедлива оцінка відхилення (5). Більше того, якщо розглядати тільки значення $K^n = 2$, k =1, 2, …, то справедлива і сильніша оцінка (5а). Усі $ЛП_{тау}$ послідовності задовольняють і тій вимозі, що проекції їхніх точок на будь-яку s-вимірну грань n K також утворюють s-вимірну $ЛП_{тау}$ послідовність ($1 \le s \le n - 1$). Описана в (8) $ЛП_{тау}$-послідовність має в деякому сенсі добре розподілені початкові відрізки 0 1 1 , , , P P K PN – для довільного N, навіть за n N < 2 . Одна з таких властивостей така. Розглянемо гіперплощини x j = 1/ 2, j = 1, 2, K, n , які ділять K^n на n s = 2 рівних частин - гіпероктантів. Якщо ми розіб'ємо нашу послідовність на відрізки P0, K, Ps-1, Ps ,K , P2s-1, P2s, K, P3s-1, K, то кожен такий відрізок містить одну і тільки одну точку в кожному гіпероктанті. Неважко переконатися, що для випадкових точок 0 1 , , P K Ps- такий розподіл вкрай неймовірний за 3 n ≥ . Імовірність такої події дорівнює $s!s^{-s}$, і вона швидко зменшується зі збільшенням n. Так, при n = 3, коли s = 8, вона дорівнює $2.4\text{-}10^{-4}$.

2.5 Визначення коефіцієнтів важливості

Визначення коефіцієнтів важливості приватних критеріїв за інформацією про аналоги об'єктів управління і проектування[4]. Розглянемо питання формалізації процедур розв'язання векторних задач оптимізації, ґрунтованих на інформації про аналоги (природні або технічні) проектованих

об'єктів. При цьому наведемо класифікацію інформації про прототипи; для розв'язування задач розрахунку вагових коефіцієнтів скалярних цільових функцій і будемо використовувати запропоновану аксіоматику компромісу (2.1) - (2.2).

Крім розрахунку коефіцієнтів важливості деяких відомих узагальнених згорток чесних показників, розглянемо питання розроблення процедури виведення «згорток» приватних критеріїв методами самоорганізації математичних моделей[5].

У роботі для розв'язання МКЗ оптимізації конструкцій виду (2.1) - (2.2), коли відомі $x^a = (x_1{}^a, ..., x_n{}^a)$ - параметри аналога проектованої конструкції, представленого як змістовно, так і за допомогою вектора керованих параметрів $x = (x_1, ..., x_n)^T$, $f_j(x)$ - приватних критеріїв, $g_k(x)$ - функції обмежень, використано модель виду

$$c(x,\alpha) = \sum_{j=1}^{N} \alpha_j f_j(x), \alpha_j \geq 0, \sum_{j=1}^{N} \alpha_j = 1, \qquad (2.1)$$

коефіцієнти якої $\{\alpha_j\}_N$, визначено з рівнянь

$$\{\partial c(x,\alpha)/\partial x_i /_{x=x^a} = 0\}_n , \qquad (2.2)$$

а регулювальний критерій, що забезпечує однозначність оцінок коефіцієнтів, отриманих під час $N \neq n+1$, визначено методом найменших квадратів.

Основні труднощі практичного використання методики зумовлені обмеженнями завдання прототипів x^a. За кількох $\{x^a\}_l$ або за відмінності у складі параметрів проектування $[x] \neq [x^a]$ процедури не визначені.

У деяких випадках обмеження, притаманні завданню, а також у разі розбіжності складу x і x^a , можуть бути усунуті під час використання замість (2.3) функції виду (2.4). Як показано вище, застосування (2.1) для розв'язання задач із моделлю (2.2), якщо мають місце властивості сукупної суперечливості (2.3) - (2.4), гарантує одержання єдиного $x^{opt} \in P_x$, який задовольняє аксіоми:

оптимальність за Парето, симетричність за j. Розглянемо використання інформації про аналоги та прототипи для побудови моделі (3.48) з коефіцієнтами важливості, яке для (2.3) - (2.4) докладно досліджено. Додатково вважаємо, що, по-перше, є кілька об'єктів-прототипів (або ідеалізацій), по-друге, оцінки коефіцієнтів не повинні залежати від розмірності вектора параметрів проектування $x_i, \quad i=\overline{1,n}$. Інформація про прототипи виду

$$\left\{ Y_j^{(r)},\ j=\overline{1,N};\quad f_k^- = \min_{r\in I} Y_k^{(r)} - \varepsilon_k;\quad f_k^+ = \max_{r\in I} Y_k(r) + \varepsilon_k \right\} \tag{2.3}$$

спільно з умовами (2.1), вважаючи, що і прототипи $\overline{Y}^{(r)}$ побудовані з використанням (2.2), дає можливість вибору міри $Q(\bullet)$, що дає змогу аналітично знайти розв'язок оберненої в сенсі задачі обчислення оцінок $\{\alpha_k\}_N$ з моделі

$$\min_{\{\beta_k\}_N} Q = \sum_{k=1}^{N-1} \sum_{p=k+1}^{N} \left(\beta_k \lambda_k^0\left(Y_k^0\right) - \beta_p \lambda_p^0\left(Y_p^0\right)\right)^2;\quad \beta_k = \alpha_k^{-1} \tag{2.4}$$

та системи лінійних рівнянь $\partial Q / \partial \beta_k = 0$ у виді

$$\tilde{\beta}_k = \prod_{j=1}^{N} \lambda_j^0 / \lambda_k^0;\quad \beta_k = \tilde{\beta}_k \Big/ \sum_{j=1}^{N} \tilde{\beta}_j;\quad \alpha_k = \beta_k^{-1};\quad k \in N \tag{2.5}$$

де $\lambda_k^0 = \left(Y_k^0 - f_k^-\right)/\left(f_k^+ - f_k^-\right)$. Наявність декількох прототипів враховується або усередненням $\{\alpha_k^r\}_N$, або безпосередньо такими співвідношеннями для $\tilde{\beta}_k$.

Оцінки коефіцієнтів (2.2) залежать від прийнятих у (2.1) значень f_k^-, f_k^+ Аналіз залежності оптимальних рішень (2.5) f^0 від величин $\{f_k^-\}_N$ приводить до висновку про перевагу призначення менших значень f_k^-; у граничному випадку знаходять значення вагових коефіцієнтів як $\lambda_k^0 = Y_k / f_k^+$ з подальшим

розрахунком оцінок $\{\alpha_k\}_N$ згідно з (2.5). Зауважимо, що тут не передбачається параметричний збіг x та x^a,

а на практиці оцінки (2.5) для (2.2) можуть застосовуватися незалежно від того, чи має властивість (2.1).

Розглянемо питання формалізації процедур розв'язання векторних задач оптимізації, що ґрунтуються на інформації про аналоги (природні або технічні) проектованих об'єктів. При цьому наведемо класифікацію інформації про прототипи; для розв'язування задач розрахунку вагових коефіцієнтів скалярних цільових функцій (названих у зворотними задачами ВО - ОЗВО) і будемо використовувати запропоновану аксіоматику компромісу (2.1) - (2.5). Крім розрахунку коефіцієнтів важливості деяких відомих узагальнених згорток чесних показників (метод "скаляризації"), розглянемо питання розроблення процедури виведення "згорток" приватних критеріїв методами самоорганізації математичних моделей.

У роботі для розв'язання МКЗ оптимізації конструкцій виду (2.1) - (2.5), коли відомі $x^a = (x_1{}^a, ..., x_n{}^a)$ - параметри аналога проектованої конструкції, представленого як змістовно, так і за допомогою вектора керованих параметрів $x = (x_1, ..., x_n)^T$, $f_j(x)$ - приватних критеріїв, $g_k(x)$ - функції обмежень, використано модель виду

$$c(x,\alpha) = \sum_{j=1}^{N} \alpha_j f_j(x), \alpha_j \geq 0, \sum_{j=1}^{N} \alpha_j = 1, \qquad (2.6)$$

коефіцієнти якої $\{\alpha_j\}_N$ визначено з рівнянь

$$\{\partial c(x,\alpha)/\partial x_i /_{x=x^a} = 0\}_n , \qquad (2.7)$$

а регулювальний критерій, що забезпечує однозначність оцінок коефіцієнтів, отриманих при $N \neq n+1$, визначено методом найменших квадратів.

Основні труднощі практичного використання методики [1] зумовлені обмеженнями завдання прототипів x^a. При декількох $\{x^a\}_l$ або в разі

відмінності у складі параметрів проектування $[x] \neq [x^a]$ процедури не визначені.

У деяких випадках обмеження, властиві задачі, а також у разі розбіжності складу x та x^a, можуть бути усунені при використанні замість функції виду (2.2). Як показано вище, застосування (2.1) для розв'язування задач із моделлю (2.1) - (2.5), якщо мають місце властивості сукупної суперечливості (2.6) - (2.7) , гарантує отримання єдиного $x^{opt} \in P_x$, задовольняючого аксіомам: оптимальність по Парето, симетрія по *j*.

Зауважимо, що тут не передбачається параметричний збіг x та x^a, а практиці оцінки (2.6) для (2.2) можуть застосовуватися незалежно від того, чи має властивість (2.1).

Відзначимо основні напрями розвитку методу ОЗВО. По-перше, використання функцій, які відрізняються від (3-1) і не передбачають параметричної відповідності об'єкта й аналога. По-друге, вибір (аксіоматичної або шляхом самоорганізації) форми згортки $c'(x,\alpha)$ поряд зі знаходженням оцінок коефіцієнтів $\{\alpha_j\}_N$. По-третє, послаблення вимог, що характеризують категорію аналога x^a, їх класифікація. Під час класифікації інформації про прототипи (або ідеалізації), використовуваної в ОЗВО, виділимо функціонально-параметричні (ФПА) і структурно-функціональні (СФА) аналоги моделей проектованих об'єктів. Для ФПА характерний збіг як складу приватних критеріїв, так і наборів параметрів $\{x_i^{a(j)}\}, j = \overline{1,l}$, де l - число прототипів. Окремий клас становлять завдання за умови l = 1. У СФА подібні лише вимоги та набори функції обмежень $g(x)$.

