Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка веб-застосунку для аналізу та візуалізації біологічних даних
(тема)

Виконав:
студент II курсу, групи *СПРм-22-2*

Лахтін В.В.
(прізвище, ініціали)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва спеціальності)

Тип програми *освітньо-наукова*

Освітня програма *Системне проектування*
(повна назва освітньої програми)

Керівник *проф. Саваневич В.Є.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри СТ _____ *проф. Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2024 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

20.06.2024  *Лахтін В.В.*

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 20 червня 2024 р.

Керівник кваліфікаційної роботи  *Саваневич В.Є.*

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*

Рівень вищої освіти другий (магістерський)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова

Освітня програма Системне проектування
(повна назва)

ЗАТВЕРДЖУЮ

Зав. кафедри *СТ*

проф. Гребеннік І.В.

" 1" *квітня* *2024 р.*

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Лахтіну Віталію Валентиновичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Розробка веб-застосунку для аналізу та візуалізації біологічних даних*
затверджена наказом по університету від " 1 **"** *квітня* **2024р. №** *259 Ст*

2. Термін подання студентом роботи до екзаменаційної комісії *20 червня 2024р.*

3. Вихідні дані до роботи (проекту) *Розробити веб-застосунку для аналізу та візуалізації біологічних даних. Система повинна являти собою веб-сайт з інтерфейсом доступу до бази даних. Перелік використовуваних програмних засобів: ОС Microsoft Windows 11, MySQL, Postman, IntelliJ IDEA, Tomcat. Технічне забезпечення: IBM-сумісний ПК з ЦП Intel Core i5 та вище.*

4. Перелік питань, що потрібно опрацювати в роботі
4.1 Вступ 4.2 Аналіз предметної області 4.3 Аналіз існуючих систем візуалізації біологічних даних 4.4 Обґрунтування веб-застосунку як форми програмного забезпечення 4.5 Обґрунтування вибору даних експресії генів 4.6 Системне проектування інформаційної системи візуалізації біологічних даних 4.7 Розробка функціональної моделі IDEF0 для візуалізації біологічних даних 4.8 Розробка діаграми потоків даних веб-застосунку аналізу та візуалізації біологічних даних 4.9 Розробка діаграми дерева вузлів та визначення її мети 4.10 Проектування діаграми варіантів використання 4.11 Визначення нефункціональних вимог 4.12 Визначення функціональних вимог 4.13 Розробка діаграми послідовності дій 4.14 Обґрунтування технології та архітектури клієнтської частини 4.15 Обґрунтування принципів та технологій для серверної частини 4.16 Алгоритм аналізу даних 4.17 Обґрунтування вибору бази даних 4.18 Проектування моделі даних 4.19 Розробка серверної частини веб-застосунку аналізу та

візуалізації біологічних даних 4.20 Розробка інтерфейсу користувача 4.21 Модульне тестування 4.22 Інтеграційне тестування 4.23 Тестування навантаження 4.24 Висновки 4.25 Перелік джерел посилання

5. **Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)** 5.1 Функціональна модель IDEF0, 5.2 Декомпозиція головного процесу «Аналіз та візуалізація біологічних даних» 5.3 Діаграма декомпозиції процесу «Адміністрування системи» 5.4 Діаграма варіантів використання адміністратора 5.5 Діаграма варіантів використання користувача 5.6 Діаграма варіантів використання візуалізації для користувача 5.7 Діаграма послідовності дій 5.8 Розроблена ER-діаграма 5.9 Логічна модель даних для системи аналізу та візуалізації біологічних даних 5.10 Інтерфейс користувача форми реєстрації 5.11 Інтерфейс користувача форми авторизації 5.12 Інтерфейс адміністрування користувачів 5.13 Вибір фільтрів та генів для відображення даних 5.14 Відображення зразків 5.15 Візуалізація за заданими фільтрами 5.16 Вибір файлу для завантаження 5.17 Результат завантаження 5.18 Графік навантаження

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Отримання завдання кваліфікаційної роботи*	*01.04.2024*	*Виконано*
2.	*Аналіз предметної області, існуючі системи, обґрунтування вибору даних для аналізу*	*02-06.04..2024*	*Виконано*
3.	*Визначення вимог до веб-застосунку аналізу та візуалізації даних*	*06-11.04.2024*	*Виконано*
4.	*Вибір мови програмування, бази даних, методів візуалізації, алгоритму аналізу даних*	*11.04-16..04.2024*	*Виконано*
5.	*Написання програмного коду веб-застосунку*	*16.04.2024-18.05.2024*	*Виконано*
6.	*Тестування програмного забезпечення. Тестування навантаження*	*18.05.2024-22.05.2024*	*Виконано*
7.	*Розробка «Посібника користувача»*	*22.05.2024*	*Виконано*
8.	*Оформлення пояснювальної записки та документація програмного коду*	*23.05-1.06.2024*	*Виконано*
9.	*Оформлення графічної частини презентаційних матеріалів, комп'ютерних матеріалів для захисту атестаційної роботи*	*01.06-5.06.2024*	*Виконано*
10.	*Представлення на рецензування*	*19.06.2024*	*Виконано*

Дата видачі завдання *01.04.2024*

Студент _____ Лахтін В.В.

(підпис)

Керівник роботи _____ проф. Саваневич В.Є.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка з кваліфікаційної роботи містить*:* *123* сторінки, *29* рисунка, 3 додатки, 24 джерел

БІОІНФОРМАТИКА, АНАЛІЗ ГЕНОМНИХ ДАНИХ, ВІЗУАЛІЗАЦІЯ ГЕНОМНИХ ДАНИХ JAVA, PYTHON, ХМАРНІ ОБЧИСЛЕННЯ В ГЕНОМІЦІ, ВЕБ ЗАСТОСУНОК

Об'єктом дослідження є веб – застосунок, що призначений для аналізу та візуалізації біологічних даних, зокрема експресії генів.

Предметом дослідження є процеси проектування, розробки та використання веб—застосунку для аналізу та візуалізації біологічних даних з фокусом на інструментах для роботи з даними експресії генів.

Мета кваліфікаційної роботи – створення веб-застосунку, що спрощує аналіз і візуалізацію експресії генів для виявлення біологічних патернів та тенденцій.

Методом дослідження є системний підхід для комплексного аналізу проектування веб-застосунку, методи структурного аналізу і моделювання для визначення архітектури та вимог до системи, об’єктно-орієнтований аналіз для реалізації логіки застосунку, порівняльний аналіз для оцінки існуючих рішень та їх адаптації до потреб проекту, а також експериментальні дослідження для перевірки функціональності та ефективності розробленого застосунку.

Область застосування – біоінформатика, наукові дослідження в медицині та фармацевтиці, геноміка.

ABSTRACT

The explanatory note on the qualification work contains: 123 pages, 29 figures, 3 appendices, 24 source

BIOINFORMATICS, GENOME DATA ANALYSIS, GENOME DATA VISUALIZATION JAVA, PYTHON, CLOUD COMPUTING IN GENOMICS, WEB APPLICATION

The object of the research is a web application designed for the analysis and visualization of biological data, in particular gene expressions.

The subject of research is the processes of designing, developing and using a web application for analysis and visualization of biological data with a focus on tools for working with gene expression.

The goal of the qualification work is to create a web application that simplifies the analysis and visualization of gene expressions to identify biological patterns and trends.

The research method is a systematic approach for comprehensive analysis of web application design, methods of structural analysis and modeling to determine architecture and system requirements, object-oriented analysis to implement application logic, comparative analysis to evaluate existing solutions and their adaptation to project needs, as well as experimental studies to test the functionality and effectiveness of the developed application.

Field of application – bioinformatics, scientific research in medicine and pharmaceuticals, genomics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

IT – інформаційні технології;

IC – інформаційна система;

UML – уніфікована мова моделювання;

GUI (Graphical User Interface) – графічний інтерфейс користувача;

Геном – це набір генетичної інформації в організмі. Метою геному є надання інформації необхідної організму для функціонування;

Java – об'єктно-орієнтована мова програмування з сильною типізацією;

JavaScript – мова програмування, що використовується для створення інтерактивних веб-сторінок;

Python – мова програмування, часто використовується в наукових обчисленнях, аналізі даних;

NCBI – національний центр біотехнологічної інформації, що забезпечує доступ до різноманітних біологічних баз даних;

EMBL-EBI – Європейський інститут біоінформатики, що надає доступ до широкого спектру ресурсів у галузі біологічних даних.

IGV – Integrative Genomics Viewer, це високопродуктивний, простий у використанні інтерактивний інструмент для візуального дослідження геномних даних.

JSON – це текстовий формат обміну даними між комп'ютерами.

ВСТУП

У сучасному світі, де обсяги біологічної інформації зростають експоненціально, особливо актуальним стає питання ефективного аналізу та візуалізації геномних даних. Ця потреба виникає у багатьох сферах, від академічних досліджень до прикладної медицини та фармацевтики. Саме тому розробка інструментів, які б спрощували процес роботи з великими об'ємами генетичної інформації, є вкрай необхідною.

Актуальність даної магістерської роботи полягає у вирішенні проблеми доступу до швидкого та ефективного аналізу біологічних даних через розробку веб-застосунку з інтуїтивно зрозумілим користувацьким інтерфейсом. Вибір розробки веб-застосунку як рішення для аналізу та візуалізації біологічних даних виправдовується його доступністю з будь-якого місця та на будь-якому пристрої з інтернетом, легкістю оновлення та обслуговування, високою масштабованістю та здатністю легко інтегруватися з іншими онлайн сервісами. Такий підхід не лише спрощує колаборативну роботу та обмін даними між дослідниками, але й забезпечує розширені можливості для захисту конфіденційної інформації. Враховуючи ці аспекти, веб-застосунок є оптимальним варіантом для забезпечення ефективного доступу та обробки біологічних даних, поєднуючи у собі зручність, безпеку та гнучкість використання.

Метою даної роботи є розробка веб-застосунку, що дозволяє аналізувати та візуалізувати біологічні дані з різних джерел, з особливим фокусом на експерсії генів. Це включає створення зручного користувацького інтерфейсу, а також реалізацію алгоритмів для обробки даних та їх візуалізації. Область застосування даного застосунку широка та включає біологічні дослідження, генетичні аналіз, може бути корисною для фармацевтичної індустрії та медицини.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Розвиток індустрії

За десятиліття сфера аналізу та візуалізації біологічних даних зазнала значного прогресу, спричиненого бурхливим розвитком високопродуктивних технологій та збільшенням доступності великих масивів біологічних даних. У цьому розділі досліджується еволюція цієї галузі, розглядаються ключові події, технологічні досягнення та їхній вплив на наукові дослідження і клінічне застосування.

Завершення проекту "Геном людини" (HGP) у 2003 році стало поворотним моментом для аналізу біологічних даних. HGP не лише надав повну карту геному людини, але й підкреслив потребу в складних обчислювальних інструментах для обробки та аналізу величезних обсягів даних, що генеруються. Це призвело до розвитку біоінформатики як окремої галузі, що інтегрує біологію, комп'ютерні науки та інформаційні технології.

Поява високопродуктивних технологій, таких як секвенування наступного покоління (NGS), мікроматриці та мас-спектрометрія, зробила революцію в біологічних дослідженнях. Ці технології дозволили генерувати величезні обсяги даних з безпрецедентною швидкістю і точністю. Як наслідок, фокус змістився з генерації даних на їх аналіз, що вимагає вдосконалених алгоритмів, статистичних методів та обчислювальних потужностей.

Зі збільшенням обсягів даних традиційних обчислювальних методів стало недостатньо. Галузь перейшла до більш надійних обчислювальних інфраструктури, включаючи хмарні обчислення та кластери високопродуктивних обчислень. Такі платформи, як Amazon Web Services та Google Cloud Platform, надають масштабовані ресурси для зберігання, обробки та аналізу великих наборів даних, що робить аналіз складних біологічних даних більш доступним та економічно ефективним [24].

Ранні методи візуалізації зосереджувалися на базових діаграмах і графіках для представлення біологічних даних. Такі інструменти, як Excel і прості бібліотеки побудови графіків, надавали початкові рішення для візуалізації даних, але їхня здатність обробляти складні та масштабні дані була обмеженою. Потреба у більш складній візуалізації призвела до розробки спеціалізованих інструментів та бібліотек. Таке програмне забезпечення, як Cytoscape, Gephi та D3.js, дозволяє створювати інтерактивні та динамічні візуалізації, що полегшує дослідження та інтерпретацію складних наборів даних. Ці інструменти підтримують різні типи візуалізації, включаючи мережі, теплові карти та багатовимірні графіки.

Розвиток веб-платформ і хмарні обчислень сприяло спільним дослідженням, дозволяючи вченим з різних дисциплін і місць працювати разом над великомасштабними проектами. Цей спільний підхід призвів до обміну даними, методами та ідеями, що сприяло подальшому розвитку галузі.

Машинне навчання і штучний інтелект суттєво вплинули на аналіз біологічних даних. Алгоритми, здатні навчатися на даних і робити прогнози, були розроблені для різних застосувань, включаючи аналіз експресії генів, прогнозування структури білків і відкриття нових ліків [8].

Інтеграція різних типів омічних даних (геноміки, транскриптоміки, протеоміки тощо) створює як можливості, так і виклики. Розробка методів інтеграції та аналізу мультиомічних даних є передовими наразі.

Поводження з чутливими біологічними даними викликає занепокоєння щодо конфіденційності та безпеки. Забезпечення захисту даних при одночасному проведенні змістовного аналізу є серйозним викликом, який вимагає надійних методів шифрування, безпечного зберігання даних і чіткої нормативно-правової бази. Оскільки обсяги даних продовжують зростати, підтримка масштабованості та продуктивності залишається критично важливою проблемою. Розробка алгоритмів та інфраструктур, здатних ефективно обробляти та аналізувати великі масиви даних, є важливою для подальшого розвитку галузі.

Сфера аналізу та візуалізації біологічних даних досягла значного прогресу завдяки технологічному прогресу та зростаючій доступності великих масивів даних. Ці розробки трансформували дослідницьку та клінічну практику, уможлививши швидші відкриття, персоналізоване лікування та спільні зусилля. Однак залишаються такі проблеми, як конфіденційність даних, масштабованість та інтеграція. Вирішення цих проблем стане ключем до розкриття повного потенціалу аналізу біологічних даних у майбутньому.

1.2 Аналіз існуючих систем візуалізації біологічних даних

Аналіз існуючих додатків та систем, які пропонують аналіз та візуалізацію біологічних даних, виявляє різноманітність підходів та рішень у цій області. Нижче представлено огляд декількох ключових систем, які мають значення для розуміння сучасного стану технологій у цьому напрямку.

Galaxy проект – це веб-платформа з відкритим кодом для біомедичних досліджень, що дозволяє користувачам виконувати, відтворювати та обмінюватися результатами складних аналізів [9].

Сильні сторони:

– зручний інтерфейс. Розроблений для науковців без досвіду програмування, що сприяє широкому доступу;

– відтворюваність. Дозволяє обмінюватися робочими процесами та результатами аналізу, сприяючи прозорості та відтворюваності досліджень.

– широкий набір інструментів. Підтримує широкий спектр геномних аналізів завдяки великій колекції інтегрованих інструментів.

Обмеження:

– не надає широкої можливості робота з дуже великими наборами даних, має ліміти щодо кількості процессів та данних для одночасної обробки;

– відсутність розширених пошукових функцій, таких як фільтрація за різними критеріями, пошук за метаданими або контекстний пошук, значно

звужує спектр використання системи та зменшує її ефективність у роботі зі складними даними;

– вся серверна частина написана за допомогою Python, не використовує переваги мультимовних систем.

Ensembl — це комплексна інформаційна система, що містить геномні дані. Вона підтримує анотації генів, варіаційні дані та порівняльну геноміку [10].

Сильні сторони:

– інтеграція різноманітних даних. Надає уніфіковану платформу для доступу до різноманітної геномної інформації;

– інтерактивні інструменти. Пропонує інструменти для порівняльної геноміки, прогнозування ефектів варіантів тощо;

– регулярні оновлення. Підтримує актуальні геномні дані, що відображають останні результати досліджень.

Обмеження:

– проблеми з навігацією. Величезна кількість даних і функціональних можливостей може ускладнити навігацію і пошук даних для нових користувачів;

– глибина аналізу. Чудово підходить для пошуку даних і базового аналізу, але може не підтримувати специфічний аналіз без зовнішніх інструментів.

IGV (Integrative Genomics Viewer) це декстопний застосунок для візуалізації та дослідження великих, складних наборів геномних даних. Він підтримує широкий спектр типів даних, включаючи вирівнювання послідовностей, анотації та варіанти [11].

Переваги:

– візуалізація. Забезпечує плавну та ефективну візуалізацію великомасштабних геномних даних;

– зручність для користувача. Зосереджується на простоті використання, з простим інтерфейсом, який дозволяє швидку навігацію та дослідження геномних регіонів;

– гнучкість. Підтримує широкий спектр типів і форматів даних, задовольняючи різноманітні дослідницькі потреби.

Обмеження:

– ресурсоємність. Через потребу обробляти великі обсяги даних, IGV вимагає значних обчислювальних ресурсів, включаючи RAM та потужність процесора. Це може стати проблемою для користувачів з менш потужними ПК або старим обладнанням;

– обмежена мобільність. Як десктопний застосунок, знижує мобільність та гнучкість, оскільки користувачі не можуть легко перейти на інший ПК або працювати з даними в дорозі або ділитися легко своїми результатами;

– відстуні модливості для колаборативної роботи.

Попри значний прогрес у розробці інструментів для аналізу та візуалізації біологічних даних, існують певні області, де новий веб-додаток може пропонувати значні переваги. Ключовими аспектами, де розроблюваний застосунок може перевершити існуючі рішення, є:

– масштабованість та гнучкість. Реалізація архітектури, оптимізованої для легкої масштабованості, дозволить системі ефективно обробляти зростаючі обсяги даних, а також адаптуватися до нових вимог дослідження без значних затрат на перепроектування;

– розширений пошук. Впровадження глибоких і гнучких пошукових можливостей, які дозволять користувачам ефективно фільтрувати та сортувати дані за різними параметрами, забезпечить більш точний та цілеспрямований доступ до інформації;

– кращий інтерфейс. Розробка інтуїтивно зрозумілого та візуально привабливого користувацького інтерфейсу може значно покращити

користувацький досвід, спрощуючи навігацію та доступ до інструментів аналізу та візуалізації даних;

– інтеграція з іншими ресурсами даних. Надання можливостей для легкої інтеграції з іншими важливими біологічними базами даних і ресурсами, такими як UCSC Genome Browser та Ensembl дозволяж користувачам залучати розгорнуті датасети та анотації без необхідності залишати екосистему додатку.