Вектори керованих параметрів об'єкта і деякі СФА несуттєво відрізняються один від одного, згідно з (2.2), або допускають інтерполяцію як результат машинного виведення методами самоорганізації форм згортки $C''(x,\alpha)$ або приватних критеріїв $f_j'(x)$. Правомочність цього зумовлена тим,

що $f_j(x) \in F(x)$ в (2.6) - (2.7) не означає використання $f_j^a(x)$ як цільової функції $f_j^a(x) \to \max\limits_x$ у прототипах.

Наведемо просту процедуру виведення методом самоорганізації [3] усіченої згортки $C^{"}(x,\alpha)$ вида (2.1), що містить, можливо, не всі $f_j(x), j \in N$.

Нехай $t = \{2,...,N\}$; утворюємо $M_t = C_n^{t-1}$ класів змінних . Якщо прийняти деякі $\{j_t\} \in M_t$ як навчальну, тоді перевірочну послідовність складуть усі $\{j_\kappa\} \in M_t$, для яких $\exists j_k^{'} \in \{j_t\}$. Використовуючи (2.6) - (2.7), при $t = 1 + [\{j_t\}]$, знайдемо оцінки $\{\alpha_j^{ob}(t)\}$ для навчальних і $\{\alpha_j^{pr}(k)\}$ перевірочних множин індексів і обчислимо критерій незміщеності:

$$\tilde{n}_{sm}(j_k) = (\alpha_j^{ob}(t) - \alpha_j^{pr}(k))^2 / (\alpha_j^{ob})^2. \qquad (2.8)$$

Згідно з МГУА, у згортку (2.2) увійдуть ті $f_j(x), j \in N$, з такими коефіцієнтами α_j^*, для яких при j_k^{ob} критерій $n_1 = \max \tilde{n}_{sm}(j_k)$ виявиться мінімальним.

У складних ЗВО оптимізація конструкцій для побудови моделей локальних властивостей методи, запропоновані в ефективно використовуються, поряд із відомими підходами апріорного завдання апроксимацій напружено-деформованого стану або імітаційного моделювання на базі спрощених розрахункових схем.

Висновки до розділу 2

У розділі визначено завдання щодо математичного моделювання процедур багатокритеріального аналізу, які розробляються в дипломній роботі. А саме – процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей. - спеціалізовану процедуру для забезпечення паралельного зондування

областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування,

- метод визначення оцінок коефіцієнтів важливості багатокритеріальних завдань на основі характеристик прототипів.

Результати математичного моделювання дозволяють створити відповідне програмне забезпечення, призначене для процедур завдано векторної оптимізації (ВО) та дослідженні числових результатів аналізу запропонованих моделей.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка веб додатку

Мета розробки ПЗ полягає у створенні математичних моделей і паралельних алгоритмів, які призначені для багатокритеріального аналізу (БКА) та оптимізації складних систем. Особливість ПЗ полягає у розробці удосконалених моделей і процедур для визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів. Наступне завдання для ПЗ було, удосконалення моделі та створення нової процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей.

Клієнтське програмне забезпечення у своїй структурі містить такі модулі:

– Модуль Entities – модуль який містить інформацію щодо сутностей котрі будуть використовуватися у проекті, а саме сутності Models, Creterials, Prototypes, Values.

– Модуль UseCases – модуль для роботи клієнтського ПЗ, з серверним ПЗ (API), в даному модулі знаходяться усі функції для запитів до API.

Інтерфейс програмного забезпечення складається з таких компонентів:

– ModelTable – таблиця з даними зображена на Рис. 3.2.

– ModelDropDown – випадаючий список, який містить усі моделі котрі зберігаються у БД.

– ModalWindow – універсальне модальне вікно для створення та редагування даних, модальне вікно зображено на Рис. 3.1.

Рис. 3.1 Модальне вікно для створення

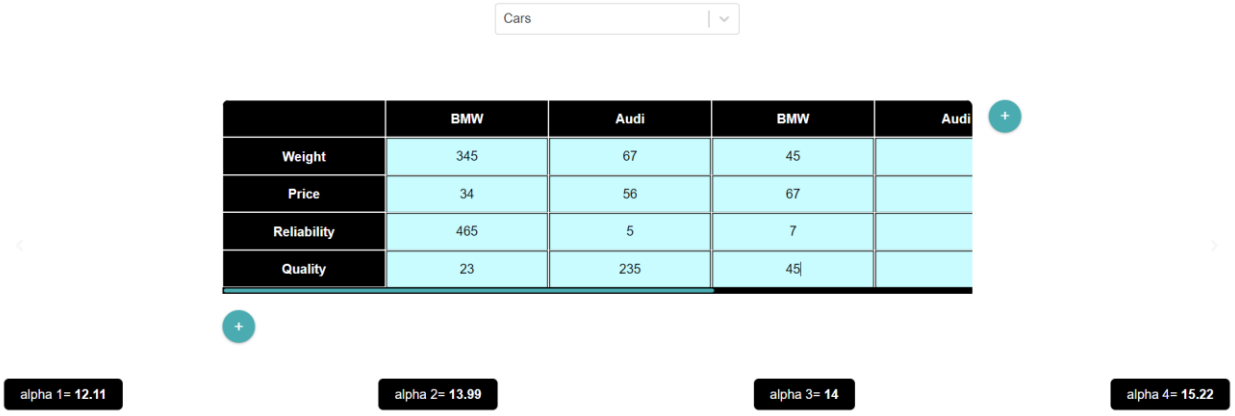

Рис. 3.2 Інтерфейс веб додатку

3.2 Розробка серверного додатку

Серверна частина додатку це API, дане API складається з таких модулів:

Модуль Model – модуль виконує роль таблиці, для котрої ми будемо створювати критерії, та прототипи, для подальшого заповнення. Модуль має такі запити:

– /model/create – POST запит для створення моделі;

– /model/get/:id – GET запит для отримання однієї таблиці моделей, разом з усіма критеріями, прототипами, та їх значеннями;

– /model/get-all – GET запит для отримання листу всіх модулів;

– /model/update – PUT запит для оновлення інформації о модулі, а саме зміна назви таблиці моделей;

– /model/delete/:id – DELETE запит для видалення однієї таблиці модулей.

Модуль Prototype – модуль виконує роль стовпців таблиці, цей модуль працює з прототипами вже існуючих прототипів для моделей. Модуль має такі запити:

– /prototype/create – POST запит для створення прототипу;

– /prototype/get/:id – GET запит для отримання одного стовпчика прототипів, разом з значеннями, для критерій;

– /prototype/get-all – GET запит для отримання листу всіх прототипів;

– /prototype/update – PUT запит для оновлення інформації о прототипі, а саме зміна назви прототипу;

– /prototype/delete/:id – DELETE запит для видалення одного прототипу з моделі.

Модуль Creterial - модуль виконує роль рядків таблиці, цей модуль працює з критеріями оцінки для моделей, та зберігає у собі вже розраховані вагові коефіцієнти. Модуль має такі запити:

– /creterial/create – POST запит для створення критерії;

– /creterial/get/:id – GET запит для отримання одного рядка критеріїв, разом з значеннями, для прототипів;

– /creterial/get-all – GET запит для отримання листу всіх критеріїв;

– /creterial/update – PUT запит для оновлення інформації о критерії, а саме зміна назви критерії;

– /creterial/delete/:id – DELETE запит для видалення однієї критерії з моделі.

Модуль Value - модуль виконує роль значень комірок у таблиці, зберігає в собі усі заповнені данні у таблиці моделі. Модуль має такі запити:

– /value/create – POST запит для зберігання значення;

– /value/get/:id – GET запит для отримання однієї комірки значення у моделі, разом з номером прототипу та номером критерії;

– /value/get-all – GET запит для отримання листу всіх значень;

– /value/update – PUT запит для оновлення інформації о значенні, а саме зміна значення;

– /value/delete/:id – DELETE запит для видалення значення. - Multicriteria-analysis – модуль виконує роль провайдеру, даний модуль зв’язаний з модулем Model та модулем Creterial. В даному модулі виконується розрахунки вагових коефіцієнтів. На Рис. 3.3 зображена блок-схема алгоритму використаного в даному модулі для розрахунку вагових коефіцієнтів для критеріїв при значення прототипів.

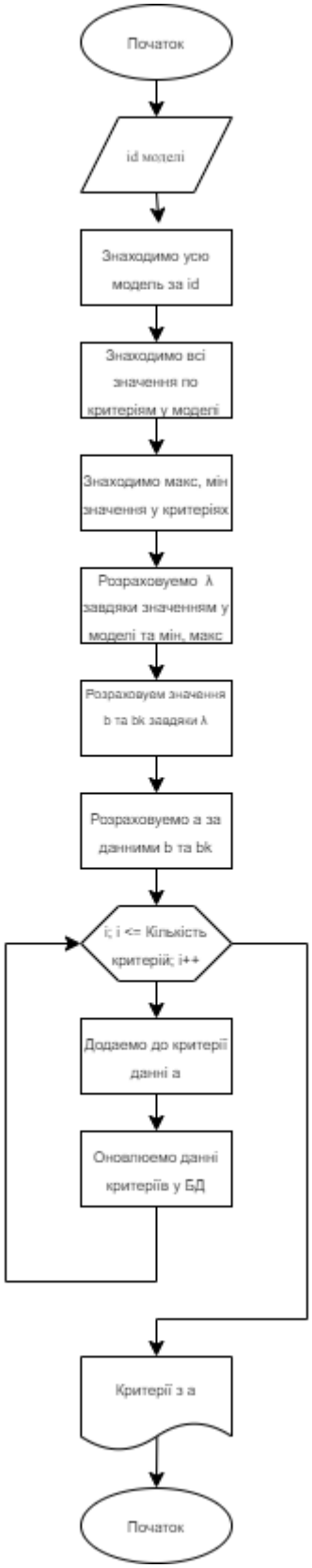

Рис. 3.3 Блок-схема алгоритму розрахунку вагових коефіцієнтів критеріїв

3.3 Розробка бази даних

В даному проекті використовується реляційна база даних (БД) PostgreSQL. На Рис. 3.4 зображено ER-діаграму БД, розроблену для даного проекту.

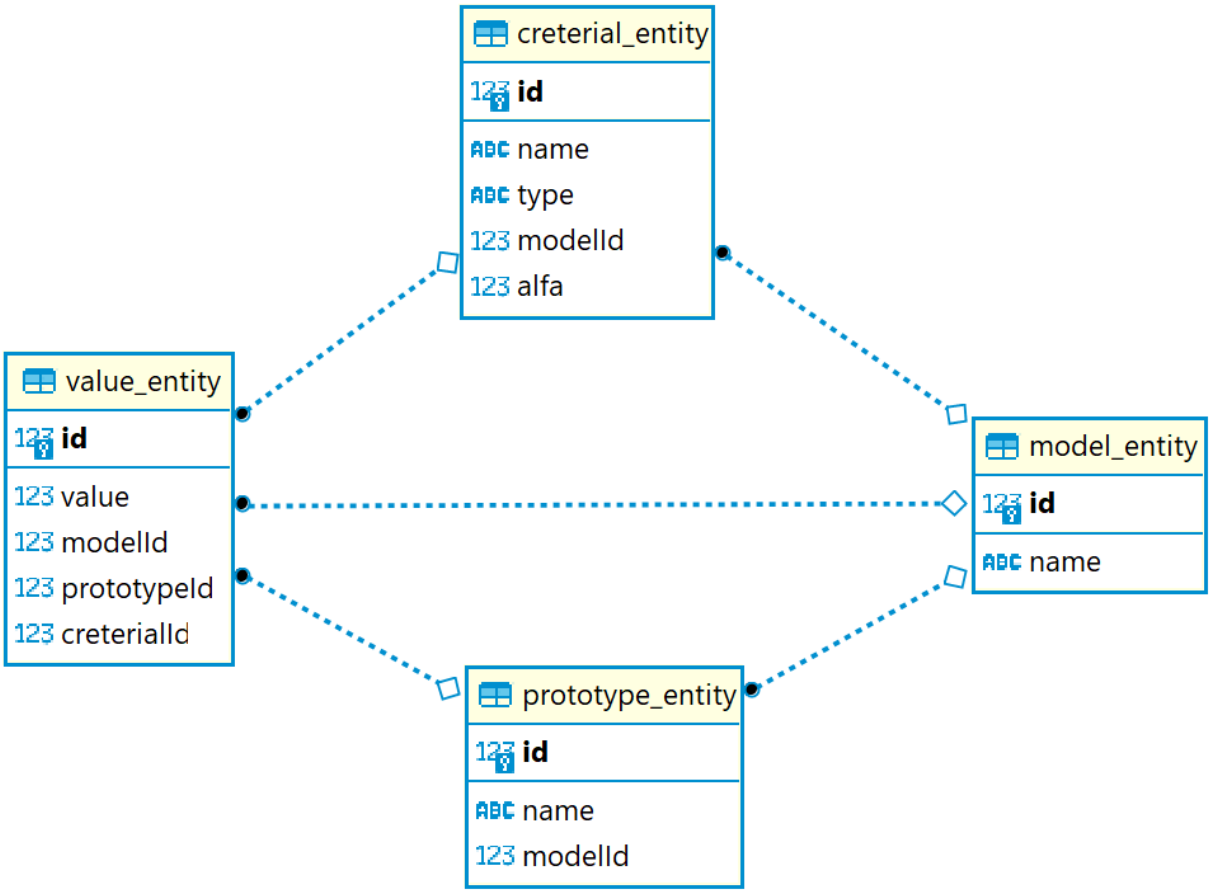

Рис. 3.4 ER-діаграма бази даних

У даному проекті було вибрано реляційну базу даних PostgreSQL[18] через її переваги, а саме:

– PostgreSQL забезпечує високу надійність і цілісність даних. Він підтримує транзакції, які дозволяють виконувати групу операцій як одну атомарну одиницю. Також в базі даних можна використовувати обмеження цілісності, такі як унікальність значень, зовнішні ключі, перевірка умов, що дозволяють контролювати допустимість даних;

– PostgreSQL підтримує багато розширень і додаткових функцій. Він має можливості для географічного розширення (PostGIS), повнотекстового пошуку (pgSearch), роботи з JSON-даними, управління правами доступу, резервне копіювання та багато іншого;

– PostgreSQL підтримує розширену версію мови запитів SQL (Structured Query Language). SQL дозволяє створювати, змінювати та отримувати дані з бази даних, виконувати складні операції з'єднання, фільтрації, сортування;

– PostgreSQL може працювати зі значними обсягами даних і масштабуватися вгору шляхом розподілення навантаження на кластер серверів або вниз за допомогою горизонтального розподілення даних;

– PostgreSQL підтримується на різних платформах, включаючи Linux, Windows, macOS та інші, що дозволяє використовувати його на різних серверах та комп'ютерах.

Висновки до розділу 3

У розділі наведено інформацію щодо розробки удосконаленої моделі та процедури, призначених для визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації складних багато параметричних систем на основі використання даних прототипів. Створено алгоритм удосконаленої моделі та нової процедури, призначеної для забезпечення паралельного зондування областей компромісно-оптимальних рішень (ефективність за Парето) наборами скалярних моделей оптимізації. Була розроблена спеціалізована евристична процедура для забезпечення паралельного зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при встановленій кількості точок зондування. Було розроблено програмне забезпечення щодо реалізації запропонованих процедур багатокритеріальної оптимізації та дослідження числових результатів аналізу моделей.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

4.1. Вибір методів тестування

Для тестування проекту було обрано метод тестування білою скринькою. Тестування білою скринькою є одним з методів тестування програмного забезпечення, при якому тестувальник має повний доступ до внутрішньої структури тестируваної системи[20]. У цьому виді тестування аналізуються внутрішні механізми роботи програми, такі як алгоритми, умовні оператори та змінні. Принцип тестування білою скринькою полягає в тому, щоб проаналізувати внутрішню логіку програми, складність алгоритмів, обробку помилок, і виконати тестування на основі цієї інформації. Тестувальник може використовувати різні методи, такі як тестування рядків коду, тестування гілок умовних операторів, тестування границь та інші, щоб забезпечити високу якість програмного забезпечення.

Тестування білою скринькою має такі плюси, а саме:

– Глибоке тестування: Тестування білою скринькою дозволяє отримати глибоке розуміння внутрішньої структури програми і перевірити її роботу на всіх можливих шляхах виконання.

– Виявлення слабких місць: За допомогою аналізу внутрішньої структури програми, можна виявити потенційні проблеми, які не були б помічені при інших видах тестування.

– Ефективність: Тестування білою скринькою може допомогти зменшити кількість тестових сценаріїв, оскільки тестувальник може спрямовувати зусилля на критичні компоненти програми.

Тестування білою скринькою має такі недоліки, а саме:

– Складність: Тестування білою скринькою вимагає розуміння внутрішньої структури програми, а це може бути часом складним завданням, особливо для великих та складних систем.

– Обмеження на рівень деталей: Велика кількість деталей, що потрібно проаналізувати, може вимагати значних зусиль тестувальника, тому можуть бути пропущені деякі важливі аспекти.

– Залежність від внутрішньої структури: Тестування білою скринькою базується на наявності відкритого доступу до внутрішньої структури програми, тому цей метод може бути непридатним для програмного забезпечення, де внутрішня структура є комерційною та недоступною

4.2 Тестування

Функція для тестування – calculateWeights;

Код функції:

```
async calculateWeights(data) {
    const model = await this.modelService.getOne({ id: data.modelId });
    const creterialValueArray = this.findAllCreterialsValues(model);
    const minMaxCreterialValues = this.getMinMaxCreterialValues(creterialValueArray);
    const lambdas = this.calcLambdas(model.values, minMaxCreterialValues);
    const betta = this.calcBetta(lambdas, model.creterials);
    const alfa = this.calcAlfa(betta);
    const finalCreterial = model.creterials.map((creterial, index) => {
      creterial.alfa = alfa[index];
      return creterial;
    });
    finalCreterial.forEach(async (creterial) => await this.creterialService.update(creterial));
    return finalCreterial;
  }
```

Тести для функції:

Тест 1:

До функції this.modelService.getOne() надходять вхідні данні.

Вхідні дані:

– {id: 1};
– {modelId: 1};
– {}.

Вихідні дані:

– ModelEntity;
– HttpException.

Тест 2:

До функції this.findAllCreterialsValues(model) надходять вхідні данні.

Вхідні дані:

– ModelEntity;
– Null.

Вихідні дані:

– { creterialId: number; values: number [] }[];
– HttpException.

Тест 3:

До функції this.getMinMaxCreterialValues(creterialValueArray) надходять вхідні данні.

Вхідні дані:

– { creterialId: number; values: number [] }[] ;
– Null.

Вихідні дані:

– { creterialId: number; maxValue: number; minValue: number; }[] ;
– HttpException.

Тест 4:

До функції this.calcLambdas(model.values, minMaxCreterialValues) надходять вхідні данні.