1.3 Обґрунтування веб-застосунку як форми програмного забезпечення

Вибір розробки веб-застосунку для аналізу та візуалізації біологічних даних замість інших форм програмного забезпечення обумовлений кількома ключовими перевагами, що роблять веб-технології особливо привабливими для цієї цілі:

– доступність та сумісність. Веб-застосунки легко доступні з будь-якого пристрою, який має інтернет – браузер та підключення до інтернету. Це забезпечує високий рівень сумісності між різними операційними системами та обладнанням, усуваючи потребу у розробці специфічних версій для різних платформ;

– централізоване оновлення та обслуговування. Розробка та підтримка веб-застосунку дозволяє централізовано оновлювати та управляти програмним забезпеченням без необхідності втручання кінцевого користувача. Це означає, що усі користувачі мають доступ до останніх функцій та виправлень помилок як тільки вони стають доступними;

– масштабованість та доступність. Веб-застосунки легше масштабувати відповідно до зростаючої кількості користувачів та збільшення об'єму даних. Крім того, вони забезпечують високий рівень доступності сервісу з будь-якої точки світу, де є інтернет-з'єднання;

– спрощення співпраці та обміну даними. Веб-застосунки сприяють легшій інтеграції з іншими онлайн сервісами та платформами, що дозволяє користувачам легко обмінюватися даними та результатами досліджень. Це також спрощує колаборативну роботу між різними дослідницькими групами;

– безпека даних. Веб-застосунки дозволяють реалізувати розширені заходи безпеки на рівні сервера, забезпечуючи захист конфіденційних біологічних даних та досліджень. Це особливо важливо, коли мова йде про роботу з генетичною інформацією та персональними даними.

Враховуючи ці переваги, розробка веб-застосунку є оптимальним рішенням для забезпечення ефективного доступу до аналізу та візуалізації біологічних даних, забезпечуючи при цьому гнучкість, масштабованість та зручність для користувачів по всьому світу.

1.4 Обґрунтування вибору даних експерсій генів

Вибір зосередитися на даних експерсій генів для аналізу та візуалізації підкріплений кількома факторами, які підкреслюють їхню важливість та зростаючу присутність на ринку:

– досягнення в технології геномного секвенування. Поява високопродуктивних технологій секвенування різко знизила вартість і час, необхідні для секвенування цілих геномів. Це призвело до експоненціального збільшення обсягу геномних даних, у тому числі експерсій генів доступних для аналізу, що зробило їх джерелом для розуміння здоров'я людини, механізмів розвитку хвороб та біорізноманіття [3];

– медицина. Геномні дані лежать в основі персоналізованої медицини, яка адаптує медичне лікування до індивідуальних особливостей пацієнта. Аналізуючи геномні дані, медичні працівники можуть визначити генетичні варіації, які можуть впливати на реакцію людини на певні ліки, тим самим покращуючи результати лікування пацієнтів [7];

– розробка ліків. Розуміючи генетичну основу захворювань, дослідники можуть розробляти ліки, спрямовані на конкретні молекулярні шляхи, що призводить до більш ефективного лікування з меншою кількістю побічних ефектів;

– зростання ринку та інвестиції. За останні роки ринок геноміки значно зріс завдяки технологічному прогресу та розширенню сфер застосування. Згідно з галузевими звітами, очікується, що світовий ринок геноміки продовжить свою траєкторію зростання, підкріплену значними інвестиціями в дослідження і розробки як з боку державного, так і приватного секторів. Це свідчить про високий попит на інструменти аналізу та візуалізації геномних даних.

– вибрані генні експресії мають значну кількість даних, доступних у відкритих репозиторіях, таких як Gene Expression Omnibus, The Cancer Genome Atlas та база даних Ensembl. Велика кількість даних забезпечує всебічний аналіз і надійну статистичну потужність, що дозволяє робити змістовні висновки і візуалізації.

– для безперешкодної інтеграції обрані генні експресії є сумісними з існуючими інструментами та базами даних. Це дозволить користувачам легко імпортувати та експортувати дані, робити перехресні посилання та перевіряти результати, використовуючи інші платформи та ресурси. Забезпечення сумісності також сприяє спільним дослідженням та обміну даними, розширюючи вплив програми.

Дані експресії генів були обрані на основі їх біологічної значущості, доступності даних, технічної здійсненності, дослідницького попиту, інтеграційного потенціалу та потенціалу для нових відкриттів.

2 ВИЗНАЧЕННЯ ВИМОГ

2.1 Системне проектування інформаційної системи візуалізації біологічних даних

Розробка функціональної моделі системи є ключовим етапом у процесі створення веб-застосунку. Цей процес передбачає визначення основних функцій, які повинен виконувати застосунок, та способів взаємодії між ними для забезпечення ефективної роботи системи. Основна мета – створити зрозумілу та логічну структуру, яка відображатиме потоки даних і процеси обробки інформації всередині системи.

Одним з найпоширеніших методів формулювання таких вимог є використання UML. Цей інструмент допомагає описувати можливі сценарії використання системи (use case), які згодом трансформуються в проектні рішення та архітектуру системи.

UML використовує стандарти, поширені в інших галузях, і підтримує кілька типів діаграм. Ці діаграми описують межі, структуру та поведінку як усієї системи, так і окремих її компонентів. Хоча UML не є мовою програмування, діаграми UML можна застосовувати для генерації коду на різних мовах за допомогою спеціальних інструментів. UML тісно пов'язаний з об’єктно-орієнтованим аналізом і проектуванням.

2.2 Розробка функціональної моделі IDEF0 для візуалізації біологічних даних

Створення функціональної моделі IDEF0 (Integration Definition for Function Modeling) є важливим етапом для визначення вимог до інформаційної системи, що дозволить розглянути взаємозв’язки між акторами та їх можливостями в системі. Мета діаграми полягає у створення візуального

представлення складних системи та її процесів, забезпечивши чіткість та розуміння структури і функціонування. Функціональна модель та її деталізація потребує обов'язкового використання входів, виходів, управління та механізмів, що забезпечують виконання функцій. Саме створення цієї моделі допоможе визначити слабкі місця та можливі покращення процесів, що може стати ключовим моментом у оптимізації та підвищенні ефективності роботи програмного забезпечення.

Створення моделі починається з визначення основної функції або мети системи, що стане вершиною ієрархічної структури моделі. Розбиття мети системи на менші компоненти, потребує чіткого розуміння роботи, оскільки буде представлено більш конкретні та чіткі дії у виконання процесу.

Можна сказати, що функціональна модель IDEF0 забезпечить можливістю аналізувати та документувати існуючи процеси, що дозволить полегшити оптимізацію та реалізувати легке провадження змін у системі [17]. Розроблена функціональна модель подана на рисунку 2.1.

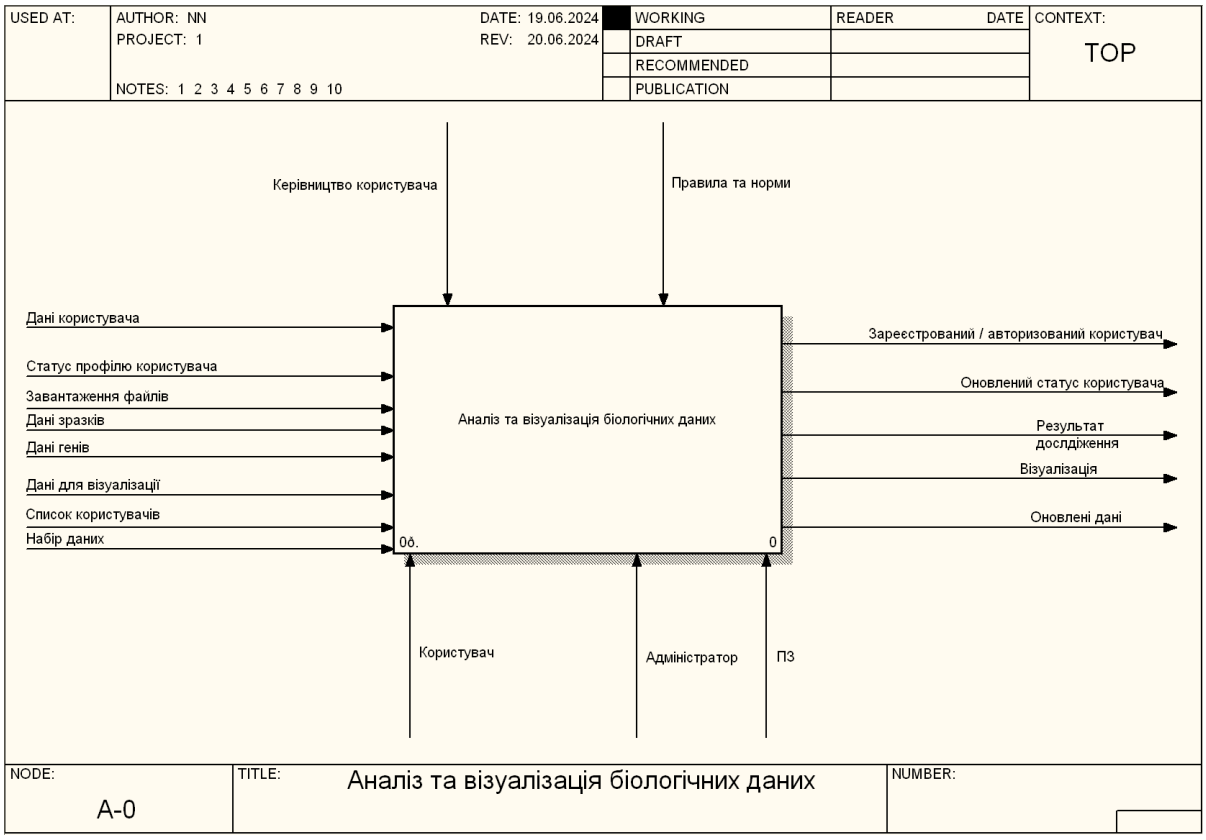

Рисунок 2.1 – Функціональна модель IDEF0

Розроблена функціональна модель містить такі перелік стрілок, що відповідають ра різні дані, що рухаються в системі, трудові затрати, наприклад, робота адміністратора.

Входи, що були визначені для системи аналізу та візуалізації біологічних даних:

– дані користувача;

– статус профілю користувача;

– завантаження файлів;

– дані зразків;

– дані генів;

– дані для візуалізації;

– список користувачів;

– набір даних.

Виходи, що були визначені для системи аналізу та візуалізації біологічних даних:

– зареєстрований / авторизований користувач;

– оновлений статус користувача;

– результат дослідження;

– візуалізація;

– оновлені дані.

Механізми, що були визначені для системи аналізу та візуалізації біологічних даних:

– користувач;

– адміністратор;

– ПЗ (Програмне забезпечення).

Управління, що були визначені для системи аналізу та візуалізації біологічних даних:

– керівництво користувача;

– правила та норми.

Для детального розуміння головної мети було прийняте рішення провести декомпозицію функціональної моделі на менші процеси і функції для кращого розуміння та використання. Даний підхід дозволяє деталізувати процеси від загального рівня до більш чіткого розуміння операцій та дій. Метою цього підходу є досягнення прозорості і простоти функціонування системи, що дозволить визначати додаткові потреби до системи. Використання декомпозиції у функціональному моделюванні забезпечує систематичний підхід до проектування та управління складними системами, підвищуючи їх простоту використання. Діаграма декомпозицій головного процесу подана на рисунку 2.2.

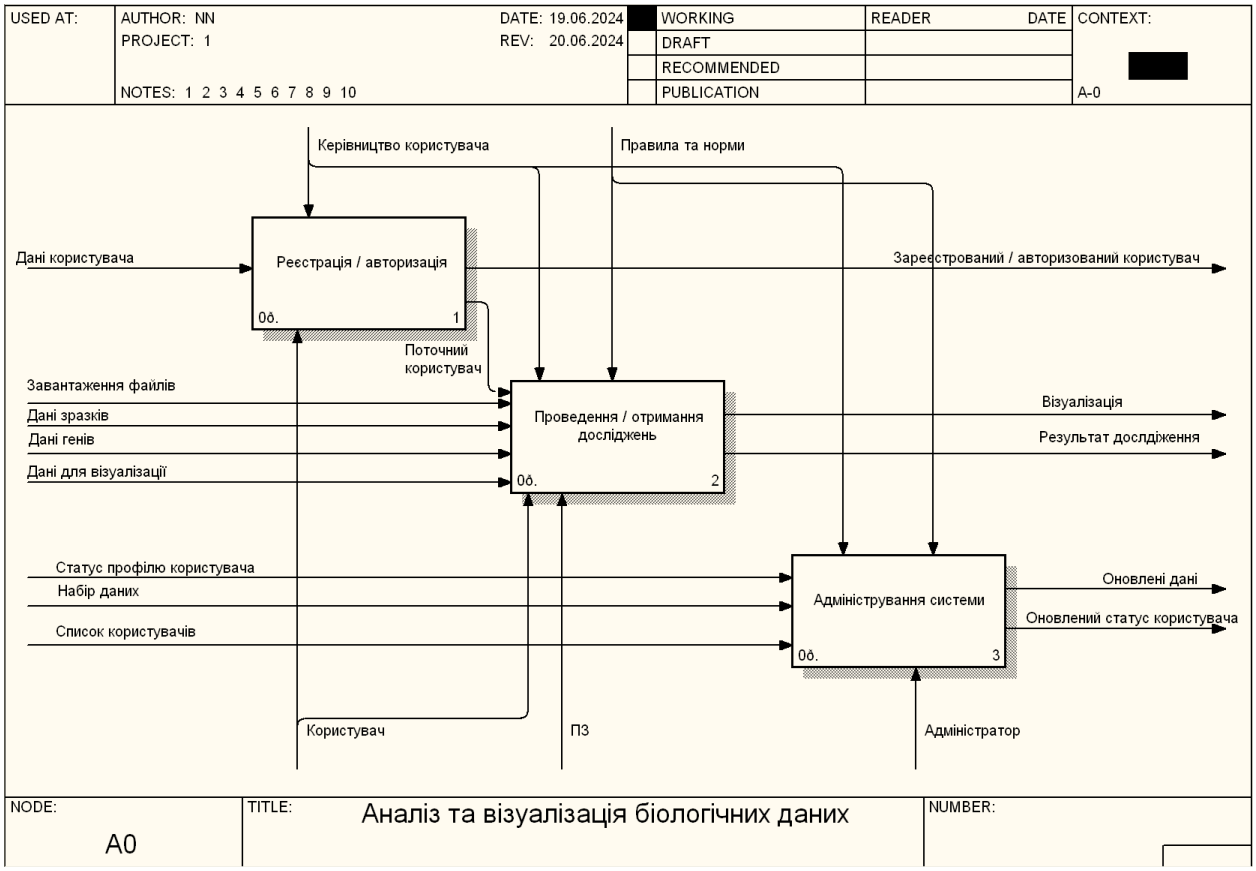

Рисунок 2.2 – Декомпозиція головного процесу «Аналіз та візуалізація біологічних даних»

Проведена декомпозиція дозволила отримати функції системи нижчого рівня, що дозволить більш чітко розуміти мету системи.

Було визначено три головні процеси, що відбуваються в системі та без яких неможливе повноцінне функціонування і використання інформаційної системи.

– реєстрація / авторизація. Реєстрація або авторизація є одною із головних функцій будь-якої інформаційної системи. Даний процес передбачає можливість створити новий профіль користувача або авторизуватись від вже існуючим. Користувачеві необхідно ввести особисту інформацію, но основі якої буде виконано функцію реєстрації та занесення даного профілю до системи. Функція авторизації надає користувачам можливість входити до системи, використовуючи обліковий запис, що дозволить отримати доступ до певного функціоналу та можливостей системи;

– проведення / отримання досліджень. Користувач може надавати інформацію для проведення досліджень з використанням специфічних біологічних даних та потреб користувачів. Для цього процесу можуть використовуватись алгоритми для відображення результатів, що будуть занесені до системи після проведення ряду лабораторних досліджень. За необхідністю користувач може обрати візуалізацію досліджень, для детального ознайомлення;

– адміністрування системи. Передбачає управління даними та користувачами системи. Даний процес повинен для проведення оновлення інформації, надання доступу та інших дій в системі, що недоступні користувачам.

Далі було проведено декомпозицію процесу «адміністрування системи», що дозволить зрозуміти, які функції та дії містяться в даному процесі. На діаграмі декомпозиції представлено можливість управління користувачами та обробкою даних, що присутні в системі. Розроблена декомпозиція подана на рисунку 2.3.

– управління користувачами передбачає можливість оновлювати статус профілю та управляти доступом до системи;

– обробка даних передбачає взаємодію з біологічними дослідженнями користувачів, наприклад, відкрити доступ до інформації всім користувачам.

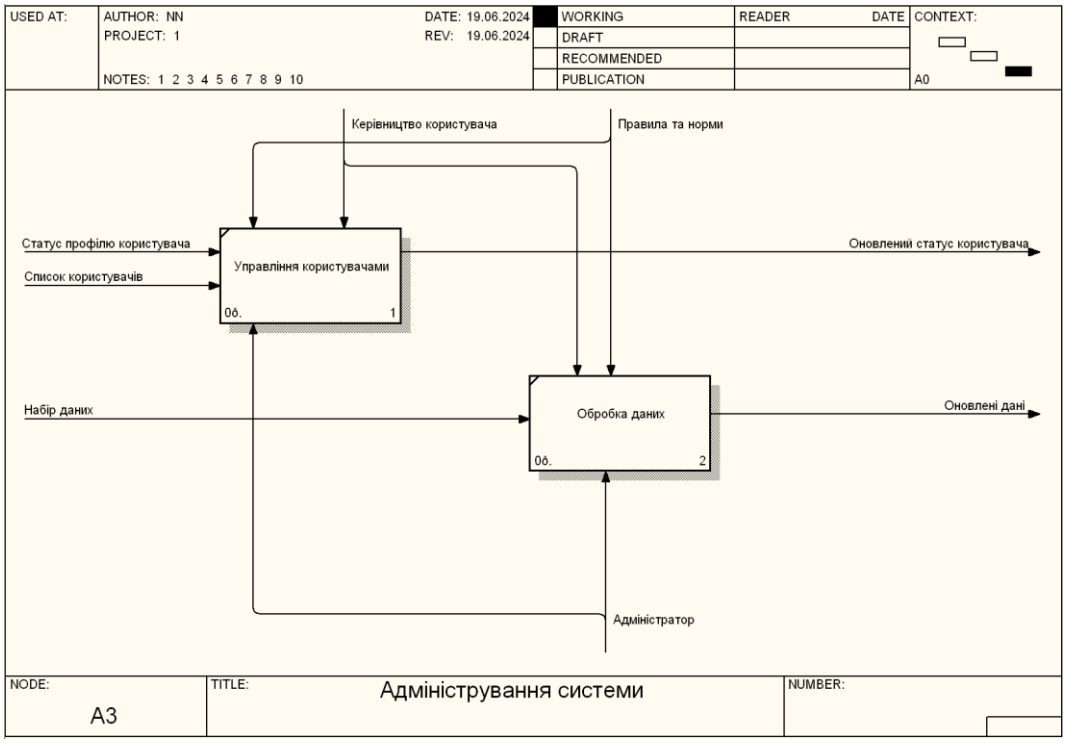

Рисунок 2.3 – Діаграма декомпозиції процесу «Адміністрування системи»

Функція «управління користувачами» доступна тільки адміністраторам системи. Мета функції полягає у оновленні статусу користувача або управлінням доступу. Адміністратор відповідає за управління доступом користувачів і підтримання цілісності бази користувачів у додатку:

– заблокувати користувача. Адміністратор повинен мати можливість заборонити користувачеві доступ до додатку, щоб забезпечити безпеку та контролювати неналежну поведінку.;

– розблокувати користувача. Адміністратор повинен мати можливість відновити доступ користувача до додатку, знімаючи попереднє блокування.

Також адміністратор контролює доступність, видимість даних у системі з можливим їх оновленням:

– зробити дані публічними. Адміністратор повинен мати можливість позначати певні набори даних як публічні, що дозволяє всім користувачам мати до них доступ та сприяє ширшому поширенню та співпраці.;

– встановити дані приватними. Адміністратор повинен мати можливість позначати певні набори даних як приватні, обмежуючи доступ до них лише авторизованим користувачам для захисту конфіденційної інформації від несанкціонованого доступу.

2.3 Розробка діаграми потоків даних веб-застосунку аналізу та візуалізації біологічних даних

Розробивши функціональну модель та провівши декомпозицію бізнес-процесів було прийняте рішення розробити діаграму потоків даних DFD (Data Flow Diagram). DFD – це графічний метод моделювання потоків у системі, що використовується для візуалізації процесів обробки даних [19]. Основною метою створення DFD діаграми полягає в тому, щоб забезпечити чітке та зрозуміле представлення, про те, як рухаються дані між різними компонентами системи, що може стати важливою частиною у вирішення можливих проблем та надати чіткий план для оптимізації процесів. Для більш чіткого розуміння діаграми, було визначено її переваги та недоліки.

Переваги DFD:

– простота. DFD дозволяє легко зрозуміти структуру та потоки даних у системі;

– чіткість уявлення процесів. Завдяки DFD можна побачити, як дані переміщуються між різними частинами системи, що допомагає визначити потенційні проблеми;

– легкість у модифікації. Оскільки DFD – це графічне представлення процесів, його можна легко модифікувати та адаптувати до змін;

– документація. DFD може стати частиною чіткої та детальної документації проекту.

Недоліки DFD:

– обмежена деталізація. DFD може не надати достатньо детальної інформації про складні системи або процеси;

– відсутність стандартів. Існують різні підходи створення DFD, що може призвести до розбіжностей між різними командами;

– велика затрата часу. Створення та підтримка DFD потребує багато часу і зусиль;

– фокус на потоки даних. DFD діаграма зосереджена тільки на потоках даних системи;

– складність для новачків. Незалежно від простоти використання, новачкам може бути складно побудувати DFD діаграму.

Отже, на рисунку 2.4 представлена діаграма потоків даних, що була розроблена на основі створеної функціональної моделі IDEF0 для теми аналіз та візуалізація біологічних даних.

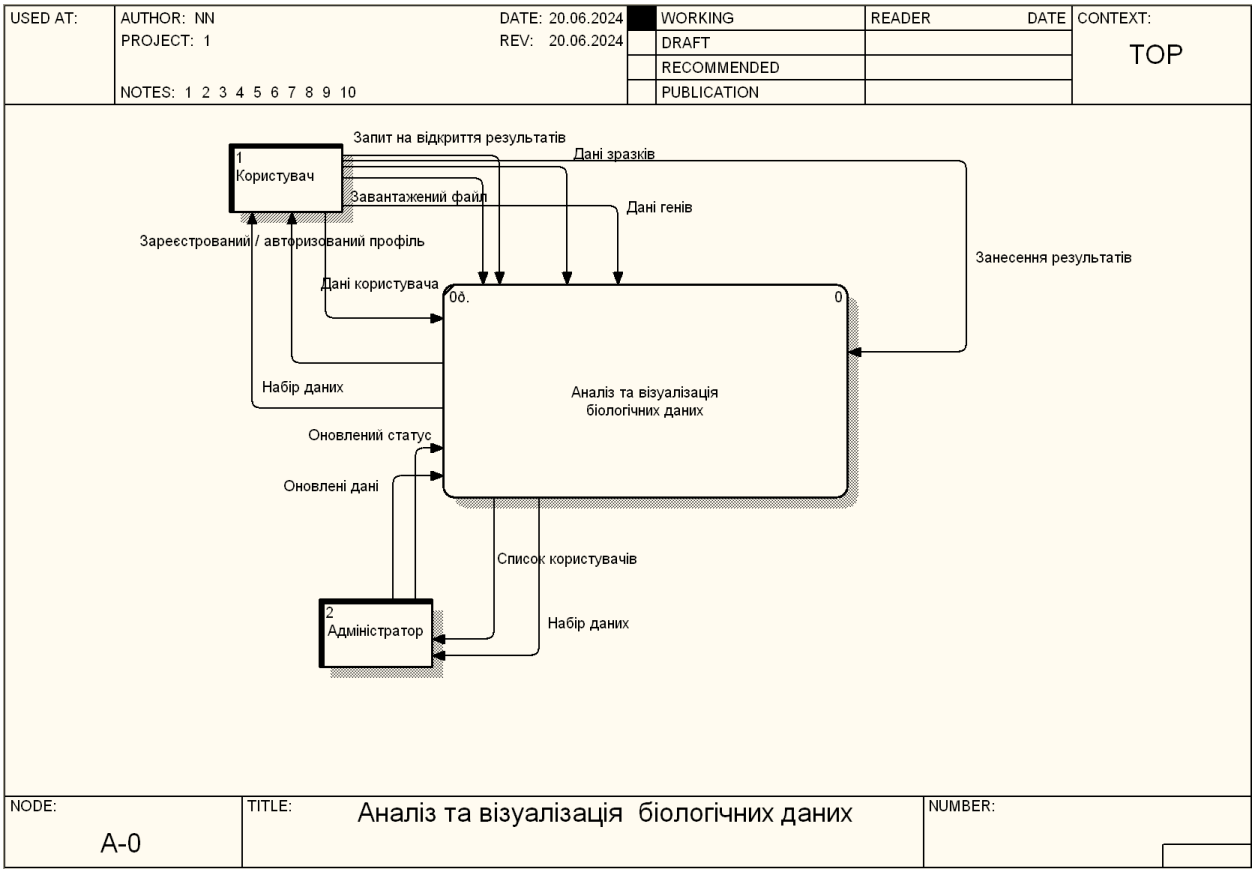

Рисунок 2.4 – Діаграма потоків даних DFD

2.4 Розробка діаграми дерева вузлів та визначення її мети

Підсумовуючи аналіз розробленої функціональної моделі та її декомпозиції, було прийняте рішення створити діаграму дерева вузлів, що відобразить декомпозиції в ієрархічному вигляді. Діаграма дерева вузлів є візуальним інструментом, що складається з вузлів і демонструє їх взаємозв’язки. Головним вузлом представлено початкову точку функціональної моделі від якої відгалужуються інші вузли.

Метою діаграми є забезпечення та відображення чіткої структури при аналізі систем різної складності, дозволяючи користувачам легко зрозуміти взаємозв’язки між елементами [20]. На рисунку 2.5 подано розроблену діаграму дерева вузлів інформаційної системи аналізу та візуалізації біологічних даних.

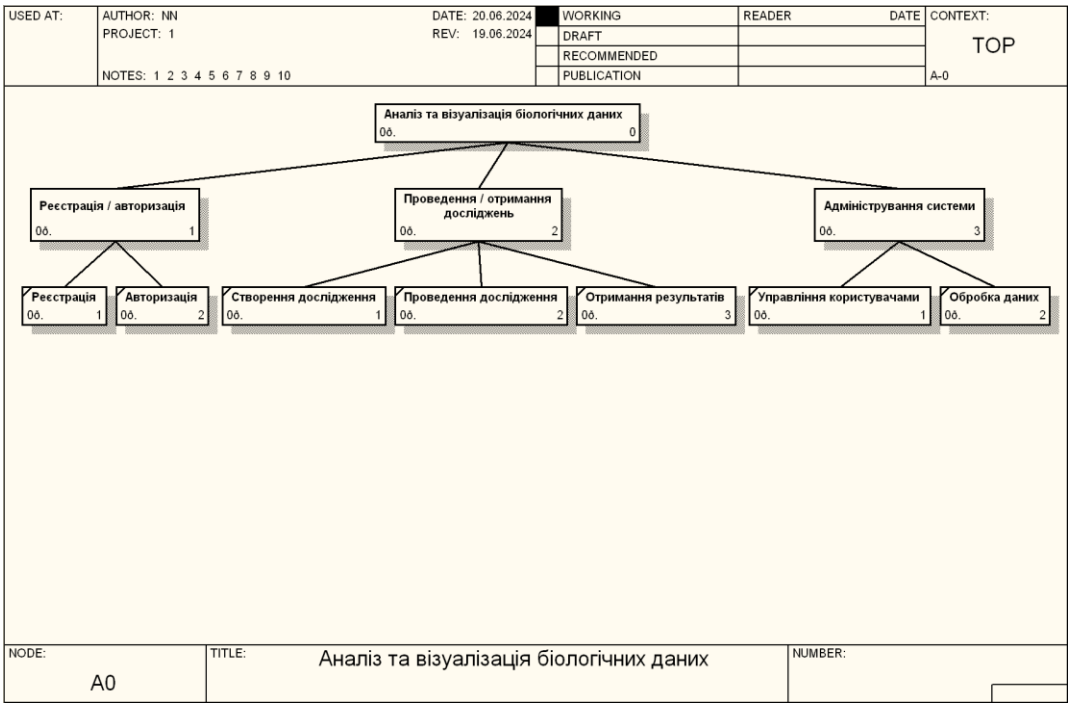

Рисунок 2.5 – Діаграма дерева вузлів інформаційної системи аналізу та візуалізації біологічних даних

На діаграмі дерева вузлів представлено вузли, що відповідають процесам та функціям поданим у функціональній моделі:

– аналіз та візуалізація біологічних процесів;

– реєстрація / авторизація;

– реєстрація;

– авторизація;

– проведення / отримання дослідження;

– створення дослідження;

– проведення дослідження;

– отримання результатів;

– адміністрування системи;

– управління користувачами;

– обробка даних.

Діаграми дерева вузлів дозволяють ефективно візуалізувати ієрархічну структуру системи, що дозволить легко зрозуміти взаємозв’язки між елементами. Саме це робить їх ефективним інструментом для управління складними системи та проектами.

2.5 Визначення функціональних вимог

Діаграма прецедентів (use case diagram) складається з акторів, випадків використання, асоціацій між ними, зв’язків між випадками та узагальнень між акторами [18]. За допомогою таких діаграм описується функціонування веб-застосунку з точки зору користувача, якого в UML називають актором. Актор може бути будь-якою сутністю, зовнішньою по відношенню до системи (особою, програмною системою або пристроєм), яка взаємодіє з системою і використовує її можливості для досягнення своїх цілей або вирішення конкретних завдань.

Веб-додаток повинен мати набір адміністративних функцій для забезпечення ефективного управління користувачами, даними та ролями. Нижче наведено ключові функціональні вимоги до ролі адміністратора.

Адміністратор відповідає за управління доступом користувачів і підтримання цілісності бази користувачів у додатку:

– Заблокувати користувача: адміністратор повинен мати можливість заблокувати користувача, таким чином запобігаючи доступу користувача до додатку. Ця функція необхідна для підтримки безпеки та управління неналежною поведінкою;

– Розблокувати користувача: адміністратор повинен мати можливість розблокувати раніше заблокованого користувача, тим самим відновивши його доступ до додатку.

Адміністратор контролює видимість і доступність даних у додатку:

– Зробити дані публічними: адміністратор повинен мати можливість позначати певні набори даних як загальнодоступні. Публічні набори даних доступні для всіх користувачів, що сприяє ширшому поширенню та співпраці;

– Встановити дані приватними: адміністратор повинен мати можливість позначити певні набори даних як приватні. Доступ до приватних наборів даних обмежений лише авторизованими користувачами, що захищає конфіденційну інформацію від несанкціонованого доступу.

Діаграма варіантів використання для адміністратора подана на рисунку 2.6

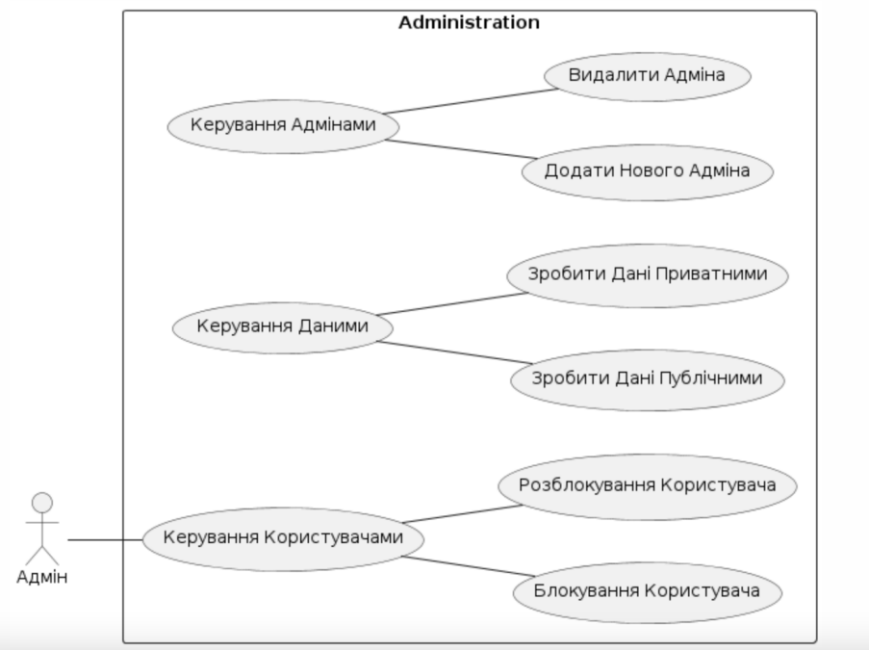

Рисунок 2.6 – Діаграма варіантів використання адміністратора

Наступні функціональні вимоги деталізують можливості, доступні для звичайних користувачів.

Авторизація та регистрація:

– Користувачі повинні мати можливість створити обліковий запис, вказавши ім'я користувача, адресу електронної пошти та пароль;

– Користувачі повинні мати можливість увійти в систему, використовуючи свою зареєстровану електронну пошту та пароль.

Керування даними:

– Завантаження файлів: користувачі повинні мати можливість завантажувати файли даних про експресію генів (наприклад, у форматах CSV, TSV). Система повинна перевіряти формат і структуру файлу;

– Перегляд завантажених файлів: користувачі повинні мати можливість бачити список всіх завантажених ними файлів;

– Видалення завантажених файлів: Користувачі повинні мати можливість видаляти будь—які завантажені ними файли.

Діаграма варіантів використання для користувача подана на рисунку 2.7

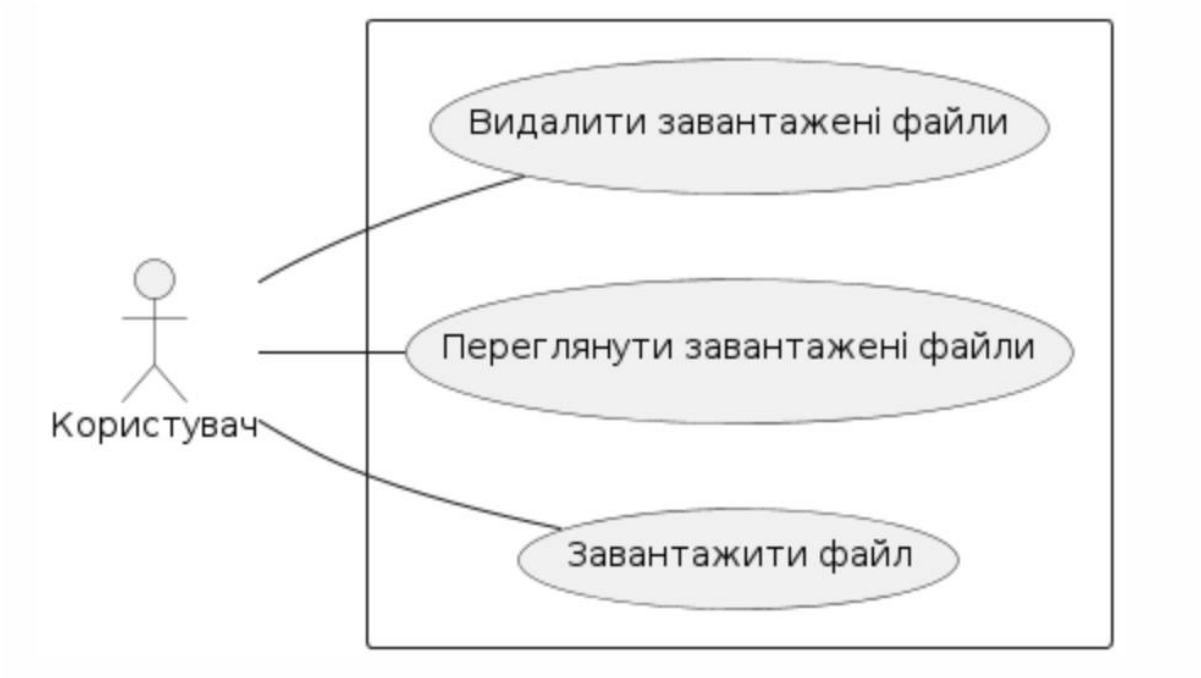

Рисунок 2.7 – Діаграма варіантів використання користувача

Візуалізація даних:

– Перегляд таблиці даних: Користувачі повинні мати можливість переглядати таблицю завантажених ними даних про експресію генів. Таблиця повинна підтримувати сортування та базову фільтрацію;

– Візуалізація теплових карт: Користувачі повинні мати можливість створювати і переглядати теплові карти для своїх даних;

– Кластерні теплові карти: Користувачі повинні мати можливість створювати та переглядати теплові карти з ієрархічною кластеризацією;

– Експорт візуалізацій: Користувачі повинні мати можливість експортувати створені візуалізації (графіки, теплові карти) у вигляді графічних файлів (наприклад, PNG, JPEG).