Вхідні дані:

– model.values = ValuesEntity / null;

– { creterialId: number; maxValue: number; minValue: number; }[] / null

Вихідні дані:

– { lambda: number; id: number; value: number; model: ModelEntity; prototype: PrototypeEntity; creterial: CreterialEntity }[];

– HttpException.

Тест 5:

До функції this.calcBetta(lambdas, model.creterials) надходять вхідні данні.

Вхідні дані:

– { lambda: number; id: number; value: number; model: ModelEntity; prototype: PrototypeEntity; creterial: CreterialEntity }[] / null;

– CreterialEntity [] / null.

Вихідні дані:

– { betta: number; bettaK: number; id: number; name: string; model: ModelEntity; values: ValueEntity[] }[];

– HttpException.

Тест 6:

До функції this.calcAlfa(betta) надходять вхідні данні.

Вхідні дані:

– { betta: number; bettaK: number; id: number; name: string; model: ModelEntity; values: ValueEntity[] }[];

– Null.

Вихідні дані:

– number [];

– HttpException.

Таблиція 4.1 – Умовні позначення параметрів:

Набір параметрів FirstTestParams (1)	ModelEntity; HttpException.
Набір параметрів SecondTestParams (2)	ModelEntity; HttpException.
Набір параметрів ThirdTestParams (3)	ModelEntity; HttpException.
Набір параметрів FourthTestParams (4)	ModelEntity; HttpException.
Набір параметрів FifthTestParams (5)	ModelEntity; HttpException.
Набір параметрів SixTestParams (6)	ModelEntity; HttpException.
Збір параметрів (7)	(1) || (2) || (3) || (4)

Тестування білою скринькою:

Таблиця 4.2 – Метод покриття операторів

Умови / Тест	(1)		(2)		(3)		(4)		(5)		(6)	
	+	-	+	-	+	-	+	-	+	-	+	-
1	+	-	-	+	-	+	-	+	-	+	-	+
2	-	+	+	-	-	+	-	+	-	+	-	+
3	-	+	-	+	+	-	-	+	-	+	-	+
4	-	+	-	+	-	+	-	+	+	-	-	+
5	-	+	-	+	-	+	-	+	-	+	-	+
6	-	+	-	+	-	+	-	+	-	+	-	+

Таблиця 4.3 – Метод комбінаторного покриття операторів

Умови / Тест	(7)															
	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+
1	+				-				-				-			
2	-				+				-				-			
3	-				-				+				-			
4	-				-				-				+			

Висновки до розділу 4

У даному розділі розгланута основна мета тестування, вона полягає в перевірці якості програмного забезпечення та переконанні у правильному функціонуванні системи

Вибрана нами функція була протестована методам білої скриньки. Під час роботи використані та описані наступні методи:

- методи білої скриньки:
  - метод покриття операторів;
  - метод комбінаторного покриття операторів

Загальні висновки

Дипломна робота присвячена завданням щодо розробки процедур та засобів вирішення завдань багатокритеріального аналізу та оптимізації складних систем шляхом визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів. Для останніх існують дані, оцінки функціональних ознак та значення наборів цільових показників систем. Наявність вектору цільових показників являється типовою ситуацію при проектуванні та оптимізації складних систем. Одним із головних методів і засобом моделювання та відображення завдань векторної оптимізації (ВО) являється формування області область Парето. Для формування таких областей необхідно вирішувати значну кількість одноцільних завдань оптимізації. Завдання ВО має велику обчислювальну складність. Тому в дипломні роботі виконано удосконалення моделей та алгоритмів таким чином, щоб підвищити точність побудови моделі області Парето, а також числову ефективність програмних засобів її формування.

Завданнями роботи були питання щодо удосконалення математичних моделей і паралельних алгоритмів для реалізації багатокритеріального аналізу (БКА) та оптимізації складних систем, а також розробки відповідних програмних засобів.

Відзначається особливість розроблених удосконалених моделей і процедур, які полягають у визначенні оцінок коефіцієнтів важливості показників ВО на основі використання даних прототипів створюваних об’єктів. Також для удосконалення засобів БКА була створена нова процедура формування області Парето, яка забезпечує паралельне зондування областей компромісів наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей. При цьому була розроблена спеціалізована процедура паралельного зондування областей рішень на основі рівномірних

$ЛП_{тау}$ послідовностей, коли відома (задана) кількість точок зондування, і створено програмне забезпечення процедур БКА. Для досягнення мети були використані методи системного аналізу, математичного моделювання, методи багатокритеріальної оптимізації, методи програмної інженерії.

В результаті виконання дипломної роботи були удосконалені моделі та процедури БКА, призначені для визначення оцінок коефіцієнтів важливості показників ВО за рахунок даних прототипів створюваних об'єктів, удосконалена процедура паралельного зондування областей компромісних рішень з використанням рівномірних $ЛП_{тау}$ послідовностей, а також розроблене програмне забезпечення щодо автоматизації процесів багатокритеріальної оптимізації складних систем і процесів.

Наукова та прикладна новизна роботи полягає в такому: удосконалена модель та процедура, призначена для визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації складних багато параметричних систем на основі використання даних прототипів створюваних об'єктів на основі багатьох прототипів і не залежить від кількості параметрів об'єкту проектування; удосконалена модель та створена нова процедура із забезпечення паралельного зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації на основі використання рівномірних $ЛП_{тау}$ послідовностей; розроблена спеціалізована евристична процедура, призначена для паралельного зондування областей компромісно-оптимальних рішень на основі рівномірних $ЛП_{тау}$ послідовностей при заданій кількості точок зондування.

Дипломна робота складається з 4 розділів, 3 таблиць, 5 рисунків, 104 сторінок.

Результатів дипломної роботи створюють нові можливості щодо застосування удосконалених моделей і процедур формування паралельних алгоритмів та програмних засобів БЛА та ВО, що дозволяє підвищити числову ефективність та точність апроксимації областей Парето за рахунок

використання рівномірних $ЛП_{тау}$ послідовностей. У цілому отримані результати сприяють підвищенню ефективності процесів проектування та аналізу складних багатокритеріальних систем на основі даних аналогів або прототипів створюваних об’єктів або технологій.

Список використаних джерел

1. Почтман Ю. М., Скалозуб В. В. Аксіоматичний підхід до багатокритеріальної оптимізації конструкцій // Вістн. АН СРСР. Механіка твердого тіла. 1984. № 6. С. 153 – 156.

2. Герасимов Є. Н., Почтман Ю. М., Скалозуб В. В. Багатокритеріальна оптимізація конструкцій. Київ: Вища школа, 1985. - 132 с.

3. Малков В. П. Оцінка важливості критеріїв у багатокритеріальній оптимізації на підставі аналогів об'єкта проектування // Прикладні проблеми міцності та пластичності. Аналіз та оптимізація конструкцій: Всесоюзн. міжвуз. зб. /Горківський ун-т, 1989. - С. 4 - 11.

4. Скалозуб В. В. Про вибір принципу оптимальності в багатокритеріальних задачах оптимізації конструкцій // Моделювання та оптимізація складних механічних систем. Київ. 1990. С. 50 – 55.

5. Скалозуб В. В. Процедури багатокритеріальної оптимізації в задачах із сукупно суперечливими критеріями // Алгоритмізація вирішення завдань міцності та оптимального проектування конструкцій. Київ. 1991. С. 53-56.

6. Скалозуб В.В. $П_{тау}$ параметризація завдань векторної оптимізації конструкцій // Проблеми обчислювальної механіки та міцності конструкцій. Т. 2. Дніпропетровськ: Навчальна книга, 1998. С. 92 – 98.

7. Скалозуб В.В. Багатоетапні процедури векторної оптимізації, що використовують аксіоматику компромісів// Комп'ютерні методи в задачах прикладної математики та механіки Зб. наук. тр. / АН України. Ін-т кібернетики ім. В.М. Глушкова. Київ, 1997. С. 117 - 123.

8. Скалозуб В.В. Аксіоматика компромісу в обернених задачах багатокритеріальної оптимізації конструкцій та технічних систем // Математичні методи в завданнях розрахунку та проектування складних механічних систем: Зб. наук. пр. / АН України. Ін-т кібернетики ім. В.М. Глушкова. – Київ, 1992. С. 62 – 65.

9. Pochtman Y. M., Skalozub V.V., Nagornyy D.V. Axiomatic approach to the multicriteria structural optimization . Polska Akademia Nauk . Computer Assisted Mechanics and Engineering Sciences, 5; 245 – 251, 1998.

10. Кайл Симпсон. « Ви не знаєте JS », 1992. С. 20 – 65.

11. [Електронний ресурс] Mobile Vs. Desktop Usage - https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics

12. Згуровський М.З. Інтегровані системи оптимального управління та проектування. - К.: Вища школа, 1990. - 351 с.

13. Компанець Л.Ф., Краснопрошина А.А. , Малюков Н.М. Математичне забезпечення наукових досліджень в автоматиці та управлінні - Київ: Вища школа, 1992. - 287 с.

14. Соболь І.М., Статніков Р.Б. Вибір оптимальних параметрів у задачах із багатьма критеріями. Наука, 1981. - 110 с.

15. Das I., Dennis J.E. Normal-Boundary Intersection: An Alternate Method for Generating Pareto Optimal Point in Multicriteria Optimization Problems. Rice University, Houston, Texas, TR96-19, 1996. – 42 p.

16. Технічна документація з NodeJS [Електронний ресурс] https://nodejs.org/docs/

17. Технічна документація з NestJS [Електронний ресурс] https://docs.nestjs.com/

18. Технічна документація з PostgreSQL [Електронний ресурс] https://www.postgresql.org/docs/

19. [Електронний ресурс] The benefits of using web-based applications - https://www.geeks.ltd.uk/about-us/blog/details/eQU5Ip/the-benefits-of-usingweb-based-applications.