Функції для пошуку та фільтрації даних:

– Пошук за геном: Користувачі повинні мати можливість шукати певні гени у своїх даних. Пошук повинен відображати відповідні візуалізації та таблиці даних.

– Пошук за атрибутами зразків: Користувачі повинні мати можливість шукати зразки на основі їхніх атрибутів (наприклад, тип зразка). Пошук повинен відображати відповідні візуалізації та таблиці даних.

Діаграма варіантів використання візуалізації для користувача подана на рисунку 2.8

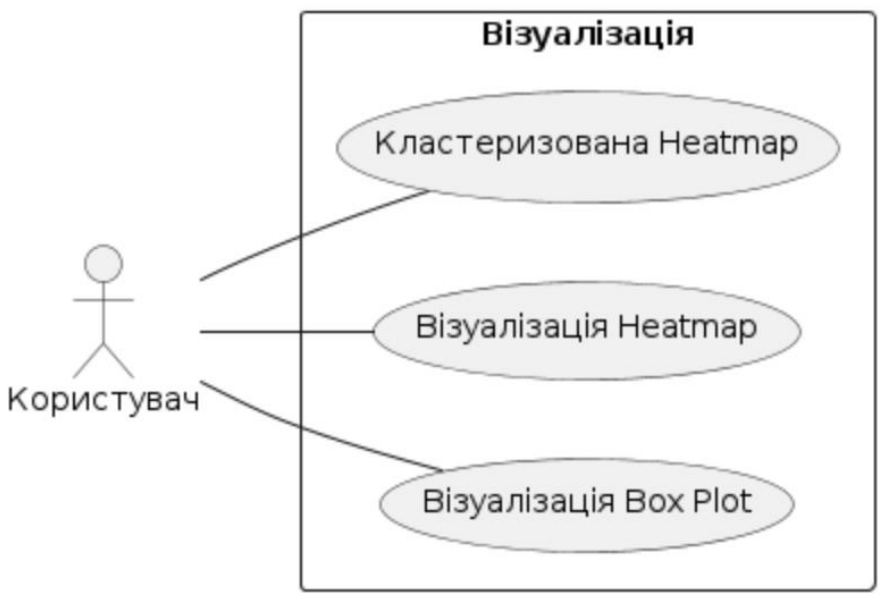

Рисунок 2.8 – Діаграма варіантів використання візуалізації для користувача

2.6 Визначення нефункціональних вимог

Окрім функціональних можливостей, веб-додаток для аналізу та візуалізації біологічних даних повинен відповідати певним нефункціональним вимогам, щоб забезпечити його ефективну, безпечну та надійну роботу. Ці вимоги стосуються загальних атрибутів якості системи, включаючи продуктивність, безпеку, зручність використання, надійність, ремонтопридатність, сумісність, резервне копіювання та відновлення, а також відповідність правовим нормам. Виконання цих нефункціональних вимог забезпечить надійну та зручну роботу програми, гарантуючи довготривалу стійкість та задоволеність користувачів:

– час відгуку. Система повинна реагувати на дії користувача (наприклад, завантаження даних, створення візуалізації) протягом 3 секунд для типових випадків використання;

– масштабованість. Додаток повинен мати можливість обробляти зростаючі обсяги даних та користувачів без значного зниження продуктивності. Він повинен підтримувати набори даних розміром до 1 ГБ;

– конфіденційність даних. Всі дані користувача, включаючи завантажені файли, повинні надійно зберігатися і бути доступними лише для авторизованих користувачів. Конфіденційні дані повинні бути зашифровані як у стані спокою, так і під час передачі;

– інтерфейс користувача. Додаток повинен мати інтуїтивно зрозумілий та зручний інтерфейс, що забезпечує легкість навігації та використання як для початківців, так і для досвідчених користувачів;

– доступність. Система повинна бути доступною 99,9% часу, за винятком планового технічного обслуговування;

– якість коду. Код програми повинен відповідати найкращим практикам щодо читабельності, модульності та документації, щоб полегшити обслуговування та оновлення;

– обробка помилок. Обробку помилок повинна надавати користувачам змістовні повідомлення;

– сумісність з браузерами. Додаток має бути сумісним з усіма основними веб—браузерами (наприклад, Chrome, Firefox, Safari, Edge);

– аудиторські сліди. Ведіть журнали аудиту дій користувачів (наприклад, вхід в систему, завантаження даних, видалення даних) для цілей підзвітності та дотримання нормативних вимог.

2.7 Розробка діаграми послідовності дій

Діаграма послідовності дій (Sequence Diagram) в UML показує, як об'єкти системи взаємодіють між собою через обмін повідомленнями в конкретній послідовності. Вона ілюструє порядок виконання операцій і допомагає виявити логіку роботи системи, показуючи, які об'єкти і в якому

порядку здійснюють взаємодію для виконання певної функції [21]. Основна мета діаграми послідовності – деталізувати і проаналізувати процеси, спрощуючи розуміння того, як система обробляє запити, обробляє дані і виконує функції. Це також допомагає виявити потенційні помилки та оптимізувати взаємодію між компонентами системи [22]. На рисунку 2.9 представлено розроблену діаграму послідовності дій.

Діаграма послідовності для веб-додатку для може проілюструвати процес від моменту входу користувача в систему до отримання ним результатів аналізу та їх візуалізації. Діла подано покрокову послідовність:

Вхід користувача:

– користувач ініціює процес, вводячи свої облікові дані в інтерфейс користувача (UI);

– інтерфейс користувача надсилає ці дані на сервер, де служба автентифікації перевіряє їх;

– після успішної перевірки сервер відповідає інтерфейсу користувача, дозволяючи користувачеві отримати доступ до інформаційної панелі.

Завантаження даних:

– користувач вибирає файли експресій генів даних для завантаження через інтерфейс завантаження даних;

– інтерфейс надсилає запит на сервер для завантаження цих файлів. Служба управління даними отримує цей запит;

– служба управління даними зберігає метадані файлу в базі даних і, можливо, ініціює зберігання вмісту файлу в зовнішньому хмарному сховищі, якщо це необхідно.

– служба сповіщення повідомляє користувача про результат обробки та збереження даних з файлу за допомогою пошти або через UI веб-застосунка.

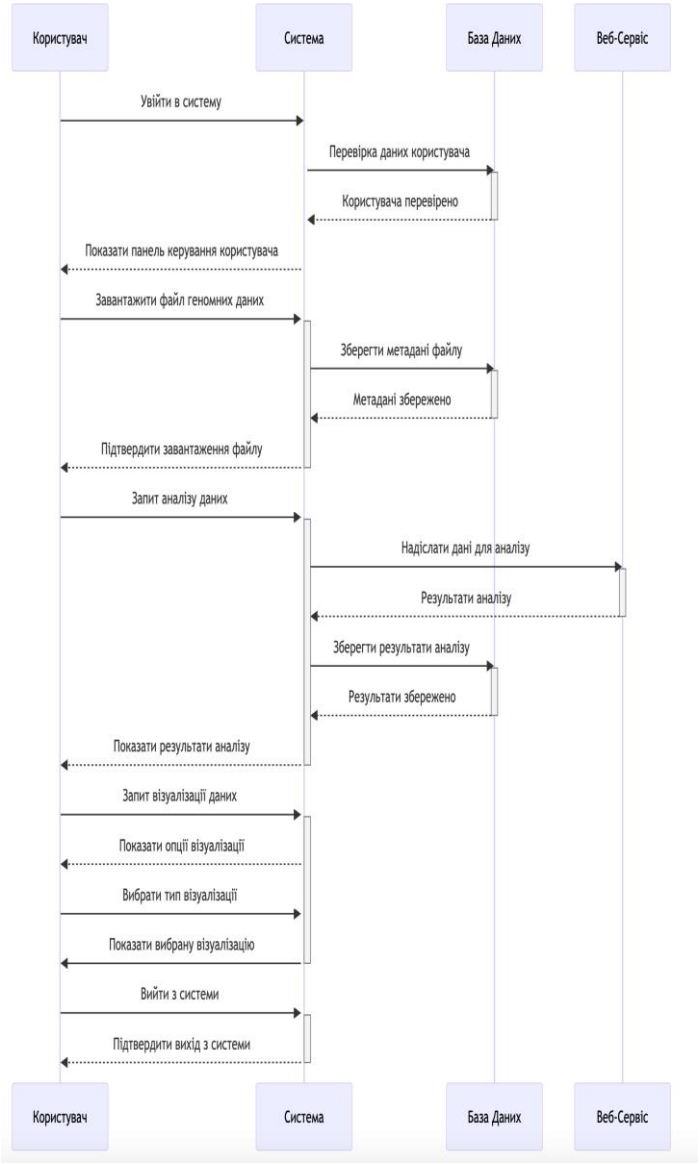

Рисунок 2.9 – Діаграма послідовності дій

3 ОПИС ПРИЙНЯТИХ РІШЕНЬ

3.1 Обґрунтування технології та архітектури клієнтської частини

Вибір сучасних технологій та ефективних архітектурних рішень має вирішальне значення для створення високопродуктивного, надійного та легкого у використанні застосунку. Основна мета архітектури інтерфейсу є забезпечення швидкої та ефективної взаємодії користувачів із системою, оптимізувати процеси візуалізації та аналізу даних, а також гарантувати легкість масштабування та підтримки застосунку.

Технологічний стек:

– Polymer. Сучасний фреймворк для створення односторінкових застосунків (SPA), що надає потужні інструменти для створення інтерактивних UI, автоматизованого тестування та оптимізації продуктивності [15]. Вибір Polymer забезпечує строгу структуру коду і підтримку типізації, що покращує якість та надійність коду;

– ReduxSaga (state management). Бібліотека для управління станом, заснована на Redux паттерні, ідеально підходить для Polymer додатків. Вона забезпечує однонаправлений потік даних і централізоване управління станом, що спрощує розробку складних інтерфейсів і покращує масштабованість проекту;

– WebSockets. Дозволяє встановлювати двостороннє з'єднання між браузером користувача та сервером в реальному часі [23]. Це важливо для застосунків, які потребують швидкого обміну даними, наприклад, для моніторингу аналізу біологічних даних у реальному часі;

– D3.js, Canvas. Бібліотеки для генерації складних і високопродуктивних візуалізацій. D3.js дозволяє маніпулювати документами на основі даних, тоді як Canvas використовується для рендерингу графіків та інших об'ємних

візуалізацій [16]. WebGL може бути розглянуто для ще більш високопродуктивних 3D візуалізацій.

Принципи та паттерни:

– модульний дизайн. Сприяє підвищенню повторного використання коду, спрощенню тестування та підтримки коду;

– однонаправлений потік даних. Забезпечує передбачуваність та легкість управління станом застосунку, покращуючи розробку та відладку;

– реактивне управління станом (Redux). Дозволяє ефективно реагувати на зміни стану застосунку, забезпечуючи високу реактивність інтерфейсу;

– розділення компонентів на розумні (контейнери) та прості (презентаційні). Сприяє чіткому розділенню логіки управління даними та UI, що полегшує розробку, тестування та підтримку коду.

Архітектурно, система розроблена з дотриманням принципів модульного дизайну та розділення компонентів на "розумні" (контейнери) та "прості" (презентаційні), що сприяє чіткому розмежуванню логіки управління даними та їх представлення. Такий підхід забезпечує високу змінність, спрощує тестування та підтримку коду, а також покращує взаємодію між розробниками та дизайнерами.

3.2 Обґрунтування принципів та технологій для серверної частини

Вибір принципів та паттернів для серверної частини, зокрема використання мікросервісної архітектури та модульності, є рішенням, спрямованим на підвищення масштабованості, гнучкості та надійності веб-застосунку для аналізу та візуалізації біологічних даних. Розбиття серверної частини на множину незалежних мікросервісів, кожен з яких відповідає за виконання вузькоспеціалізованої функції (наприклад, управління користувацькими даними, обробка геномних послідовностей, візуалізація даних):

– гнучкість у розробці та розгортанні. Мікросервіси можуть бути розроблені, тестовані та розгорнуті незалежно один від одного, що спрощує внесення змін та оновлень без впливу на роботу всієї системи [6];

– масштабованість. Кожен мікросервіс може бути масштабований незалежно, залежно від потреби, дозволяючи оптимізувати ресурси та обробку запитів;

– надійність. Помилки в одному мікросервісі менше впливають на роботу інших частин системи, що підвищує загальну стабільність застосунку;

Розробка серверного програмного забезпечення з використанням модульного підходу, де кожен модуль має чітко визначену відповідальність і може бути використаний у різних частинах системи або в інших проектах:

– повторне використання коду. Модулі можуть бути легко використані у різних частинах проекту або навіть у інших проектах, що знижує загальні затрати на розробку;

– легкість управління та підтримки. Модулі можна незалежно розвивати, тестувати та вдосконалювати, що полегшує управління кодом та його підтримку;

– гнучкість системи. Модульна структура дозволяє легко додавати, замінювати або видаляти компоненти системи без необхідності переписування великих блоків коду;

Застосування мікросервісної архітектури та модульності дозволяє створити систему, яка легко адаптується до змінних вимог та обсягів даних, забезпечуючи при цьому високу продуктивність, надійність та легкість управління [5]. Ці підходи відіграють ключову роль у досягненні цілей проекту, забезпечуючи стабільну та ефективну роботу веб-застосунку для аналізу та візуалізації біологічних даних.

Вибір Java, Python та AWS (Amazon Web Services) для розробки веб-застосунку підкріплюється наступними перевагами цих технологій:

Java:

– переносність. Java дозволяє "пиши раз, запускай скрізь" завдяки віртуальній машині Java (JVM), що забезпечує високий рівень переносності коду між різними платформами;

– масштабованість та продуктивність. Java ефективно масштабується, що критично для обробки великих об'ємів даних і роботи з розподіленими системами, і є вибором для багатьох великих корпоративних застосунків;

– сильна типізація. Строга типізація допомагає уникнути помилок, пов'язаних з неправильним використанням типів даних, що спрощує відладку та підтримку коду;

– безпека. Java пропонує розширені можливості для безпечного кодування, що є важливим для веб-застосунків, які працюють з чутливими даними.

Python:

– бібліотеки для наукових розрахунків та машинного навчання. Наявність широкого спектру бібліотек, таких як NumPy, Pandas, SciPy, Matplotlib, та бібліотек машинного навчання, наприклад TensorFlow і PyTorch, робить Python ідеальною мовою для аналізу даних та розробки алгоритмів;

– гнучкість. Python можна легко інтегрувати з іншими мовами програмування та технологіями, включаючи Java, дозволяючи використовувати кожну мову там, де це найбільш доцільно.

AWS:

– масштабованість. AWS надає масштабовану інфраструктуру, яка дозволяє збільшувати або зменшувати ресурси згідно з потребами застосунку, забезпечуючи оптимізацію вартості;

– надійність та доступність. Amazon Web Services пропонує високий рівень надійності та доступності, розподіляючи додатки через множину географічно розосереджених центрів обробки даних;

– широкий спектр послуг. AWS пропонує широкий спектр послуг, від обчислень та баз даних до інструментів для машинного навчання та аналізу даних, що дозволяє будувати повноцінні рішення в єдиному середовищі;

– безпека. AWS забезпечує розширені можливості конфігурації безпеки та відповідності стандартам, що дуже важливо для застосунків, які працюють з чутливими даними.

Комбінування Java та Python дозволяє об'єднати силу Java у масштабованих корпоративних рішеннях із гнучкістю Python для швидкої розробки та аналізу даних. AWS як платформа для розгортання дозволяє використовувати ці переваги на повну, забезпечуючи масштабованість, надійність та безпеку застосунку.

3.3 Алгоритм аналізу даних

Кластеризація є найпоширенішим методом аналізу даних експресій генів завдяки кільком ключовим факторам [8]:

– результати кластеризації часто легко інтерпретувати, а візуальні представлення, такі як теплові карти та дендрограми, дають чітке уявлення про структуру даних.

– здатність кластеризації групувати ко-експресуючі гени робить її дуже важливою для біологічної інтерпретації. Гени, що експресуються спільно, часто функціонально пов'язані, і кластеризація може виявити ці функціональні зв'язки.

– кластеризація може бути застосована до різних типів даних про експресію генів, незалежно від того, чи вони отримані з різних тканин, часових точок або умов експерименту. Ця гнучкість робить його універсальним інструментом для широкого спектру досліджень;

– кластеризацію можна легко інтегрувати з іншими подальшими аналізами, такими як аналіз збагачення шляхів і мережевий аналіз, що ще більше підвищує її корисність у біологічних дослідженнях;

– існують усталені методології та інструменти для кластеризації даних про експресію генів, що мають широку літературну базу та підтримку спільноти. Це полегшує дослідникам застосування та перевірку методів кластеризації у своїх дослідженнях.

Отже, кластеризація є потужним інструментом для аналізу даних про експресію генів завдяки своїй здатності виявляти основні закономірності, зменшувати розмірність, генерувати гіпотези, зменшувати шум і ефективно обробляти великі масиви даних. Його широке використання та ефективність у виявленні біологічно значущих знань підкреслюють його важливість у галузі біоінформатики. Вибір методів кластеризації в цій роботі зумовлений необхідністю спростити та покращити інтерпретацію складних біологічних даних, що в кінцевому підсумку сприятиме глибшому розумінню функцій генів та їх взаємодій.

Кластеризація - це метод розподілення об'єктів даних в окремі групи, відомі як кластери, де об'єкти в межах однієї групи дуже схожі між собою, а об'єкти в різних групах дуже відрізняються [13]. Існує два основних підходи до кластерного аналізу даних про експресію генів:

– кластеризація на основі генів. У цьому підході гени розглядаються як об'єкти, а зразки слугують ознаками [3];

– кластеризація на основі зразків. Зразки розглядаються як об'єкти, а гени як ознаки. Зразки поділяються на однорідні групи, кожна з яких потенційно відповідає певному макроскопічному фенотипу, наприклад, клінічним захворюванням або типам раку.

Ієрархічна кластеризація обрана як метод аналізу даних про експресію генів у цій роботі з таких причин:

– широке використання та перевірка в літературі [8]. Ієрархічна кластеризація є добре відомим методом в галузі біоінформатики, і велика кількість літератури підтримує його використання в аналізі експресії генів. Її ефективність і надійність були підтверджені в численних дослідженнях;

– сумісність з іншими методами. Результати ієрархічної кластеризації можна інтегрувати з іншими аналітичними методами, такими як аналіз збагачення шляхів і мережевий аналіз. Ієрархічна структура даних полегшує ідентифікацію ключових генів і шляхів, покращуючи загальну біологічну інтерпретацію результатів;

– візуальне представлення взаємозв'язків даних. Ієрархічна кластеризація створює ієрархію кластерів, яку можна візуально представити у вигляді дендрограми. Таке візуальне представлення особливо корисне для вивчення даних і розуміння взаємозв'язків між різними генами. Дендрограма надає інтуїтивний спосіб побачити, як гени кластеризуються разом, виявляючи закономірності, які можуть бути не одразу помітними за допомогою інших методів;

– не потрібно заздалегідь визначати кількість кластерів. На відміну від методів розбиття, таких як K-середнє, ієрархічна кластеризація не вимагає заздалегідь визначати кількість кластерів. Така гнучкість корисна, коли оптимальна кількість кластерів невідома або коли структура даних складна. Вона дозволяє даним визначати кластери природним чином, що призводить до більш змістовних біологічних інтерпретацій;

– стійкість до шуму та викидів. Ієрархічна кластеризація є відносно стійкою до шуму та викидів. Оскільки вона враховує загальну структуру даних, а не фокусується на окремих точках, вона може краще впоратися з мінливістю, притаманною даним про експресію генів. Ця стійкість має вирішальне значення для біологічних наборів даних, які часто містять експериментальний шум і викиди;

У підсумку, ієрархічну кластеризацію було обрано за її здатність забезпечувати детальний і гнучкий аналіз даних про експресію генів. Її візуальне представлення, стійкість до шуму, універсальність і сумісність з іншими аналітичними методами роблять її ідеальним вибором для виявлення значущих генетичних особливостей у спектрі експресії генів.

Для перевірки точності кластеризації був обраний коефіцієнт кофенетичної кореляції (ККК), визнаний як ефективний метод перевірки результатів кластеризації в біоінформатиці та геноміці [14]. Таке визнання випливає зумовлене наступними факторами:

ККК широко використовується і валідується в біоінформатичній літературі, що робить його надійною і достовірною метрикою для валідації кластеризації [14].

Дані про експресію генів, як правило, включають дані високої розмірності з тисячами генів. ККК добре підходить для перевірки кластеризації в таких високорозмірних просторах, оскільки він забезпечує надійну міру того, наскільки добре кластеризація відображає притаманну структурі даних. Був підрахований коефіцієнт кофенетичної кореляції для кластерного аналізу набору даних. Набір даних мав 410 зразків (стовпчиків) та 19050 генів. Програмний код сервісу поданий на рисунку 3.1. Для різних методів зв'язування коефіцієнт кофенетичної кореляції показав наступні значення:

– метод одиночного зв'язування (Single): 0.1228;

– метод повного зв'язування (Complete): 0.6013;

– метод середнього зв'язування (Average): 0.7783;

– метод Уорда (Ward): 0.6742.

Отримані результати вказують на те, що метод середнього зв'язування (Average) має найвищий коефіцієнт кофенетичної кореляції, що свідчить про найкращу якість кластеризації серед розглянутих методів для даного набору даних. Значення коефіцієнта кофенетичної кореляції вище 0.7 вважається гарним показником якості кластеризації [8]. Алгоритм подано на рисунку 3.1.

```
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import linkage, cophenet
from scipy.spatial.distance import pdist
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
file_path = 'rna.txt'
data = pd.read_csv(file_path, sep='\t')
# Set the 'Gene' column as the index
data.set_index('Gene', inplace=True)
# Display the filtered data
# print(data)

# Compute the distance matrix
distance_matrix = pdist(data.values)
# Define linkage methods
linkage_methods = ['single', 'complete', 'average', 'ward']
cophenet_corr = {}
# Perform hierarchical clustering and compute cophenetic correlation
coefficient
for method in linkage_methods:
Z = linkage(distance_matrix, method=method)
c, coph_dists = cophenet(Z, distance_matrix)
cophenet_corr[method] = c
# Display the cophenetic correlation coefficients
print("Cophenetic Correlation Coefficient for different linkage
methods:")
for method, c in cophenet_corr.items():
print(f"{method.capitalize()}: {c}")
# Visualize the clustered heatmap for the best linkage method
best_method = max(cophenet_corr, key=cophenet_corr.get)
sns.clustermap(data, cmap="YlGnBu", method=best_method,
metric='euclidean', annot=True, figsize=(10, 10))
plt.title(f'Clustered Heatmap with {best_method.capitalize()}
Linkage')
plt.show()
```

Рисунок 3.1 – Програмний код перевірки кластерного аналізу

Кінцевою метою кластеризації в геноміці є виявлення груп генів, що експресуються спільно, які мають функціональну схожість або регуляторні механізми [8].

3.4 Обґрунтування вибору бази даних

Вибір MySQL як системи управління базами даних (СУБД) для веб-застосунку, є обґрунтованим наступними перевагами цієї технології:

– MySQL відомий своєю підтримкою широкого спектру типів даних, включаючи, але не обмежуючись, примітивними типами, JSON, масивами, геопросторовими даними та іншими користувацькими типами. Це робить його ідеальним рішенням для застосунків, що потребують складної обробки та зберігання різноманітних даних, таких як біологічні та геномні послідовності;

– MySQL підтримує розширені можливості для вертикального та горизонтального масштабування. Він може ефективно обробляти великі об'ємами даних та складні запити, що робить його відмінним вибором для високонавантажених систем та застосунків, які швидко зростають;

– MySQL має потужні можливості для виконання складних SQL-запитів, включаючи підтримку підзапитів, об'єднань, віконних функцій та агрегацій. Це дозволяє розробникам ефективно аналізувати дані без потреби переміщення їх в окремі аналітичні системи;

– Як система з відкритим кодом, MySQL має активне співтовариство розробників та користувачів, що постійно працюють над поліпшенням продукту. Це забезпечує регулярні оновлення, покращення безпеки та доступність нових функцій. Крім того, використання відкритого коду може допомогти знизити загальні витрати на ліцензування та експлуатацію [12].

– MySQL пропонує розширені можливості для забезпечення безпеки, включаючи підтримку різних механізмів аутентифікації, шифрування даних у спокої та під час передачі, а також детальне керування доступом на рівні користувачів та ролей.

4 ОПИС ПРОЕКТУВАННЯ ЗАСТОСУНКУ

4.1 Проектування моделі даних

Процес розробки моделі даних для веб-додатку включає кілька важливих етапів, починаючи зі створення діаграми "сутність-зв'язок" (ER-діаграми). Ця діаграма слугує планом, що візуально представляє сутності даних, які мають відношення до аналізу генної експресії, такі як гени, експерименти та рівні експресії, а також зв'язки між цими сутностями. Наступним етапом після ER-діаграми є перетворення цієї концептуальної моделі в SQL-схему. Ця схема визначає структуру бази даних, деталізуючи таблиці, стовпці, типи даних та обмеження, необхідні для ефективного зберігання та управління біологічними даними. Забезпечення підтримки схемою ефективних запитів і цілісності даних має першорядне значення, оскільки це безпосередньо впливає на продуктивність і надійність програми при роботі з великими наборами даних.

Були виявлено такі основні сутності: Проект, Набір даних, Обліковий запис користувача, Зразок, Набір даних експресії, Результат експресії, Запис експресії та Ген. Кожна сутність представляє важливий аспект програми:

– проект (Project). Ця сутність інкапсулює дослідницький проект, деталізуючи такі атрибути, як часові мітки створення і модифікації, публічний статус, право власності (зв'язок з UserAccount), а також асоціації зі Зразком і Набором даних;

– набір даних (Dataset). Представлений як абстрактна сутність, включає підкласи на кшталт ExpressionDataset. Містить атрибути даних, такі як дата створення, статус, опис, ім'я файлу, а також зв'язки з Project і UserAccount;

– профіль Користувача (UserAccount). Ця сутність моделює інформацію про користувача, зокрема ім'я користувача, пароль, роль і статус облікового

запису. Вона також підтримує зв'язок з наборами даних, якими володіє користувач;

– зразок (Sample). Представляє біологічні зразки, що використовуються в експериментах;

– expressionDataset. Підклас Dataset, набір даних експерсій;

– expressionRecord. Представляє окремі вимірювання експресії генів;

– ген (Gene). Описує дані гена;

Детальний опис атрибутів сутностей є важливим аспектом для більш глибокого розуміння та аналізу даних БД:

Проект:

– name. Назва проекту;

– created. Мітка часу, яка вказує, коли проект було створено;

– last modified. Мітка часу, яка вказує, коли проєкт було змінено востаннє;

– isPublic. Булеве значення, яке вказує, чи є проект публічним або приватним.

Набір даних (Dataset):

– type. Представляє тип набору даних;

– creation date. Мітка часу, що вказує, коли було створено набір даних;

– public status. Статус чи є дані приватні або публічні;

– public request date. Мітка часу, яка вказує, коли було зроблено запит на оприлюднення набору даних;

– description. Строка, що містить опис набору даних;

– ім'я файлу. Строка, що представляє ім'я файлу, пов'язаного з набором даних;

– cloud storage path. Строка, що представляє шлях до набору даних у хмарному сховищі;

– статус. Зчислення, що вказує на поточний статус збереження даних з файлу;

– storage type. Зчислення StorageType, що вказує тип використовуваного сховища;

– progress. Число з плаваючою комою, що вказує на прогрес завантаження;

– message. Строка, що містить повідомлення, пов'язані з вивантаженням.

Набір полів таблиці UserAccount:

– username. Строка, що представляє ім'я користувача;

– password. Строка що представляє пароль користувача;

– role. Зчислення, що вказує на роль користувача;

– заблоковано. Булеве значення, що вказує на те, чи заблоковано акаунт користувача;

Набір необхідних полів для «результат експерсій»:

– dataset. Набір даних, що вказує на набір даних, до якого належить цей результат;

– isPublic. Булеве значення, що вказує на те, чи є результат загальнодоступним;

Таблиця sample представляє зразок, що вказує на зразок, пов'язаний з цим результатом;

Запис, ячейка експресії гена:

– value — Дробдне число, що представляє виміряне значення виразу.

Ген:

– original name. Рядок, що представляє оригінальну назву гена;

– name. Рядок, що представляє альтернативну назву гена.

Зразок:

– name. Строка, що представляє назву зразка, яка не може бути нульовою;

– type. Зчислення типу SampleType, що вказує на тип зразка;

– disease type. Строка, що вказує на тип захворювання, пов'язаного з зразком;

– primarySite. Строка, що вказує на місце походження зразка;

– tissueType. зчислення , що вказує на тип тканини;

Взаємозв'язки між сутностями були визначені на основі аналізу домену та кейсів використання:

– Проект (Project) може мати багато Зразків (Samples), але Зразок (Sample) належить одному Проекту (Project);

– Проект (Project) може мати багато Наборів даних (Datasets), але Набір даних (Dataset) належить одному Проекту (Project);

– Користувач (UserAccount) може володіти багатьма Проектами (Projects), але кожен Проект (Project) має одного власника;

– Користувач (UserAccount) може володіти багатьма Наборами даних (Datasets), але кожен Набір даних (Dataset) має одного власника;

– Набір даних (Dataset) має один набір деталей завантаження даних (DatasetUploadDetails);

– Набір експресійних даних (ExpressionDataset) може мати багато Результатів експресії (ExpressionResults), але кожен Результат експресії (ExpressionResult) належить одному Набору даних (Dataset);

– Результат експресії (ExpressionResult) може мати багато Записів експресії (ExpressionRecords), але кожен Запис експресії (ExpressionRecord) належить одному Результату експресії (ExpressionResult);

– Кожен Запис експресії (ExpressionRecord) асоціюється з одним Геном (Gene);

– Зразок (Sample) може мати багато Результатів експресії (ExpressionResults), але кожен Результат експресії (ExpressionResult) асоціюється з одним Зразком (Sample).

Ці зв'язки були візуалізовані на діаграмі "сутність-зв'язок" (ER). Діаграма допомогла виявити будь-які відсутні зв'язки і гарантувала, що модель точно відображає предметну область. Діаграма "сутність-зв'язок" подана на рисунку 4.1.

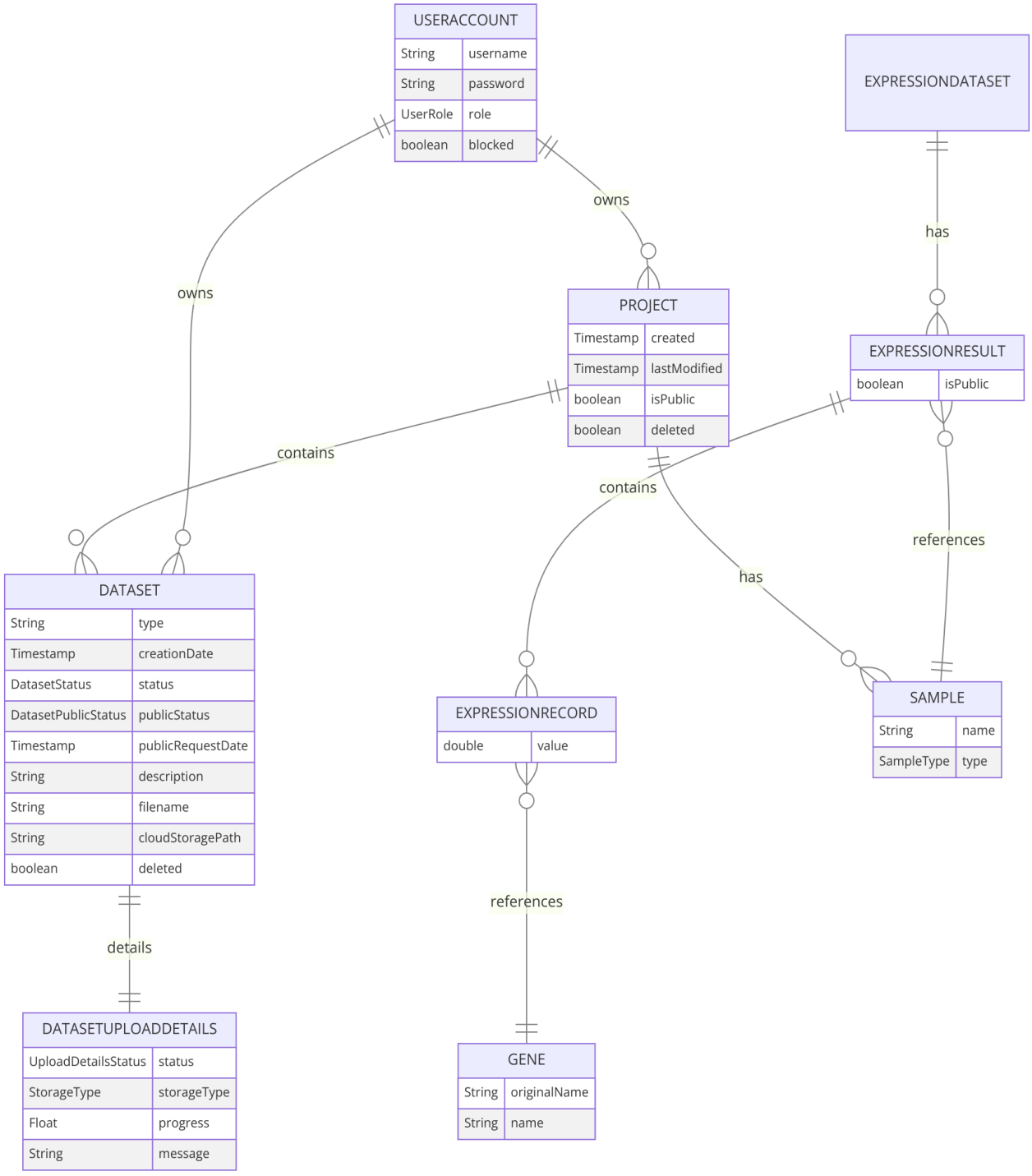

Рисунок 4.1 – Розроблена ER-діаграма

Для розробки логічної моделі буде використана ER-діаграма системи для аналізу та візуалізації біологічних даних. Логічна модель базується на основі ER-діаграми і включає більш детальний опис структури даних та її

обмежень, але без прив’язки до конкретної системи управління базами даних (СУБД).

Саме кінцева логічна модель повинна реалізовувати оптимальну для системи нормалізацію БД та включати визначення сутності і атрибутів, визначення їх типів даних, визначення первинних та зовнішніх ключів, що створюють зв’язок між таблицями. Процес розробки логічної моделі допомагає створити детальну і структуровану модель даних, що можна реалізовувати в СУБД. Логічну модель даних представлено на рисунку 4.2.