20. Джек Фальк, Хунг К Нгуен, Сэм Канер. Testing Computer Software. 1988. - 110 с.

Додатки

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

ПРОГРАМНИЙ ПРОДУКТ «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем»

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130. 01321-01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Владислав СКАЛОЗУБ

Виконавець

Вячеслав ПЕСОЦЬКИЙ

Норм-контролер

Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО

1116130. 01321-01-ЛЗ

ПРОГРАМНИЙ ПРОДУКТ «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем»

Технічне завдання

1116130. 01321-01-ЛЗ

Листів 14

2023

ЗМІСТ

1 ВВЕДЕННЯ

Програмний продукт «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем», що розробляється, має виконувати функції багатокритеріального аналізу (БКА) та оптимізації складних систем.

Особливість удосконалених моделей і процедур полягає у вирішенні завдання щодо визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів.

Користувач, використовуючи цей програмний продукт, отримує змогу працювати з удосконаленим функціоналом моделі та з результатом створення нової процедури формування області Парето, яка забезпечує паралельне зондування областей компромісно-оптимальних рішень наборами скалярних моделей оптимізації з використанням рівномірних $ЛП_{тау}$ послідовностей.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 08.12.21 №77ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи – «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем». Керівник - Скалозуб В.В.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмний продукт має генерувати моделі та процедури із визначення оцінок коефіцієнтів важливості показників векторних моделей оптимізації на основі використання даних прототипів створюваних об’єктів.

Експлуатаційне призначення – результати розробки надає нові можливості застосування удосконалених моделей і процедур формування паралельних алгоритмів та відповідних засобів БЛА, що дозволяє підвищити числову ефективність та точність апроксимації областей Парето за рахунок використання рівномірних $ЛП_{тау}$ послідовностей.

4 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Програмний продукт повинний забезпечувати можливість створювати та отримувати моделі для подальшого створення та заповнення моделей критеріями та прототипами існуючих об’єктів.

До моделі можуть належать:

- моделі транспортних засобів;
- моделі будівель;
- моделі технічних приладів;

Програмний продукт повинний забезпечувати можливість змінювати данні о прототипах, їх кількість, назву.

Програмний продукт повинний забезпечувати можливість працювати з даними о критеріях, отримувати вже існуючі критерії для моделей, отримувати попередні розраховані вагові коефіцієнти для критеріїв.

Програмний продукт має розраховувати вагові коефіцієнти, за значеннями по існуючим критеріям, існуючих прототипів об’єктів.

Вимоги до вхідних даних:

На кожному етапі користувач має можливість створити, свою модель, критерії, прототипи, користувач має змогу заповнити їх даними.

Вимоги до вихідних даних:

Вихідними даними є:

- список усіх моделей;
- список усіх критеріїв у моделі;
- список збережених прототипів для моделі;
- значення комірок по критеріям для первинного прототипу;
- значення розрахованих вагових коефіцієнтів для критеріїв.

Вимоги до надійності

Вимоги до надійності наступні:

- при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;
- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії бази даних на зовнішньому носії.

4.2 Вимоги експлуатації

Програмний продукт повинен використовуватись у приміщеннях які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.13-99 (див. табл. 1).

Таблиця 1. Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.01-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	Оптимальна	Оптимальна
Холодна	легка-1-а	22-24	40-60	0,1
	легка-1-б	21-23	40-60	0,1
Тепла	легка-1-а	23-25	40-60	0,1
	легка-1-б	22-24	40-60	0,2

Працювати з програмою може людина, що має навички роботи з веб додатками та ознайомлена з керівництвом користувача програмного продукту.

4.3 Вимоги до складу та параметрів технічних засобів

Продукт, що розробляється повинен використовуватись на мобільних пристроях, персональних комп’ютерах тощо, що мають наступні характеристики:

- роздільна здатність дисплею – HD (1280x720);
- оперативна пам’ять – 2 ГБ;
- операційна система – ANDROID, IOS, Windows, Linux, MacOS;
- частота процесора – 1.6 ГГц;

4.4 Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем котрі мають можливість користуватися браузером з підтримкою JavaScript.

4.5 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги.

На зворотній стороні упаковки вказується розробник та його юридична адреса.

ПРОГРАМНИЙ ПРОДУКТ «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем» **Мінімальні системні вимоги:** - роздільна здатність дисплею – HD (1280x720); - оперативна пам'ять – 2 ГБ; - операційна система – ANDROID, IOS, Windows, Linux, MacOS; - частота процесора – 1.6 ГГц;	Програмний продукт «Моделі і паралельні алгоритми багатокритеріального аналізу та оптимізації складних систем» **Розробник: Песоцький В.А** **Кафедра "КІТ", УДУНТ** **м. Дніпро, вул. Лазаряна 2** **2023**

4.6 Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на фізичному носії.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача.

Вся документація програмного додатку повинна задовольняти вимоги до програмної документації.

.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиці 1. – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	
Робочий проект	Програмування та відлагодження програми.	
	Тестування програми	
	Розробка, узгодження і затвердження програмної документації.	

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки Скалозуб В.В.

Прийом здійснюється комісією у складі:

- Горячкін В. М. (керівник підрозділу);
- Скалозуб В. В. (керівник розробки).