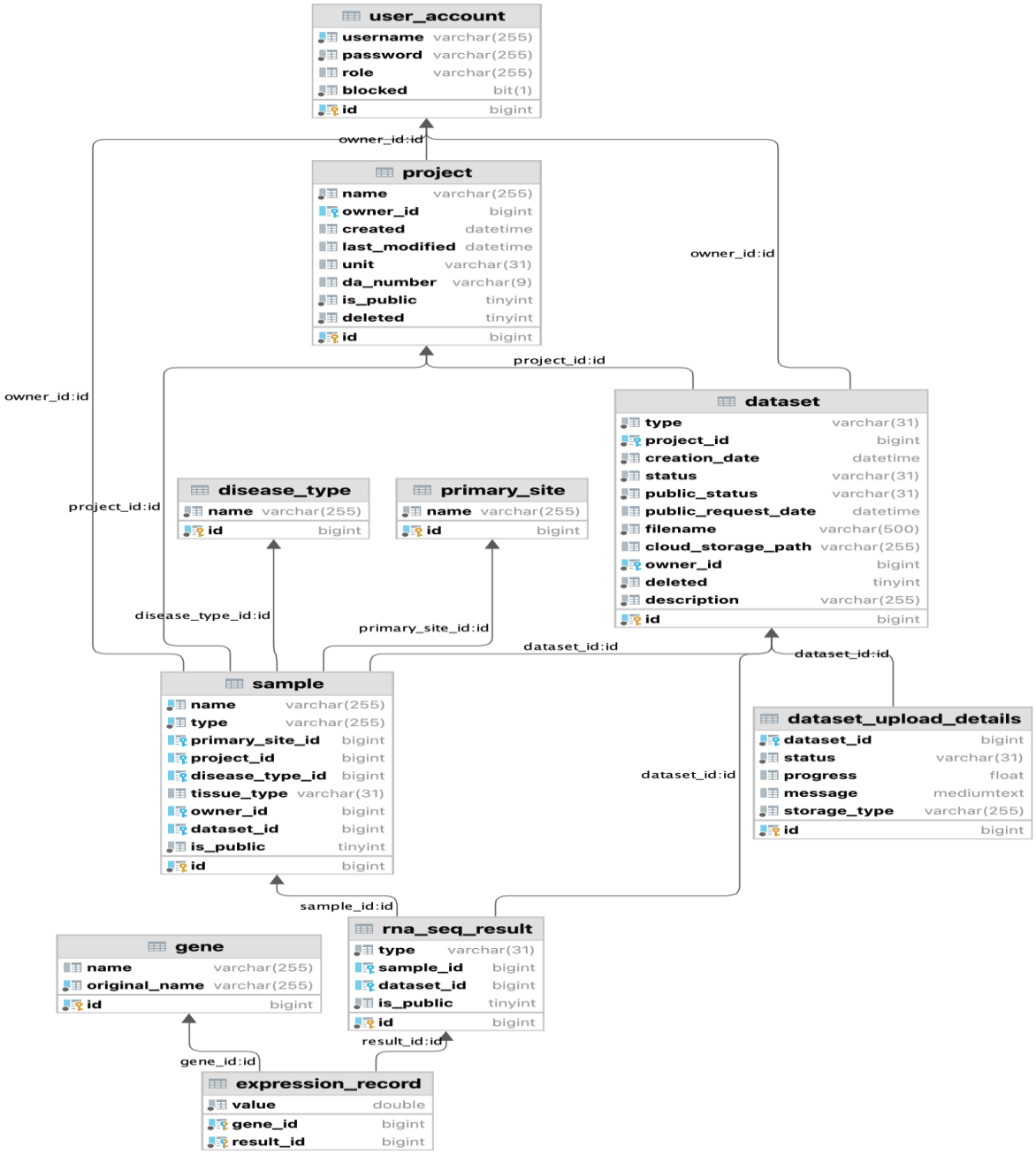

Рисунок 4.2 – Логічна модель даних для системи аналізу та візуалізації біологічних даних

4.2 Розробка серверної частини веб-застосунку аналізу та візуалізації біологічних даних

Контролери в Java в контексті веб-розробки відповідають за обробку HTTP-запитів від клієнтів і взаємодією між моделями та представленнями. Вони виконують роль посередника між моделлю та користувачем, приймаючи запити від клієнтів, оброблюючи їх та визначаючи, що повинно бути у відповіді.

Розглянемо код сервісу, що оброблює запити для отримання і кластеризації даних на основі певних фільтрів.

Клас «DefaultExpressionRecordService» є реалізацією інтерфейсу «ExpressionRecordService», який надає методи для отримання даних та їх кластеризації. Він позначений як @Service, що робить його компонентом Spring, який може бути автоматично виявлений і використаний у програмі [4]. Даний сервіс призначений для обробки запитів користувачів для виконання кластеризації цих даних на основі певних фільтрів. Він забезпечує гнучкість і масштабованість у роботі з великими наборами даних, що дозволить ефективно їх оброблювати. Програмний код сервісу поданий на рисунку 4.3.

```
@Service
public class DefaultExpressionRecordService implements
ExpressionRecordService {
    private final ExprRecordRepo recordRepository;
    private final RecordRepository resultRepository;
    private final ClusterService clusterService;
    @Autowired
    public DefaultExpressionRecordService(ExprRecordRepo
recordRepository,
                                          RecordRepository
resultRepository,
                                          ClusterService
clusterService) {
        this.recordRepository = recordRepository;
```

```
        this.resultRepository = resultRepository;
        this.clusterService = clusterService;
    }
    @Override
    public List<ItemWithRecordSummaries> getData(List<FilterQuery>
queries) {
        final List<AggregateExpressionRecordDto> records =
getDataByQuery(queries);
        final Map<IdAndNameDto, List<EntityRecordSummary>> map =
                records.stream()
    .collect(groupingBy(AggregateExpressionRecordDto::getPrimarySite);
        return map.keySet()
                .stream()
                .map(key -> new ItemWithRecordSummaries(key.getId(),
key.getName(), map.get(key)))
                .collect(Collectors.toList());
    }
    @Override
    public ClusteredDataSummary getClusteredData(List<FilterQuery>
queries) {
        final boolean entitiesAmountExceeded =
clusterService.entitiesAmountExceeded(queries, GEN);
        final boolean samplesAmountExceeded =
clusterService.samplesAmountExceeded(queries, GEN, resultRepository);
        if (entitiesAmountExceeded || samplesAmountExceeded) {
            return
ClusteredDataSummary.builder().setLimitExceeded(true).build();
        }
        final List<AggregateExpressionRecordDto> records =
getDataByQuery(queries);
        if (records.isEmpty()) {
            return ClusteredDataSummary.builder().build();
        }
        return clusterService.getClusteredDataSummary(records);
    }
    @Override
    public ClusteredDataSummary getMatrixData(List<FilterQuery>
queries) {
        final boolean entitiesAmountExceeded =
clusterService.entitiesAmountExceeded(queries, GEN);
        final boolean samplesAmountExceeded =
clusterService.samplesAmountExceeded(queries, GEN, resultRepository);
        final List<AggregateExpressionRecordDto> records =
getDataByQuery(queries);
        if (records.isEmpty()) {
            return ClusteredDataSummary.builder().build();
        }
        return entitiesAmountExceeded || samplesAmountExceeded
                ? clusterService.getHeatmapData(records, true)
                : clusterService.getClusteredDataSummary(records);
    }
```

```
    private List<AggregateExpressionRecordDto>
getDataByQuery(List<FilterQuery> queries) {
        final List<Specification<ExpressionRecord>> specifications =
new ArrayList<>();
        final Specification<ExpressionRecord> specification =
joinWithAnd(specifications);
        return recordRepository.findRecords(specification); }
    private Specification<ExpressionRecord>
joinWithAnd(List<Specification<ExpressionRecord>> specifications) {
        return specifications.stream()
                .reduce((spec1, spec2) -> spec1.and(spec2))
                .orElse(null);
    }
}
```

Рисунок 4.3 – Програмний код сервісу «DefaultExpressionRecordService»

Розглянемо код контролера, який використовується для управління функціоналом користувачів. «UserController» надає API для управління користувачами. Він використовує сервіс «UserService» для операції з користувачами та «AuthenticationService» для аутентифікації. Саме цей контролер надає набір інтерфейсів для управління, включаючи реєстрацію, авторизацію, оновлення профілю, перевірку статусу авторизації, також, блокування та розблокування користувачів. На рисунку 4.4 подано програмний код контролеру.

```
@RestController
@RequestMapping("/api/user")
public class UserController {
    private final UserService userService;
    private final AuthenticationService authenticationService;
    /**
     * Constructor to provide required services.
     */
    @Autowired
    public UserController(UserService userService,
                          AuthenticationService authenticationService)
{
        this.userService = userService;
        this.authenticationService = authenticationService;
    }
    /**
     * Retrieves details about currently logged in user.
     *
     * @return logged in user details
```

```
     */
    @GetMapping
    public UserAccountDto getCurrent(HttpServletRequest request)
throws ApplicationException {
        UserAccountDto userDto = userService.getCurrent();
userDto.setSessionInactiveTime(request.getSession().getMaxInactiveInte
rval());
        return userDto;
    }
    /**
     * Checks whether a user is currently logged in.
     *
     * @return true if user is logged in, otherwise false
     */
    @GetMapping("/logged-in")
    public boolean isLoggedIn() {
        return userLoggedIn();
    }
    /**
     * Does login.
     */
    @PostMapping("/login")
    public UserAccountDto login(@RequestBody UserAuthRequest
authQuery,
                                HttpServletRequest request) throws
ApplicationException {
        UserAccountDto userDto =
authenticationService.login(authQuery);
userDto.setSessionInactiveTime(request.getSession().getMaxInactiveInte
rval());
        return userDto;
    }
    /**
     * Does logout.
     */
    @GetMapping("/logout")
    public String logout(HttpServletRequest request,
HttpServletResponse response) throws ApplicationException {
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        if (auth != null) {
            return "OK";
        }
        ExceptionCode.USER_LOGOUT_ANONYMOUS_DENIED.throwException();
        return "NOT OK";
    }
    /**
     * Does registration.
     */
    @PostMapping("/registration")
```

```
    public String registration(@RequestBody UserRegistrationRequest
userRegReq) throws ApplicationException {
        UserAccount newUser = new UserAccount();
        newUser.setUsername(userRegReq.getUsername());
        newUser.setPassword(userRegReq.getPassword());
        newUser.setRole(UserRole.USER);

        userService.registerNew(newUser);
        return "OK";
    }
    /**
     * Does update of profile.
     */
    @PutMapping("/profile")
    public String updateProfile(@RequestBody UserRegistrationRequest
request) throws ApplicationException {

        userService.changePassword(request);

        return HttpStatus.OK.getReasonPhrase();
    }

    @GetMapping("/list")
    public Page<UserAccountDto> getPage(Pageable pageable) {
        return userService.getPage(pageable);
    }

    @PutMapping("/update")
    public UserAccountDto setUserAccountRole(@RequestBody
UserUpdateRequest request) throws ApplicationException {
        return userService.changeRole(request);
    }


    @PutMapping("/{id}/block")
    public void blockUser(@PathVariable("id") Long userId) {
        userService.setBlocked(userId, true);
    }

    @PutMapping("/{id}/unlock")
    public void unlockUser(@PathVariable("id") Long userId) {
        userService.setBlocked(userId, false);
    }
}
```

Рисунок 4.4 – Програмний код контролера «UserController»

Розглянемо контролер, що є частиною сервісу, який оброблює запити пов'язані з еспресією даних. Він використовує Spring Framework для обробки

HTTP-запитів і включає три основні методи для взаємодії з клієнтами. На рисунку 4.5 подано програмний код контролера «ExpressionController».

```
@RestController
@RequestMapping("/api/expression")
@RequiredArgsConstructor
public class ExpressionController {

    private static final Logger LOGGER =
LoggerFactory.getLogger(ExpressionController.class);

    private final ExpressionRecordService expressionService;
    private final FilterService filterService;

    /**
     * Returns expression data.
     */

    @GetMapping("/data")
    public List<ItemWithRecordSummaries> getExpressionData(
        @RequestParam(name = "queries", required = false)
List<FilterQuery> queries) {
        return expressionService.getData(queries);
    }


    @GetMapping("/data/clustered")
    public ClusteredDataSummary getHeatMapData(
        @RequestParam(name = "queries", required = false)
List<FilterQuery> queries) {
        return expressionService.getClusteredData(queries);
    }


    @GetMapping("/options")
    public ModelFilters getOptions(

        @RequestParam(name = "queries", required = false)
List<FilterQuery> queries) {
        return filterService.getOptions(queries);
    }
}
```

Рисунок 4.5 – Програмний код контролера «ExpressionController»

4.3 Розробка інтерфейсу користувача

Створення графічного інтерфейсу користувача (GUI) включає в себе розробку візуальної частини програми, яка дозволяє користувачам взаємодіяти з додатком через графічні елементи, такі як кнопки, поля введення, меню, діалогові вікна тощо. Основні етапи включають планування дизайну інтерфейсу, створення макетів (layouts), додавання обробників подій для взаємодії з користувачем, а також тестування інтерфейсу на зручність використання і ефективність.

При розробці GUI важливо дотримуватись принципів зручності та доступності, щоб користувачі могли легко знаходити потрібні функції і використовувати додаток без зайвих труднощів. Використання сучасних дизайнів, таких як Material Design або Fluent Design, може значно покращити зовнішній вигляд і досвід користувача. Після створення основних елементів інтерфейсу, проводиться інтеграція з логікою програми, що дозволяє забезпечити коректну роботу всіх функцій при взаємодії з користувачем.

Розглянемо розроблений інтерфейс користувача для авторизації та реєстрації в системі забезпечує зручний і інтуїтивно зрозумілий процес входу та створення облікового запису. Користувачі можуть легко ввести свої облікові дані для входу через просту форму авторизації, яка включає поля для введення електронної пошти та пароля, а також кнопку для входу.

Якщо користувач ще не має облікового запису, він може перейти до форми реєстрації через відповідне посилання. Форма реєстрації містить поля для введення необхідної інформації, такої як ім'я, електронна пошта, пароль та підтвердження пароля.

Також передбачені механізми перевірки введених даних для забезпечення правильності та безпеки інформації. Після успішної реєстрації або авторизації користувач автоматично перенаправляється на головну сторінку системи. Форму реєстрації та авторизації подано на рисунку 4.6 та 4.7

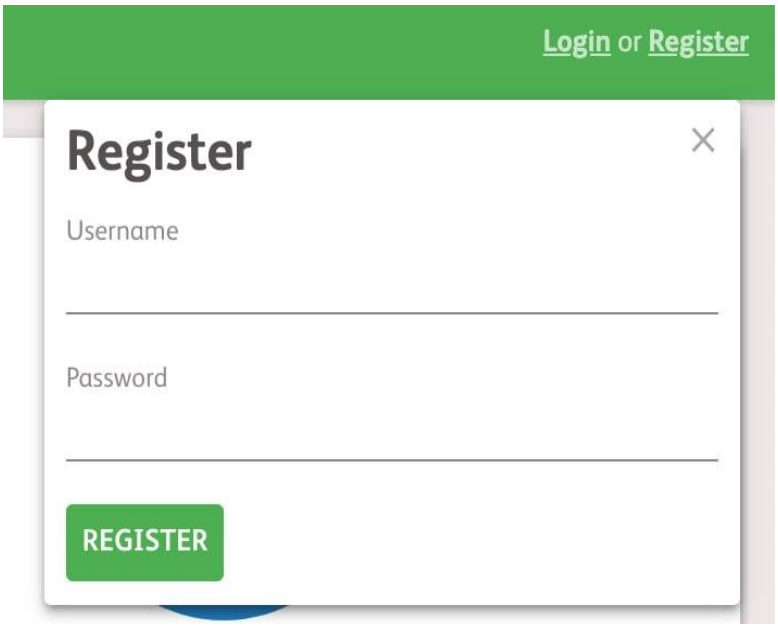

Рисунок 4.6 – Інтерфейс користувача форми реєстрації

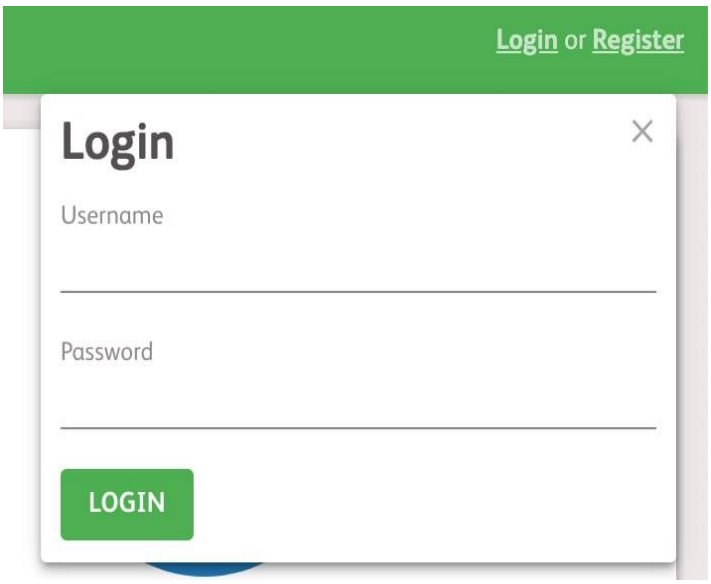

Рисунок 4.7 – Інтерфейс користувача форми авторизації

Важливою частиною інтерфейсу користувача є адміністрування користувачів. На сторінці адміністрування можна побачити загальну кількість користувачів системи, їх статус та іншу інформацію. Найголовніше в цьому інтерфейсі – це можливість блокувати або розблоковувати користувачів системи, що буде надавати або забирати доступ входу. Даний інтерфейс подано на рисунку 4.8.