8 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

Додаток Б

Текст програми

```
import { Body, Controller, Delete, Get, Param, Post, Put } from '@nestjs/common';
import { CreterialService } from './creterial.service';

@Controller('creterial')
export class CreterialController {
  constructor(private readonly creterialService: CreterialService) {}
  @Post('create')
  async create(@Body() dto) {
    return await this.creterialService.tryCreate(dto);
  }

  @Get('get/:id')
  async getOne(@Param() dto) {
    return await this.creterialService.tryGetOne(dto);
  }

  @Get('get-all')
  async getAll(@Param() dto) {
    return await this.creterialService.tryGetAll(dto);
  }

  @Put('update')
  async update(@Body() dto) {
    return await this.creterialService.tryUpdate(dto);
  }

  @Delete('delete/:id')
  async delete(@Param() dto) {
```

```
        return await this.creterialService.tryDelete(dto);
    }
}
import { Column, Entity, ManyToOne, OneToMany, PrimaryGeneratedColumn } from 'typeorm';
import { CreterialTypeEnum } from './enum/creterial.type.enum';
import { ValueEntity } from '../value/value.entity';
import { ModelEntity } from 'src/model/model.entity';

@Entity()
export class CreterialEntity {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ type: 'text' })
    name: string;

    @Column({ type: 'enum', enum: CreterialTypeEnum })
    type: CreterialTypeEnum;

    @Column({ type: 'float', nullable: true })
    alfa: number;

    @OneToMany(() => ValueEntity, (value) => value.creterial, { onDelete: 'CASCADE' })
    values: ValueEntity[];

    @ManyToOne(() => ModelEntity, (model) => model.creterials, { onDelete: 'CASCADE' })
    model: ModelEntity;
}
```

```
import { Module } from '@nestjs/common';
import { CreterialController } from './creterial.controller';
import { CreterialService } from './creterial.service';
import { CreterialRepository } from './creterial.repository';
import { TypeOrmModule } from '@nestjs/typeorm';
import { CreterialEntity } from './creterial.entity';
import { ModelModule } from 'src/model/model.module';

@Module({
    controllers: [CreterialController],
    providers: [CreterialService, CreterialRepository],
    imports: [TypeOrmModule.forFeature([CreterialEntity]), ModelModule],
    exports: [CreterialService],
})
export class CreterialModule { }
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { DeepPartial, Repository } from 'typeorm';
import { CreterialEntity } from './creterial.entity';

@Injectable()
export class CreterialRepository {
    constructor(@InjectRepository(CreterialEntity) private repository:
Repository<CreterialEntity>) {}
    async createOne(data: DeepPartial<CreterialEntity>) {
        const creterial = await this.repository.create(data);
        return await this.repository.save(creterial);
    }

    async delete(creterial: CreterialEntity) {
```

```
        return await this.repository.remove(creterial);
    }

    async findOne(id: number) {
        return await this.repository.createQueryBuilder('creterial').where(`creterial.id = ${id}`).getOne();
    }

    async findMany(data) {
        return await this.repository.createQueryBuilder('creterial').getMany();
    }

    async update(data) {
        return await this.repository
            .createQueryBuilder()
            .update(CreterialEntity)
            .set(data)
            .where({ id: data.id })
            .returning('*')
            .execute();
    }
}
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { CreterialRepository } from './creterial.repository';
import e from 'express';
import { ModelService } from 'src/model/model.service';

@Injectable()
export class CreterialService {
```

```
    constructor(private creterialRepository: CreterialRepository, private modelService: ModelService) {}

  async tryCreate(data) {
    try {
      return await this.create(data);
    } catch (error) {
      console.log('CreterialService.tryCreate => ', error);
                                    throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetOne(data) {
    try {
      return await this.getOne(data);
    } catch (error) {
      console.log('CreterialService.tryGetOne => ', error);
                                    throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetAll(data) {
    try {
      return await this.getAll(data);
    } catch (error) {
      console.log('CreterialService.tryGetAll => ', error);
                                    throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
```

```
    }
  }

  async tryUpdate(data) {
    try {
      return await this.update(data);
    } catch (error) {
      console.log('CreterialService.tryUpdate => ', error);
                                    throw new HttpException(error.message,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryDelete(data) {
    try {
      return await this.delete(data);
    } catch (error) {
      console.log('CreterialService.tryDelete => ', error);
                                    throw new HttpException(error.message,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async create(data) {
    const model = await this.modelService.getOne({ id: data.modelId });
                if (!model) throw new HttpException('model_not_found',
HttpStatus.BAD_REQUEST);
     return await this.creterialRepository.createOne({ name: data.name, type: data.type,
model });
  }
```

```
  async getOne(data) {
    return await this.creterialRepository.findOne(data.id);
  }

  async getAll(data) {
    return await this.creterialRepository.findMany(data);
  }

  async update(data) {
    return await this.creterialRepository.update(data);
  }

  async delete(data) {
    const creterial = await this.creterialRepository.findOne(data.id);
                if (!creterial) throw new HttpException('creterial_not_found',
HttpStatus.BAD_REQUEST);
    return await this.creterialRepository.delete(creterial);
  }
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsString } from 'class-validator';

export class CreateModelDto {
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  name: string;
}
import { ApiProperty } from '@nestjs/swagger';
```

```
import { IsNotEmpty, IsNumberString } from 'class-validator';

export class DeleteModelDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumberString()
    id: number;
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumberString } from 'class-validator';

export class GetModelDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumberString()
    id: number;
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class UpdateModelDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumber()
    id: number;

    @ApiProperty()
    @IsNotEmpty()
    @IsString()
    name: string;
```

```
}
import { Body, Controller, Delete, Get, Param, Post, Put } from '@nestjs/common';
import { ModelService } from './model.service';
import { CreateModelDto } from './dto/create.model.dto';
import { GetModelDto } from './dto/get.model.dto';
import { UpdateModelDto } from './dto/update.model.dto';
import { DeleteModelDto } from './dto/delete.model.dto';
import { ApiTags } from '@nestjs/swagger';

@ApiTags('model')
@Controller('model')
export class ModelController {
   constructor(private readonly modelService: ModelService) {}
   @Post('create')
   async create(@Body() dto: CreateModelDto) {
      return await this.modelService.tryCreate(dto);
   }

   @Get('get/:id')
   async getOne(@Param() dto: GetModelDto) {
      return await this.modelService.tryGetOne(dto);
   }

   @Get('get-all')
   async getAll() {
      return await this.modelService.tryGetAll();
   }

   @Put('update')
   async update(@Body() dto: UpdateModelDto) {
```

```
        return await this.modelService.tryUpdate(dto);
    }

    @Delete('delete/:id')
    async delete(@Param() dto: DeleteModelDto) {
        return await this.modelService.tryDelete(dto);
    }
}
import { Column, Entity, OneToMany, PrimaryGeneratedColumn } from 'typeorm';
import { ValueEntity } from '../value/value.entity';
import { PrototypeEntity } from 'src/prototype/prototype.entity';
import { CreterialEntity } from 'src/creterial/creterial.entity';

@Entity()
export class ModelEntity {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ type: 'text' })
    name: string;

    @OneToMany(() => ValueEntity, (value) => value.model)
    values: ValueEntity[];

    @OneToMany(() => PrototypeEntity, (prototype) => prototype.model, { onDelete: 'CASCADE' })
    prototypes: PrototypeEntity[];

    @OneToMany(() => CreterialEntity, (creterial) => creterial.model, { onDelete: 'CASCADE' })
```

```
    creterials: CreterialEntity[];
}
import { Module } from '@nestjs/common';
import { ModelController } from './model.controller';
import { ModelService } from './model.service';
import { ModelEntity } from './model.entity';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ModelRepository } from './model.repository';

@Module({
    controllers: [ModelController],
    providers: [ModelService, ModelRepository],
    imports: [TypeOrmModule.forFeature([ModelEntity])],
    exports: [ModelService],
})
export class ModelModule {}
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { DeepPartial, Repository } from 'typeorm';
import { ModelEntity } from './model.entity';

@Injectable()
export class ModelRepository {
    constructor(@InjectRepository(ModelEntity)                private                repository:
Repository<ModelEntity>) {}
    async createOne(data: DeepPartial<ModelEntity>) {
        const model = await this.repository.create(data);
        return await this.repository.save(model);
    }
```

```
    async delete(model: ModelEntity) {
        return await this.repository.remove(model);
    }

    async findOne(id: number) {
        return await this.repository
            .createQueryBuilder('model')
            .leftJoinAndSelect('model.prototypes', 'prototype')
            .leftJoinAndSelect('model.creterials', 'creterial')
            .leftJoinAndSelect('model.values', 'value')
            .leftJoinAndSelect('value.prototype', 'valuePrototype')
            .leftJoinAndSelect('value.creterial', 'valueCreterial')
            .where(`model.id = ${id}`)
            .getOne();
    }

    async findMany() {
        return await this.repository.createQueryBuilder('model').getMany();
    }

    async update(data) {
        return await this.repository
            .createQueryBuilder()
            .update(ModelEntity)
            .set(data)
            .where({ id: data.id })
            .returning('*')
            .execute();
    }
}
```

```
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { ModelRepository } from './model.repository';
import { CreateModelDto } from './dto/create.model.dto';
import { GetModelDto } from './dto/get.model.dto';
import { UpdateModelDto } from './dto/update.model.dto';
import { DeleteModelDto } from './dto/delete.model.dto';

@Injectable()
export class ModelService {
  constructor(private modelRepository: ModelRepository) {}

  async tryCreate(data: CreateModelDto) {
    try {
      return await this.create(data);
    } catch (error) {
      console.log('ModelService.tryCreate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetOne(data: GetModelDto) {
    try {
      return await this.getOne(data);
    } catch (error) {
      console.log('ModelService.tryGetOne => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }
```

```
  async tryGetAll() {
    try {
      return await this.getAll();
    } catch (error) {
      console.log('ModelService.tryGetAll => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryUpdate(data: UpdateModelDto) {
    try {
      return await this.update(data);
    } catch (error) {
      console.log('ModelService.tryUpdate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryDelete(data: DeleteModelDto) {
    try {
      return await this.delete(data);
    } catch (error) {
      console.log('ModelService.tryDelete => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }
```

```
  async create(data: CreateModelDto) {
    return await this.modelRepository.createOne(data);
  }

  async getOne(data: GetModelDto) {
    return await this.modelRepository.findOne(data.id);
  }

  async getAll() {
    return await this.modelRepository.findMany();
  }

  async update(data: UpdateModelDto) {
    return await this.modelRepository.update(data);
  }

  async delete(data: DeleteModelDto) {
    const model = await this.getOne(data);
    if (!model) throw new HttpException('model_not_found', HttpStatus.BAD_REQUEST);
    return await this.modelRepository.delete(model);
  }
}
import { Body, Controller, Post } from '@nestjs/common';
import { MulticriteriaAnalysisService } from './multicriteria-analysis.service';

@Controller('multicriteria-analysis')
export class MulticriteriaAnalysisController {
  constructor(private multicriteriaAnalysisService: MulticriteriaAnalysisService) { }
```

```
  @Post('calculate-weights')
  async calculateWeights(@Body() dto) {
    return await this.multicriteriaAnalysisService.calculateWeights(dto);
  }
}
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { CreterialEntity } from '../creterial/creterial.entity';
import { ModelEntity } from '../model/model.entity';
import { PrototypeEntity } from '../prototype/prototype.entity';
import { ValueEntity } from '../value/value.entity';
import { MulticriteriaAnalysisController } from './multicriteria-analysis.controller';
import { MulticriteriaAnalysisService } from './multicriteria-analysis.service';
import { ModelModule } from 'src/model/model.module';
import { CreterialModule } from 'src/creterial/creterial.module';

@Module({
  controllers: [MulticriteriaAnalysisController],
  providers: [MulticriteriaAnalysisService],
  imports: [
    TypeOrmModule.forFeature([ModelEntity, CreterialEntity, PrototypeEntity,
ValueEntity]),
    ModelModule,
    CreterialModule,
  ],
})
export class MulticriteriaAnalysisModule {}
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { CreterialEntity } from 'src/creterial/creterial.entity';
import { CreterialService } from 'src/creterial/creterial.service';
```

```
import { ModelEntity } from 'src/model/model.entity';
import { ModelService } from 'src/model/model.service';
import { PrototypeEntity } from 'src/prototype/prototype.entity';
import { ValueEntity } from 'src/value/value.entity';

@Injectable()
export class MulticriteriaAnalysisService {
  constructor(private modelService: ModelService, private creterialService: CreterialService) {}
  async tryCalculateWeights(data) {
    try {
      return await this.calculateWeights(data);
    } catch (error) {
      console.log('MulticriteriaAnalysisService.tryCalculateWeights => ', error);
      throw new HttpException(error.message, error.status || HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async calculateWeights(data) {
    const model = await this.modelService.getOne({ id: data.modelId });
    const creterialValueArray = this.findAllCreterialsValues(model);
    const minMaxCreterialValues = this.getMinMaxCreterialValues(creterialValueArray);
    const lambdas = this.calcLambdas(model.values, minMaxCreterialValues);
    const betta = this.calcBetta(lambdas, model.creterials);
    const alfa = this.calcAlfa(betta);
    const finalCreterial = model.creterials.map((creterial, index) => {
      creterial.alfa = alfa[index];
      return creterial;
```

```
        });
        finalCreterial.forEach(async (creterial) => await this.creterialService.update(creterial));
        return finalCreterial;
    }

    findAllCreterialsValues(model: ModelEntity) {
        return model.creterials.map((creterial) => {
            const allCreterialValues: number[] = [];
            model.values.forEach((value) => {
                if (value.creterial.id === creterial.id) {
                    allCreterialValues.push(value.value);
                }
            });
            return { creterialId: creterial.id, values: allCreterialValues };
        });
    }

    getMinMaxCreterialValues(creterialValue: { creterialId: number; values: number[] }[]) {
        return creterialValue.map((item) => {
            const minValue = Math.min(...item.values);
            const maxValue = Math.max(...item.values);
            return { creterialId: item.creterialId, maxValue: maxValue + 1, minValue: minValue - 1 };
        });
    }

    calcLambdas(values: ValueEntity[], minMaxCreterialValues: { creterialId: number; minValue: number; maxValue: number }[]) {
```

```
    return values.map((value) => {
        let minValue;
        let maxValue;
        minMaxCreterialValues.forEach((item) => {
            if (item.creterialId === value.creterial.id) {
                minValue = item.minValue;
                maxValue = item.maxValue;
            }
        });
        const lambda = (value.value - minValue) / (maxValue - minValue);
        return { ...value, lambda };
    });
}

calcBetta(
    data: {
        lambda: number;
        id: number;
        value: number;
        model: ModelEntity;
        prototype: PrototypeEntity;
        creterial: CreterialEntity;
    }[],
    prototypes: PrototypeEntity[],
) {
    return prototypes.map((prototype) => {
        const allProtorypeLabmda: number[] = [];
        data.forEach((item) => {
            if (item.prototype.id === prototype.id) {
                allProtorypeLabmda.push(item.lambda);
```

```
            }
        });
        const betta = allProtorypeLabmda.reduce((acc, curr) => acc * curr, 1);
        const bettaSum = allProtorypeLabmda.reduce((acc, curr) => acc + curr, 1) - 1;
        const bettaAvgSum = bettaSum / allProtorypeLabmda.length;
        const bettaK = betta / bettaAvgSum;
        return { ...prototype, betta, bettaK };
      });
  }

  calcAlfa(
    data: {
      betta: number;
      bettaK: number;
      id: number;
      name: string;
      model: ModelEntity;
      values: ValueEntity[];
    }[],
  ) {
    const sumBettaK = data.reduce((acc, curr) => acc + curr.bettaK, 1) - 1 + data[0].bettaK;
    return data.map((item) => item.bettaK / sumBettaK);
  }
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class CreatePrototypeDto {
  @ApiProperty()
  @IsNotEmpty()
```

```
    @IsNumber()
    modelId: number;

    @ApiProperty()
    @IsNotEmpty()
    @IsString()
    name: string;
}

import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumberString } from 'class-validator';

export class DeletePrototypeDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumberString()
    id: number;
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumberString, IsOptional } from 'class-validator';

export class GetListPrototypeDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumberString()
    @IsOptional()
    modelId: number;
}
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';
```

```
export class UpdatePrototypeDto {
    @ApiProperty()
    @IsNotEmpty()
    @IsNumber()
    id: number;

    @ApiProperty()
    @IsNotEmpty()
    @IsString()
    name: string;
}
import { Body, Controller, Delete, Get, Param, Post, Put } from '@nestjs/common';
import { PrototypeService } from './prototype.service';

@Controller('prototype')
export class PrototypeController {
    constructor(private readonly prototypeService: PrototypeService) {}
    @Post('create')
    async create(@Body() dto) {
        return await this.prototypeService.tryCreate(dto);
    }

    @Get('get/:id')
    async getOne(@Param() dto) {
        return await this.prototypeService.tryGetOne(dto);
    }

    @Get('get-all')
    async getAll(@Param() dto) {
```

```
        return await this.prototypeService.tryGetAll(dto);
    }

    @Put('update')
    async update(@Body() dto) {
        return await this.prototypeService.tryUpdate(dto);
    }

    @Delete('delete/:id')
    async delete(@Param() dto) {
        return await this.prototypeService.tryDelete(dto);
    }
}
import { Column, Entity, ManyToOne, OneToMany, PrimaryGeneratedColumn } from 'typeorm';
import { ValueEntity } from '../value/value.entity';
import { ModelEntity } from 'src/model/model.entity';

@Entity()
export class PrototypeEntity {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ type: 'text' })
    name: string;

    @ManyToOne(() => ModelEntity, (model) => model.prototypes, { onDelete: 'CASCADE' })
    model: ModelEntity;
```

```
  @OneToMany(() => ValueEntity, (value) => value.prototype, { onDelete: 'CASCADE' })
  values: ValueEntity[];
}
import { Module } from '@nestjs/common';
import { PrototypeController } from './prototype.controller';
import { PrototypeService } from './prototype.service';
import { PrototypeEntity } from './prototype.entity';
import { TypeOrmModule } from '@nestjs/typeorm';
import { PrototypeRepository } from './prototype.repository';
import { ModelModule } from 'src/model/model.module';

@Module({
  controllers: [PrototypeController],
  providers: [PrototypeService, PrototypeRepository],
  imports: [TypeOrmModule.forFeature([PrototypeEntity]), ModelModule],
  exports: [PrototypeService],
})
export class PrototypeModule { }
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { DeepPartial, Repository } from 'typeorm';
import { PrototypeEntity } from './prototype.entity';

@Injectable()
export class PrototypeRepository {
  constructor(@InjectRepository(PrototypeEntity) private repository: Repository<PrototypeEntity>) {}

  async createOne(data: DeepPartial<PrototypeEntity>) {
```

```
        const prototype = await this.repository.create(data);
        return await this.repository.save(prototype);
    }

    async delete(prototype: PrototypeEntity) {
        return await this.repository.remove(prototype);
    }

    async findOne(id: number) {
        return await this.repository.createQueryBuilder('prototype').where(`prototype.id = ${id}`).getOne();
    }

    async findMany(data) {
        return await this.repository.createQueryBuilder('prototype').getMany();
    }

    async update(data) {
        return await this.repository
            .createQueryBuilder()
            .update(PrototypeEntity)
            .set(data)
            .where({ id: data.id })
            .returning('*')
            .execute();
    }
}
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { PrototypeRepository } from './prototype.repository';
import { ModelService } from 'src/model/model.service';
```

```
@Injectable()
export class PrototypeService {
  constructor(private prototypeRepository: PrototypeRepository, private modelService: ModelService) {}

  async tryCreate(data) {
    try {
      return await this.create(data);
    } catch (error) {
      console.log('PrototypeService.tryCreate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetOne(data) {
    try {
      return await this.getOne(data);
    } catch (error) {
      console.log('PrototypeService.tryGetOne => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetAll(data) {
    try {
      return await this.getAll(data);
    } catch (error) {
```

```
      console.log('PrototypeService.tryGetAll => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryUpdate(data) {
    try {
      return await this.update(data);
    } catch (error) {
      console.log('PrototypeService.tryUpdate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryDelete(data) {
    try {
      return await this.delete(data);
    } catch (error) {
      console.log('PrototypeService.tryDelete => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async create(data) {
    const model = await this.modelService.getOne({ id: data.modelId });
    if (!model) throw new HttpException('model_not_found', HttpStatus.BAD_REQUEST);
```

```
    return await this.prototypeRepository.createOne({ name: data.name, model });
  }

  async getOne(data) {
    return await this.prototypeRepository.findOne(data.id);
  }

  async getAll(data) {
    return await this.prototypeRepository.findMany(data);
  }

  async update(data) {
    return await this.prototypeRepository.update(data);
  }

  async delete(data) {
    const prototype = await this.getOne(data);
    if (!prototype) throw new HttpException('prototype_not_found', HttpStatus.BAD_REQUEST);
    return await this.prototypeRepository.delete(prototype);
  }
}
import { Body, Controller, Delete, Get, Param, Post, Put } from '@nestjs/common';
import { ValueService } from './value.service';

@Controller('value')
export class ValueController {
  constructor(private readonly valueService: ValueService) { }

  @Post('create')
```

```
    async create(@Body() dto) {