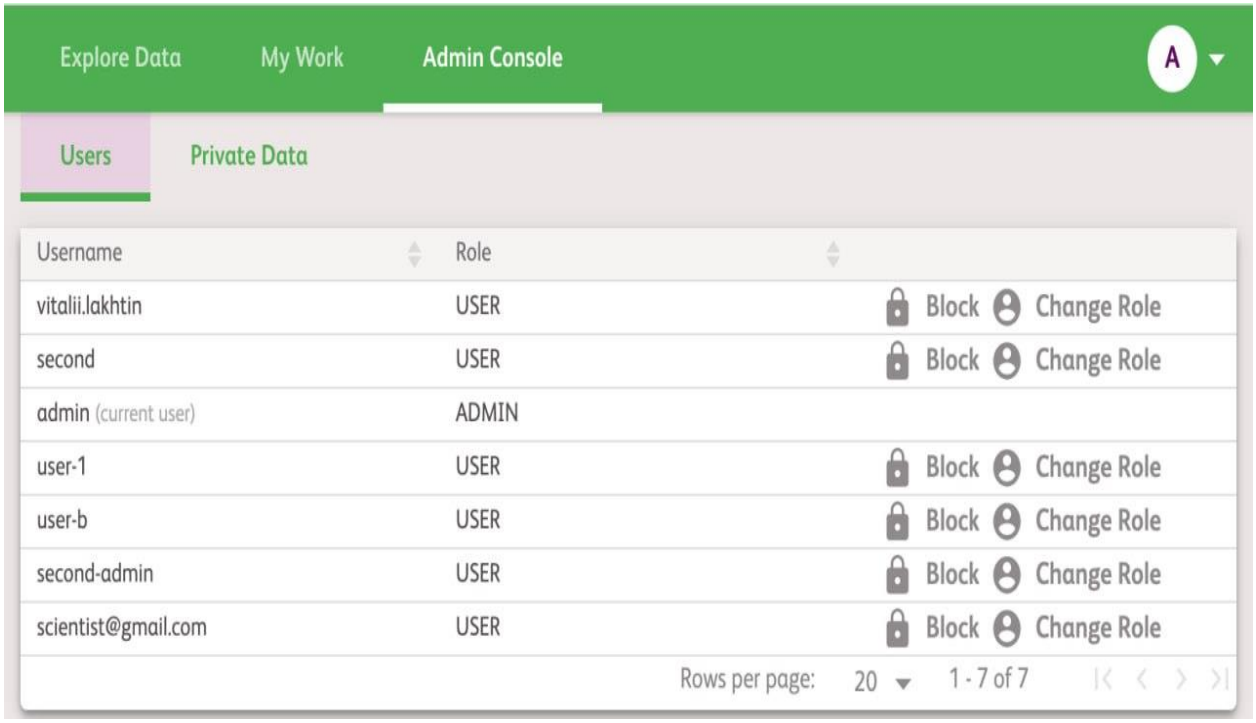

Рисунок 4.8 – Інтерфейс адміністрування користувачів

Програмний код даного інтерфейсу був створений з використанням бібліотеки Polymer. Він забезпечує графічний інтерфейс для адміністрування користувачів у вигляді таблиці з можливістю сортування, пагінації та виконання дій над користувачами. На рисунку 4.9 подано програмний код інтерфейсу адміністрування.

```
<dom-module id="admin-users">
    <template>
        <div class="view-list-container">

<vaadin-grid items="[[_list]]"
on-sorter-changed="_handleSorterChange" id="grid">

<vaadin-grid-column resizable>
                    <template class="header">
                        <vaadin-grid-sorter path="username"><div
class="header-cell">Username</div></vaadin-grid-sorter>
                    </template>
                    <template>
                        <div class="cell">[[item.username]] <span
class="description"
hidden$="[[!_isUserCurrent(_currentUser,
```

```
item.id)]]">(current user)</span></div>
                    </template>
                </vaadin-grid-column>
                <vaadin-grid-column resizable>
                    <template class="header">
                        <vaadin-grid-sorter path="role"><div
class="header-cell">Role</div></vaadin-grid-sorter>
                    </template>
                    <template>
                        <div class="cell">[[item.role]]</div>
                    </template>
                </vaadin-grid-column>

 <vaadin-grid-column flex-grow="0" width="230px">
                    <template class="header">
                        <div class="header-cell"></div>
                    </template>
                    <template>
                        <div class="cell">
                            <div
hidden$="[[_isUserCurrent(_currentUser, item.id)]]">

<button on-tap="_blockUser"

on-mousedown="_preventFocusBehavior"

class="trigger-button"

 hidden$="[[item.blocked]]">

                                    <iron-icon
icon="icons:lock"></iron-icon>
                                    <span class="trigger-button-
text">Block</span>
                                </button>
                                <button on-tap="_unblockUser"
                                        on-
mousedown="_preventFocusBehavior"
                                        class="trigger-button"
                                        hidden$="[[!item.blocked]]">
                                    <iron-icon icon="icons:lock-
open"></iron-icon>
                                    <span class="trigger-button-
text">Unblock</span>
                                </button>

<button on-tap="_changeRole"

on-mousedown="_preventFocusBehavior"
                                        class="trigger-button">
```

```
<iron-icon icon="icons:account-circle"></iron-icon>
                                    <span class="trigger-button-
text">Change role</span>
                                </button>
                            </div>
                        </div>
                    </template>
                </vaadin-grid-column>
            </vaadin-grid>

            <table-pagination
on-table-page-changed="_handlePaginationChange"
                                    page-size="[[_pagination.pageSize]]"
                                    current-
page="[[_pagination.pageNumber]]"
                                    total-
pages="[[_pagination.totalPages]]"
                                    total-
rows="[[_pagination.totalElements]]"></table-pagination>

        </div>

        <portal-spinner class="spinner"
active="[[_isLoading]]"></portal-spinner>
    </template>
    <script>
{
        const {PDX_PORTAL_ACTIONS, DIALOG_ACTIONS, RouteSegments} =
PdxPortalContext;

        class PdxAdminUsers extends
PdxPortalContext.ReduxMixin(Polymer.Element) {
            static get is() {
                return 'admin-users';
            }

            static get properties() {
                return {
                    _isLoading: {
                        type: Boolean,
                        statePath: 'contents.userListLoading'
                    },

                    page: {
                        type: Object,
                        observer: '_handlePageChange'
                    },

                    _currentUser: {
                        type: Object,
```

```
                        statePath: 'contents.currentUser'
                    },

                    _list: {
                        type: Array,
                        statePath: 'contents.userList'
                    },

                    _pagination: {
                        type: Object,
                        statePath: 'contents.userListPagination'
                    },

                    _sorting: {
                        type: Object,
                        statePath: 'contents.userListSorting'
                    }
                };
            }

                if (JSON.stringify(this._sorting) !==
JSON.stringify(sorting))
{
                    this._sorting = sorting;
                    this._loadUserList(this._pagination,
this._sorting);
                }
            }

        window.customElements.define(PdxAdminUsers.is, PdxAdminUsers);
    }
</script>
</dom-module>
```

Рисунок 4.9 – Програмний код інтерфейсу адміністрування

5 ТЕСТУВАННЯ ЗАСТОСУНКУ

5.1 Мета тестування програмного застосунку

Тестування є критично важливим етапом життєвого циклу розробки програмного забезпечення, який гарантує, що розроблений веб-застосунок відповідає заданим вимогам і працює оптимально. Застосунок повинен точно обробляти великі масиви даних, забезпечувати точну візуалізацію і зберігати надійність за різних умов. У цьому розділі детально описано тестування, яка використовувалось для перевірки функціональності, продуктивності та надійності веб-застосунку.

5.2 Модульне тестування

Для перевірки функціональності окремих компонентів програми були проведені модульні тести. Кожен модуль тестувався ізольовано, щоб забезпечити коректність роботи. Для написання та виконання модульних тестів використовувався фреймворк Java JUnit, який гарантував, що кожен компонент відповідає своїм проектним специфікаціям. На рисунку 5.1 подано програмний код модульного тесту для збереження проекту та даних у проект.

```
@DisplayName("ProjectService")
class ProjectServiceTest {
    @Mock
    private SampleRepository sampleRepository;

    @Mock
    private DataService DataService;

    @Mock
    private UserReposiotry userReposiotry;

    @Mock
```

```
    private ProjectRepository repository;
    private ProjectService service;
    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks(this);
        service = new ProjectService(repository, sampleRepository,
userReposiotry, DataService);
    }
    @Test
    @DisplayName("Should save values")
    void save() throws NullFoundException {
        when(repository.findAllByNameIn(any())).thenReturn(new
ArrayList<>());
        when(repository.saveAll(any())).thenAnswer(invocation -> {
            final Iterable<? extends Project> argument =
invocation.getArgument(0);
            return new ArrayList<>(argument);
        });
        final String[] inputValues = {"One", "Two"};
        final Map<String, Project> actual =
service.save(Arrays.asList(inputValues));
        assertEquals(inputValues.length, actual.keySet().size());
        assertTrue(actual.containsKey(inputValues[0]));
        assertTrue(actual.containsKey(inputValues[1]));
        final Project project1 = new Project(inputValues[0]);
        final Project project2 = new Project(inputValues[1]);
        project1.setPublic(true);
        project2.setPublic(true);
        assertEquals(project1.getName(),
actual.get(inputValues[0]).getName());
        assertEquals(project2.getName(),
actual.get(inputValues[1]).getName());
    }
    @Test
    @DisplayName("Should throw an error if the project with the
provided name already exists")
    void testDuplicateNames() {
        final String projectName = "project";
        final Project project = new Project(projectName);
        when(repository.findAllByNameIn(any())).thenReturn(new
ArrayList<>());

when(repository.findByName(any())).thenReturn(Optional.of(project));
        assertThrows(ValidationException.class, () ->
service.savePrivate(projectName, "TPM", null));
    }
    @Test
    @DisplayName("Should get projects for current user")
    void shouldGetPage() {
        final int pageNumber = 0;
        final int pageSize = 5;
```

```
        final Pageable pageRequest = PageRequest.of(pageNumber,
pageSize);
        final UserAccount owner = createUserAccount();
        final Project project = createProject(owner);
        final List<Project> projects =
Collections.singletonList(project);
        final Page<Project> page = new PageImpl<>(projects,
pageRequest, projects.size());
        when(repository.getPage(anyLong(),
any(Pageable.class))).thenReturn(page);
        final ProjectDto projectDto = ProjectDto.from(project);
        final Page<ProjectDto> projectPage =
service.getPageForCurrentUser(pageRequest);
        then(repository).should(times(1)).getPage(USER_ID,
pageRequest);
        final List<ProjectDto> pageContent = projectPage.getContent();
        assertAll("Project Page",
            () -> assertEquals(1L, projectPage.getTotalElements()),
            () -> assertEquals(pageNumber, projectPage.getNumber()),
            () -> assertEquals(pageSize, projectPage.getSize()),
            () -> assertEquals(Collections.singletonList(projectDto),
pageContent),
            () -> assertEquals(2,
pageContent.get(0).getDatasets().size())
        );
    }
    private UserAccount createUserAccount() {
        UserAccount owner = new UserAccount();
        owner.setId(USER_ID);
        owner.setUsername(TEST_USERNAME);
        owner.setPassword(TEST_PASSWORD);
        owner.setRole(UserRole.USER);
        return owner;
    }
    private Project createProject(UserAccount owner) {
        Project project = new Project("Project 1");
        project.setId(1L);
        project.setOwner(owner);
        Dataset expressionDataset = new ExpressionDataset();
        expressionDataset.setType(DatasetType.EXPRESSION);
        expressionDataset.setOwner(owner);
        Dataset sampleDataset = new SampleDataset();
        sampleDataset.setType(DatasetType.MODEL);
        sampleDataset.setOwner(owner);
        project.getDatasets().addAll(Arrays.asList(expressionDataset,
sampleDataset));
        return project;
    }
}
```

Рисунок 5.1 – Модульні тести збереження проекту та даних у проект

5.3 Інтеграційне тестування

Інтеграційні тести були розроблені для перевірки взаємодії між різними компонентами програми. Етап інтеграційного тестування забезпечив безперебійну роботу таких модулів, як введення даних, фільтрація та компоненти візуалізації. Тестові сценарії були розроблені для імітації реальних завдань аналізу даних, перевіряючи правильність проходження даних через систему і точність отримання та відображення результатів.

Було розроблені критичні тестові сценарії для перевірки інтеграції фільтрації даних, генерації таблиць і візуалізації теплових карт.

Вибір фільтра та відображення таблиці:

– мета. Протестувати взаємодію між модулем фільтрації даних і компонентом відображення таблиці;

– хід. Завантажте в програму набір даних про експресію генів.

Необхідно застосувати певні фільтри для вибору підмножини генів, типу зразка, проекта. Переконатися, що відфільтровані дані коректно відображаються в табличному форматі.

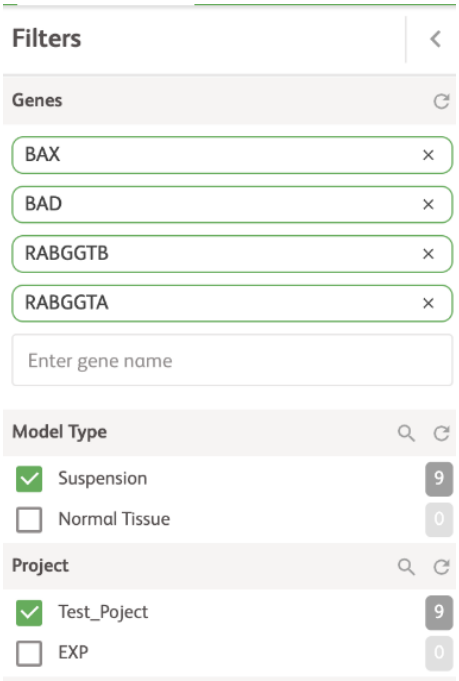

Рисунок 5.2 – Вибір фільтрів та генів для відображення даних

Очікуваний результат. Таблиця повинна відображати лише ті гени та зразки, які відповідають критеріям фільтрації, з усіма відповідними даними про експресію генів.

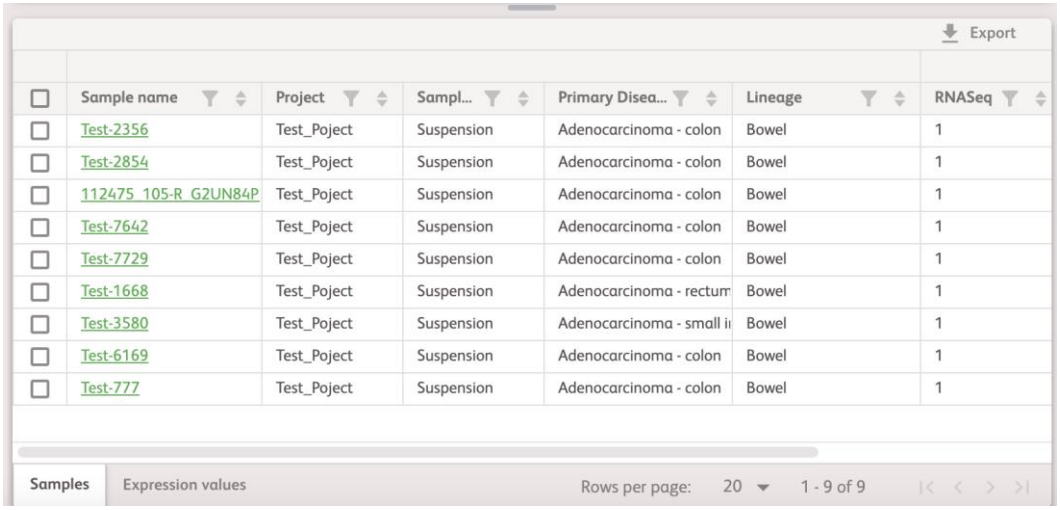

Рисунок 5.3 – Відображення зразків

Вибір фільтрів та створення теплової карти:

– мета. Протестувати взаємодію між модулем фільтрації даних і компонентом візуалізації теплової карти;

– хід. Завантажте набір даних експресії генів у програму. Застосуйте певні фільтри для вибору підмножини генів.

На рисунку 5.4 подано обрані фільтри для проведення тестування.

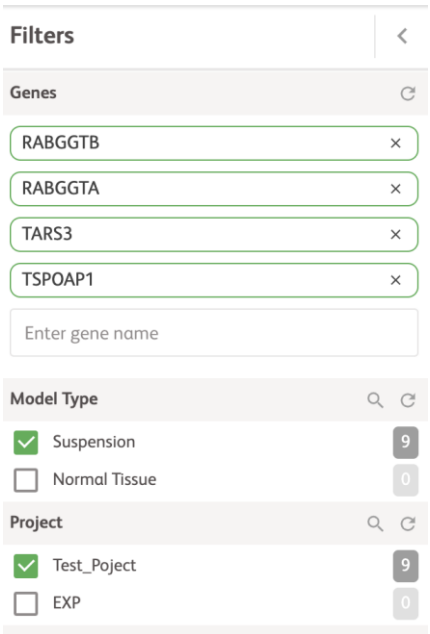

Рисунок 5.4 – Вибір фільтрів для тесту

Створіть теплову карту на основі відфільтрованих даних, використовуючи алгоритми кластеризації для групування генів зі схожими паттернами експресії.

Очікуваний результат: теплова карта повинна точно візуалізувати відфільтровані дані генів, з чіткими кластерами, що представляють групи генів зі схожими профілями експресії.

На рисунку 5.5 подано візуалізацію тесту, для якого було обрано фільтри, що провести тестування.

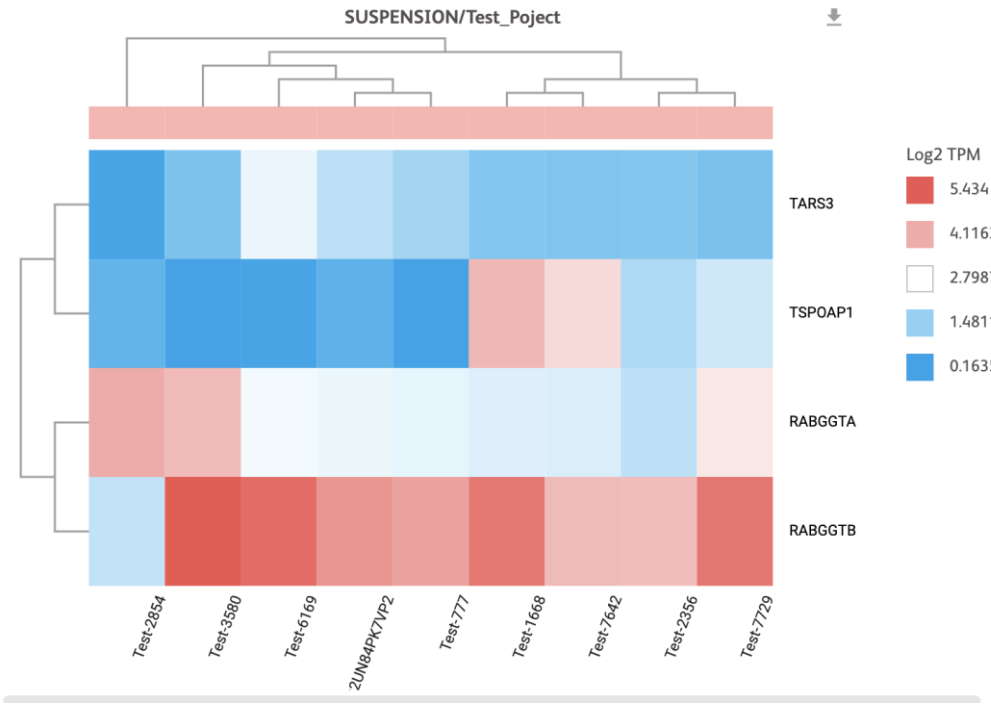

Рисунок 5.5 – Візуалізація за заданими фільтрами

На етапі інтеграційного тестування були виявлені незначні проблеми, пов'язані з реакцією компонента теплових карт під час великих навантажень даних, які згодом були оптимізовані. Загалом, тести підтвердили, що модулі додатку працюють злагоджено, надаючи користувачам надійні та точні інструменти для аналізу та візуалізації даних.

Наступний тестовий приклад має на меті перевірити функціональність блокування та розблокування користувацьких функцій, якими керує

адміністратор у веб-додатку. Ці функції є важливими для підтримки цілісності та безпеки додатку, гарантуючи, що адміністратори можуть ефективно керувати доступом користувачів.

Опис: адміністратор може успішно заблокувати та розблокувати обліковий запис користувача, і що доступ користувача до програми буде належним чином обмежено та відновлено.

Передумови:

– користувач admin увійшов до програми;

– в системі існують облікові записи користувачів;

– користувач, якого потрібно заблокувати/розблокувати, ідентифікований і доступний з панелі адміністратора.

Етапи тестування:

– перейдіть до Керування користувачами;

– крок. Адміністратор переходить до розділу управління користувачами в додатку;

– Очікуваний результат. Відображається інформаційна панель керування користувачами зі списком усіх зареєстрованих користувачів.

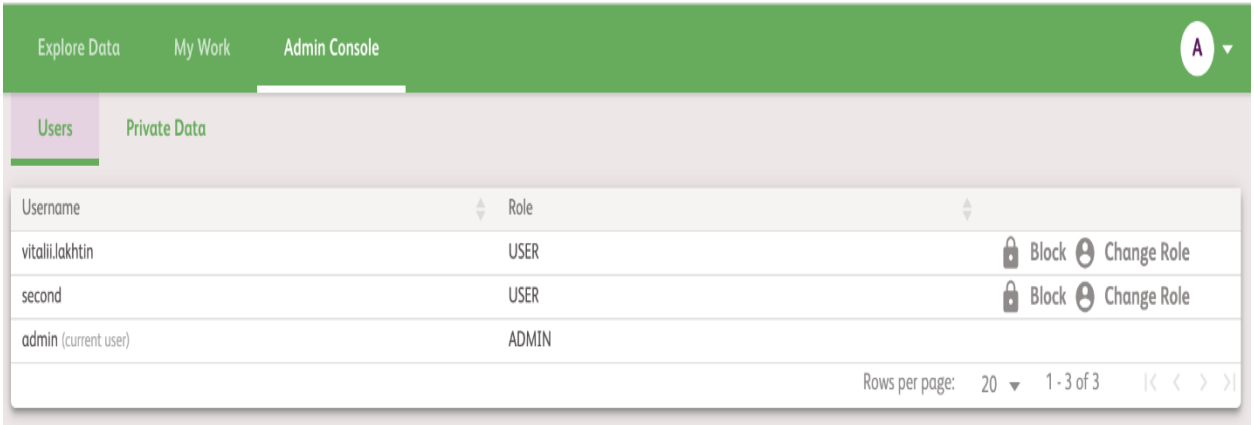

Рисунок 5.6 – Список користувачів

– крок. Адміністратор вибирає користувача зі списку для блокування;

– очікуваний результат. Відображається інформація про користувача, включно з опціями для його блокування;

– крок. Адміністратор натискає кнопку "Заблокувати користувача";

– очікуваний результат. Статус користувача змінюється на "Заблокований" в панелі управління користувачами, а користувач отримує сповіщення про свій заблокований статус (якщо сповіщення реалізовано);

Необхідно переконатися, що доступ користувача заблоковано:

– крок. Спробуйте увійти за допомогою облікових даних заблокованого користувача;

– очікуваний результат. Програма відмовить у доступі, відобразивши повідомлення про те, що доступ до сторінки заблокований.

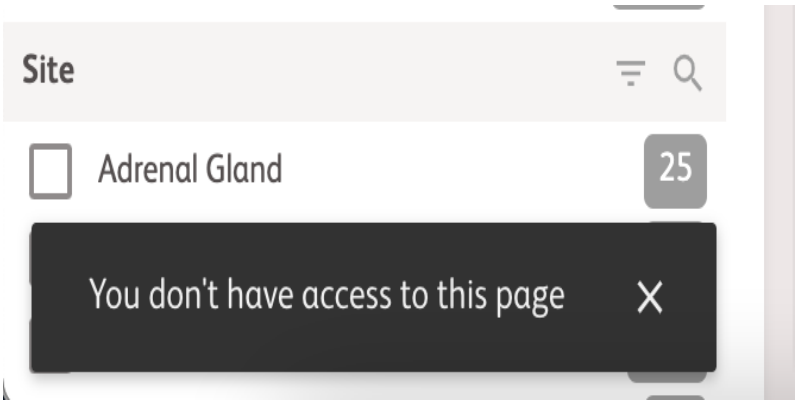

Рисунок 5.7 – Користувач не може зайти в застосунок

Розблокування користувача:

– крок. Адміністратор вибирає раніше заблокованого користувача з панелі керування користувачами;

– очікуваний результат. Відображається інформація про користувача, включно з можливістю розблокувати його;

– крок. Адміністратор натискає кнопку "Розблокувати";

– очікуваний результат. Статус користувача змінюється на "Активний" або "Розблокований" на панелі керування користувачами, а користувач отримує сповіщення про розблокування свого статусу (якщо сповіщення реалізовано).

Необхідно переконатися, що доступ користувача відновлено:

– крок. Спробуйте увійти за допомогою облікових даних розблокованого користувача;

– очікуваний результат. Додаток надає доступ, що дозволяє користувачеві увійти в систему і отримати доступ до свого облікового запису у звичайному режимі.

Завантаження файлу та перевірка його наявності в таблиці завантажених файлів:

– передумови. Користувач увійшов до системи, існуючі проекти доступні для вибору, користувач має дозвіл на завантаження файлів.

Кроки перевірки:

– перейдіть до розділу "Завантаження", виберіть проект, виберіть файл;

– очікувані результати. Ім'я файлу з'являється поруч з кнопкою "Вибрати файл";

– крок. Користувач натискає кнопку "Завантажити";

– очікуваний результат. Відображається хід завантаження, після чого з'являється повідомлення про успішне додавання файла у чергу;

– крок. Користувач перевіряє таблицю завантажених файлів;

– очікуваний результат. Завантажений файл відображається у списку з деталями, такими як ім'я файлу і дата завантаження

Додана інформація для тесту подана на рисунку 5.8.

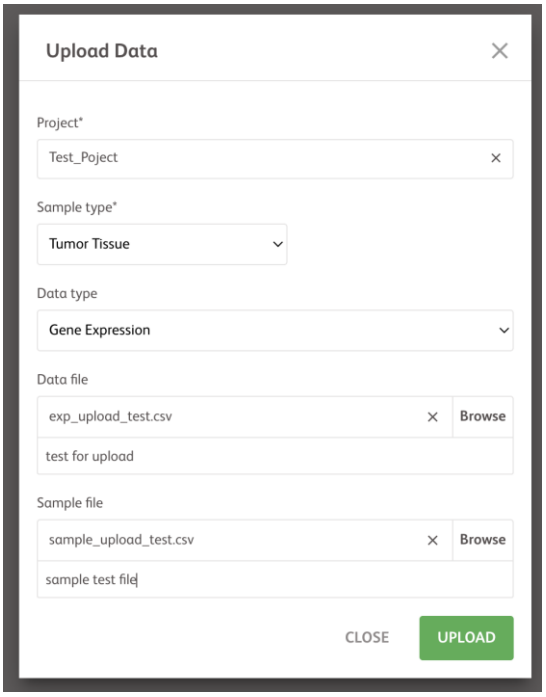

Рисунок 5.8 – Вибір файлу для завантаження.

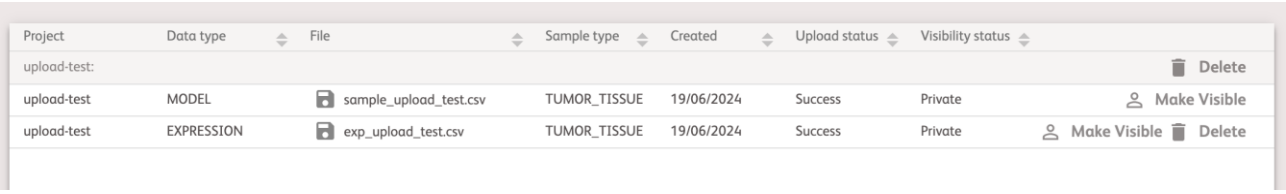

Рисунок 5.9 – Результат завантаження

5.4 Тестування навантаження

Зважаючи на те, що програма має обробляти великі набори даних про експресію генів, тестування продуктивності є важливим. Інструмент Apache Jmeter був використан для моделювання сценаріїв з високим навантаженням і вимірювання часу відгуку, пропускної здатності та використання ресурсів додатку. Були встановлені контрольні показники продуктивності, а застосунок був оптимізований на основі результатів тестування, щоб забезпечити його відповідність необхідним критеріям продуктивності.

Перший тестовий сценарій має на меті оцінити продуктивність і надійність веб-застосунку при одночасному завантаженні багатьох файлів.

Основна увага приділяється тому, щоб переконатися, що система може обробляти масові завантаження файлів без збоїв і що файли ефективно обробляються за допомогою черги.

Проведемо тестування продуктивності при масовому завантаженні файлів і обробці в черзі.

Етапи тестування:

– крок. Використовуйте автоматизований скрипт або інструмент (Apache JMeter) для імітації одночасного завантаження великої кількості файлів (100 файлів). Очікуваний результат: Всі файли успішно завантажуються в додаток без помилок.

Моніторинг продуктивності системи під час завантаження:

– крок. Відстежуйте ключові показники продуктивності, такі як використання процесора, пам'яті, дискового вводу/виводу та пропускної здатності мережі під час процесу завантаження файлів;

– очікуваний результат. Система повинна підтримувати прийнятний рівень продуктивності, і жоден ресурс не повинен стати вузьким місцем;

– крок. Перевірте систему керування чергою, щоб переконатися, що всі завантажені файли ставляться в чергу на обробку;

– очікуваний результат. Всі файли з'являються в черзі, і жоден файл не втрачається і не пропускається.

На рисунку 5.10 подано отриманий графік навантаження, на якому зображено використання процесора та навантаження мережі.

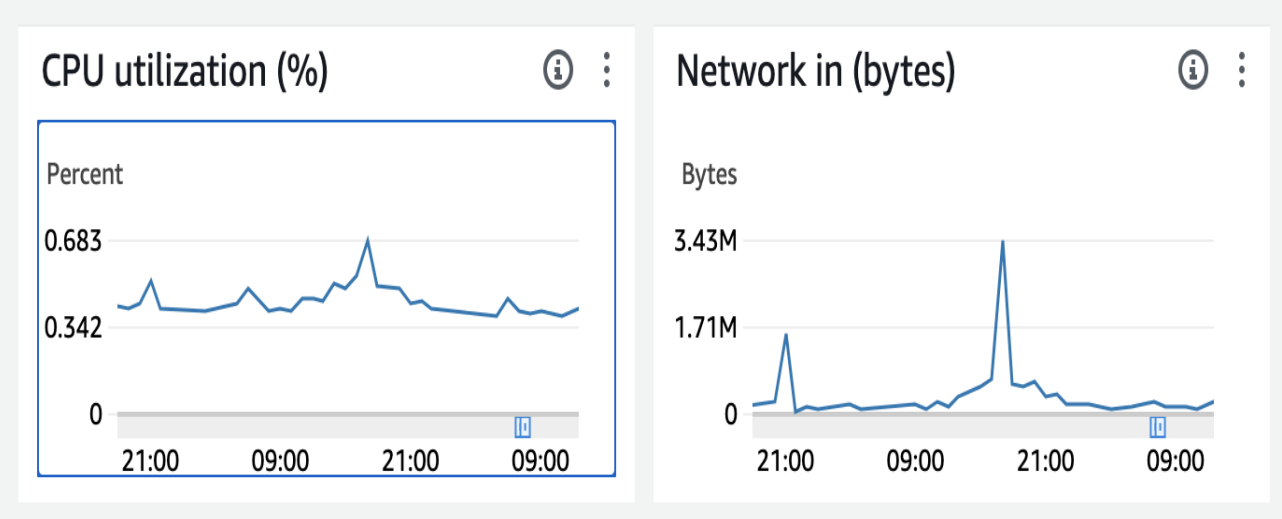

Рисунок 5.10 – Графік навантаження

ВИСНОВКИ

Під час проведення аналізу предметної області було детально розглянуто існуючі аналоги для візуалізації та аналізу біологічних даних. Головним конкурентом нашої системи в області аналізу та візуалізації біологічних даних є Galaxy проект, який має багатий функціонал для автоматизованого аналізу та візуалізації генетичних даних.

Проектуючи веб-застосунок для аналізу біологічних даних, було проведено функціональне моделювання бізнес-процесів і бізнес-функцій системи, які були визначені в ході роботи. Проектування бізнес-функцій та їх декомпозиція дозволили чітко зрозуміти та визначити вимоги до бізнес-логіки системи та здійснити моделювання процесів і їх взаємозв’язків за допомогою уніфікованої мови моделювання (UML). Цей підхід забезпечив структурованість і впорядкованість під час розробки, що є важливим аспектом при створенні надійної системи.

Серверна частина застосунку була реалізована з використанням Java та AWS, що дозволило скористатися потужними інструментами для розробки, такими як управління маршрутизацією, автентифікацією та авторизацією користувачів, а також інтеграцією з базами даних через Amazon RDS. Клієнтська частина застосунку була реалізована з використанням сучасних фреймворків та бібліотек, що забезпечило зручність і гнучкість у створенні інтерфейсу користувача.

Підсумовуючи, детальне проектування бізнес-функцій з використанням сучасних технологій та інструментів для написання програмного забезпечення дозволило створити потужний та гнучкий до змін веб-застосунок для аналізу та візуалізації біологічних даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Петров Е.Г., Новожилова М.В., Гребеннік І.В., Соколова Н.А. Методи та засоби прийняття рішень у соціально-економічних та технічних системах: Херсон: Олді – плюс, 2003. – 380 с.

2. І.В.Гребеннік, М.Ю.Вишняк, В.Г.Іванов, З.А.Імангулова, Н.І.Калита Елементи системного проектування (за редакцією І.В.Гребенніка): Навч. посібник. – Харків: ХНУРЕ, 2016. – 322 с.

3. Kim J. Genome Data Analysis. 1th ed. Berlin : Springer, 2019. 75 p.

4. Документація до Spring. URL: https://docs.spring.io (дата звернення: 02.03.2024).

5. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. 1th ed. London : O′Reilly, 2020. 345 p.

6. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 624 p.

7. Willard H. F., Ginsburg G. S. Genomic and Personalized Medicine. Elsevier Science & Technology Books, 2012. 1350 p.

8. Pevsner J. Bioinformatics and Functional Genomics. Wiley & Sons, Incorporated, John, 2015. 1168 p.

9. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update / E. Afgan et al. Nucleic Acids Research. 2022. URL: https://doi.org/10.1093/nar/gkac247 (дата звернення: 28.05.2024).

10. Ensembl 2022 / F. Cunningham et al. Nucleic Acids Research. 2021. Vol. 50, no. D1. P. D988–D995. URL: https://doi.org/10.1093/nar/gkab1049 (дата звернення: 25.05.2024).

11. Variant Review with the Integrative Genomics Viewer / J. T. Robinson et al. Cancer Research. 2017. Vol. 77, no. 21. P. e31-e34. URL: https://doi.org/10.1158/0008-5472.can-17-0337 (дата звернення: 19.05.2024).

12. Krogh J. W. MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds. Apress L. P., 2020.

13. Aggarwal C. C., Reddy C. K. Data Clustering: Algorithms and Applications. Taylor & Francis Group, 2018. 652 p.

14. Cophenetic correlation analysis as a strategy to select phylogenetically informative proteins: an example from the fungal kingdom / E. E. Kuramae et al. BMC Evolutionary Biology. 2007. Vol. 7, no. 1. P. 134. URL: https://doi.org/10.1186/1471-2148-7-134 (дата звернення: 20.05.2024).

15. Strimpel J., Overson J. Developing Web Components: UI from jQuery to Polymer. O'Reilly Media, 2015. 252 p.

16. Meeks E. D3.js in Action. Manning Publications, 2015. 352 p.

17. Учасники проектів Вікімедіа. IDEF0 – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/IDEF0 (дата звернення: 26.05.2024).

18. UML Use Case Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 26.05.2024).

19. Bilyk V. Data Flow Diagrams (DFD) Explained. ArtofBA. URL: https://www.artofba.com/uk/post/data-flow-diagrams-dfd-explained-1 (дата звернення: 26.05.2024).

20. 11.2.4. Інші діаграми bpWin. StudFiles. URL: https://studfile.net/preview/9445425/page:40/ (дата звернення: 15.06.2024).

21. Учасники проектів Вікімедіа. Діаграма послідовності – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_послідовності (дата звернення: 20.06.2024).

22. Maxym Z. Діаграма послідовності (Sequence Diagrams). Maxym Zosym. URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 20.06.2024).

23. Учасники проектів Вікімедіа. WebSocket – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/WebSocket (дата звернення: 20.06.2024).

24. Лахтін В. В. Розробка веб-застосунку для аналізу та візуалізації біологічних даних: 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., – Харків: ХНУРЕ. 2024. С. 105-106.