        return await this.valueService.tryCreate(dto);
    }

    @Get('get/:id')
    async getOne(@Param() dto) {
        return await this.valueService.tryGetOne(dto);
    }

    @Get('get-all')
    async getAll(@Param() dto) {
        return await this.valueService.tryGetAll(dto);
    }

    @Put('update')
    async update(@Body() dto) {
        return await this.valueService.tryUpdate(dto);
    }

    @Delete('delete/:id')
    async delete(@Param() dto) {
        return await this.valueService.tryDelete(dto);
    }
}
import { Column, Entity, ManyToOne, PrimaryGeneratedColumn } from 'typeorm';
import { CreterialEntity } from '../creterial/creterial.entity';
import { ModelEntity } from '../model/model.entity';
import { PrototypeEntity } from '../prototype/prototype.entity';

@Entity()
```

```
export class ValueEntity {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ type: 'float' })
    value: number;

    @ManyToOne(() => ModelEntity, (model) => model.values)
    model: ModelEntity;

    @ManyToOne(() => PrototypeEntity, (prototype) => prototype.values)
    prototype: PrototypeEntity;

    @ManyToOne(() => CreterialEntity, (creterial) => creterial.values)
    creterial: CreterialEntity;
}
import { Module } from '@nestjs/common';
import { ValueService } from './value.service';
import { ValueEntity } from './value.entity';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ValueController } from './value.controller';
import { ValueRepository } from './value.repository';
import { ModelModule } from 'src/model/model.module';
import { PrototypeModule } from 'src/prototype/prototype.module';
import { CreterialModule } from 'src/creterial/creterial.module';

@Module({
    controllers: [ValueController],
    providers: [ValueService, ValueRepository],
```

```
  imports: [TypeOrmModule.forFeature([ValueEntity]), ModelModule, PrototypeModule, CreterialModule],
})
export class ValueModule { }
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { DeepPartial, Repository } from 'typeorm';
import { ValueEntity } from './value.entity';

@Injectable()
export class ValueRepository {
  constructor(@InjectRepository(ValueEntity) private repository: Repository<ValueEntity>) { }

  async createOne(data: DeepPartial<ValueEntity>) {
    const value = await this.repository.create(data);
    return await this.repository.save(value);
  }

  async delete(value: ValueEntity) {
    return await this.repository.remove(value);
  }

  async findOne(id: number) {
    return await this.repository.createQueryBuilder('value').where(`value.id = ${id}`).getOne();
  }

  async findMany(data) {
    return await this.repository.createQueryBuilder('prototype').getMany();
```

```
    }

    async update(data) {
        return await this.repository
            .createQueryBuilder()
            .update(ValueEntity)
            .set(data)
            .where({ id: data.id })
            .returning('*')
            .execute();
    }
}
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { CreterialService } from 'src/creterial/creterial.service';
import { ModelService } from 'src/model/model.service';
import { PrototypeService } from 'src/prototype/prototype.service';
import { ValueRepository } from './value.repository';

@Injectable()
export class ValueService {
    constructor(
        private valueRepository: ValueRepository,
        private creterialService: CreterialService,
        private prototypeService: PrototypeService,
        private modelService: ModelService,
    ) { }

    async tryCreate(data) {
        try {
            return await this.create(data);
```

```
    } catch (error) {
      console.log('ValueService.tryCreate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetOne(data) {
    try {
      return await this.getOne(data);
    } catch (error) {
      console.log('ValueService.tryGetOne => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryGetAll(data) {
    try {
      return await this.getAll(data);
    } catch (error) {
      console.log('ValueService.tryGetAll => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryUpdate(data) {
    try {
      return await this.update(data);
```

```
    } catch (error) {
      console.log('ValueService.tryUpdate => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async tryDelete(data) {
    try {
      return await this.delete(data);
    } catch (error) {
      console.log('ValueService.tryDelete => ', error);
      throw new HttpException(error.message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  async create(data) {
    const creterial = await this.creterialService.getOne({ id: data.creterialId });
    if (!creterial) throw new HttpException('creterial_not_found', HttpStatus.BAD_REQUEST);
    const prototype = await this.prototypeService.getOne({ id: data.prototypeId });
    if (!prototype) throw new HttpException('prototype_not_found', HttpStatus.BAD_REQUEST);
    const model = await this.modelService.getOne({ id: data.modelId });
    if (!model) throw new HttpException('model_not_found', HttpStatus.BAD_REQUEST);
    return await this.valueRepository.createOne({ value: data.value, creterial, prototype, model });
  }
```

```
  async getOne(data) {
    return await this.valueRepository.findOne(data.id);
  }

  async getAll(data) {
    return await this.valueRepository.findMany(data);
  }

  async update(data) {
    return await this.valueRepository.update(data);
  }

  async delete(data) {
    const value = await this.getOne(data);
    if (!value) throw new HttpException('value_not_found', HttpStatus.BAD_REQUEST);
    return await this.valueRepository.delete(value);
  }
}
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { CreterialEntity } from './creterial/creterial.entity';
import { ModelEntity } from './model/model.entity';
import { PrototypeEntity } from './prototype/prototype.entity';
import { ValueEntity } from './value/value.entity';
import { MulticriteriaAnalysisModule } from './multicriteria-analysis/multicriteria-analysis.module';
import { ModelModule } from './model/model.module';
```

```
import { PrototypeModule } from './prototype/prototype.module';
import { CreterialModule } from './creterial/creterial.module';
import { ValueModule } from './value/value.module';

@Module({
  imports: [
    MulticriteriaAnalysisModule,
    ConfigModule.forRoot({ envFilePath: `.${process.env.NODE_ENV}.env` }),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.PG_HOST,
      port: Number(process.env.PG_PORT),
      username: process.env.PG_USERNAME,
      password: process.env.PG_PASSWORD,
      database: process.env.PG_DATABASE,
      entities: [ModelEntity, CreterialEntity, PrototypeEntity, ValueEntity],
      synchronize: true,
    }),
    ModelModule,
    PrototypeModule,
    CreterialModule,
    ValueModule,
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';
```

```
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

async function bootstrap() {
    const app = await NestFactory.create(AppModule);
    const port = process.env.PORT || 3000;
    app.useGlobalPipes(
        new ValidationPipe({
            transform: true,
            whitelist: true,
            forbidUnknownValues: false,
        }),
    );

    const config = new DocumentBuilder()
        .setTitle('Multicriteria-analysis')
        .setDescription('Multicriteria-analysis docs')
        .setVersion('1.0')
        .addTag('Multicriteria-analysis')
        .build();
    const document = SwaggerModule.createDocument(app, config);

    SwaggerModule.setup('/api/docs', app, document);
    app.enableCors();
    await app.listen(port, () => console.log(`Server started on ${port}`));
}
bootstrap();
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
```

```
  tsconfigRootDir: __dirname,
  sourceType: 'module',
},
plugins: ['@typescript-eslint/eslint-plugin'],
extends: [
  'plugin:@typescript-eslint/recommended',
  'plugin:prettier/recommended',
],
root: true,
env: {
  node: true,
  jest: true,
},
ignorePatterns: ['.eslintrc.js'],
rules: {
  '@typescript-eslint/interface-name-prefix': 0,
  '@typescript-eslint/explicit-function-return-type': 0,
  '@typescript-eslint/no-explicit-any': 0,
  '@typescript-eslint/type-annotation-spacing': [2, {}],
  '@typescript-eslint/semi': ['error', 'always'],
  "@typescript-eslint/no-unsafe-assignment": "warn",
  "@typescript-eslint/no-unsafe-member-access": "warn",
  "@typescript-eslint/no-unsafe-call": "warn",
  "@typescript-eslint/no-unsafe-return": "warn",
  "@typescript-eslint/no-unsafe-optional-chaining": "warn",
  'no-unused-vars': 0,
  '@typescript-eslint/no-non-null-assertion': 0,
  'import/no-unresolved': 0,
  'no-restricted-syntax': 0,
  'import/extensions': 0,
```

```
'no-use-before-define': 0,
'import/prefer-default-export': 0,
'class-methods-use-this': 0,
'no-case-declarations': 0,
'no-useless-constructor': 0,
'no-empty-function': 0,
'lines-between-class-members': 0,
'guard-for-in': 0,
'import/no-extraneous-dependencies': 0,
'one-var-declaration-per-line': 0,
'one-var': 0,
'no-shadow': 0,
'no-underscore-dangle': 0,
'no-unused-expressions': 0,
'consistent-return': 0,
'no-negated-in-lhs': 2,
'no-continue': 0,
'no-await-in-loop': 0,
'block-scoped-var': 2,
'no-plusplus': 0,
'max-len': ['error', { code: 130 }],
'no-param-reassign': 0,
'no-warning-comments': 1,
'vars-on-top': 2,
'no-catch-shadow': 2,
'handle-callback-err': 1,
'consistent-this': [2, 'self'],
'no-bitwise': 2,
camelcase: 0,
'no-mixed-spaces-and-tabs': 2,
```

```
    'generator-star-spacing': [2, { before: true, after: false }],
    'no-console': 1,
    'operator-linebreak': ['error', 'before'],
    'brace-style': [2, '1tbs'],
    'padding-line-between-statements': [
      1,
      // { blankLine: 'always', prev: ['const', 'let', 'var'], next: '*' },
      { blankLine: 'any', prev: ['const', 'let', 'var'], next: ['const', 'let', 'var'] },
      { blankLine: 'always', prev: 'directive', next: '*' },
    ],
  },
};
{
  "singleQuote": true,
  "trailingComma": "all",
  "tabWidth": 4,
  "printWidth": 130,
  "endOfLine": "auto"
}
{
  "name": "models-of-many-creterials",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "prebuild": "rimraf dist",
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
```

```
    "start": "cross-env NODE_ENV=production nest start",
    "start:dev": "cross-env NODE_ENV=development nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^9.0.0",
    "@nestjs/config": "^2.3.2",
    "@nestjs/core": "^9.0.0",
    "@nestjs/platform-express": "^9.0.0",
    "@nestjs/swagger": "^7.0.1",
    "@nestjs/typeorm": "^9.0.1",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.0",
    "cross-env": "^7.0.3",
    "pg": "^8.11.0",
    "postgres": "^3.3.4",
    "reflect-metadata": "^0.1.13",
    "rimraf": "^3.0.2",
    "rxjs": "^7.2.0",
    "typeorm": "^0.3.16"
  },
  "devDependencies": {
```

```
  "@nestjs/cli": "^9.0.0",
  "@nestjs/schematics": "^9.0.0",
  "@nestjs/testing": "^9.0.0",
  "@types/express": "^4.17.13",
  "@types/jest": "28.1.8",
  "@types/node": "^16.0.0",
  "@types/supertest": "^2.0.11",
  "@typescript-eslint/eslint-plugin": "^5.0.0",
  "@typescript-eslint/parser": "^5.0.0",
  "eslint": "^8.0.1",
  "eslint-config-prettier": "^8.3.0",
  "eslint-plugin-prettier": "^4.0.0",
  "jest": "28.1.3",
  "prettier": "^2.3.2",
  "source-map-support": "^0.5.20",
  "supertest": "^6.1.3",
  "ts-jest": "28.0.8",
  "ts-loader": "^9.2.3",
  "ts-node": "^10.0.0",
  "tsconfig-paths": "4.1.0",
  "typescript": "^4.7.4"
},
"jest": {
  "moduleFileExtensions": [
    "js",
    "json",
    "ts"
  ],
  "rootDir": "src",
  "testRegex": ".*\\.spec\\.ts$",
```

```
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
      "**/*.(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  } }
{
  "compilerOptions": {
    "module": "commonjs",
    "declaration": true,
    "removeComments": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "allowSyntheticDefaultImports": true,
    "target": "es2017",
    "sourceMap": true,
    "outDir": "./dist",
    "baseUrl": "./",
    "incremental": true,
    "skipLibCheck": true,
    "strictNullChecks": false,
    "noImplicitAny": false,
    "strictBindCallApply": false,
    "forceConsistentCasingInFileNames": false,
    "noFallthroughCasesInSwitch": false
  } }
```