НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Платформа для аналізу та проєктування топологічних організацій розподілених систем

Виконав студент IV курсу, групи ІТ-02
(шифр групи)

Лазанчук Костянтин Валерійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доц. к.т.н. Волокита А. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц. к.т.н Катін П. Ю. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Лазанчук Костянтин Валерійович
(прізвище, ім'я, по батькові)

1. Тема проєкту Платформа для аналізу та проєктування топологічних організацій розподілених систем

керівник проєкту Волокита Артем Миколайович доц. к.т.н. Волокита А. М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27»травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
4) Технологічний розділ: керівництво користувача.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Схема бази даних

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	14.03.2024	
3	Постановка та формалізація задачі	17.03.2024	
4	Розробка інформаційного забезпечення	22.03.2024	
5	Алгоритмізація задачі	25.03.2024	
6	Обґрунтування вибору використаних технічних засобів	30.03.2024	
7	Розробка програмного забезпечення	15.04.2024	
8	Налагодження програми	10.05.2024	
9	Виконання графічних документів	27.05.2024	
10	Оформлення пояснювальної записки	30.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Костянтин ЛАЗАНЧУК

(підпис) (ініціали, прізвище)

Керівник _____ Артем ВОЛОКИТА

(підпис) (ініціали, прізвище)

Анотація

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 47 таблиць, 39 рисунків та 12 джерел – загалом 93 сторінки.

Дипломний проєкт присвячений розробці платформи для аналізу та проєктування топологічних організацій розподілених систем.

Метою цієї розробки є оптимізація робочих процесів та створення рішення для роботи з топологічними графами та їх аналізу шляхом створення платформи для аналізу та проєктування топологічних структур розподілених систем.

Об'єкт дослідження: платформи для аналізу топологічних структур.

Предмет дослідження: створення платформи з розширеним функціоналом у порівнянні з аналогами.

У розділі передпроєктного обстеження було розглянуто та проаналізовано обрану предметну область. Визначено ключові концепції, методології, алгоритмічні рішення, архітектуру та допоміжні засоби, а також оптимізовано бізнес-процеси за допомогою BPMN моделей. На завершення сформульовано мету розробки та завдання для її досягнення.

Розділ розробки вимог до програмного забезпечення присвячений аналізу потреб користувачів та формуванню вимог. Описано бізнес-процеси, створено діаграму варіантів використання, виділено функціональні та нефункціональні вимоги. Як результат, сформовано технічне завдання на розробку програмного забезпечення.

У розділі конструювання та розробки програмного забезпечення проведено моделювання та проектування. Розглянуто архітектуру програмного забезпечення, вибір засобів розробки, аналіз структури бази даних та питання захисту даних.

Розділ аналізу якості та тестування програмного забезпечення присвячений методам аналізу якості коду. Проведено ручне тестування, розроблено тест-кейси та контрольний приклад, що підтвердили успішне функціонування програмного забезпечення.

У розділі розгортання та супроводу програмного забезпечення описано процес впровадження продукту та його розгортання. Визначено вимоги до

підтримки, що забезпечують цілодобову доступність системи. КЛЮЧОВІ СЛОВА:ПЛАТФОРМА, ТОПОЛОГІЯ, VISUAL STUDIO, ASP.NET, NEO4J, ANGULAR.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 47 tables, 39 figures and 12 sources – in total 93 pages.

The diploma project is dedicated to the development of a platform for analyzing and designing topological organizations of distributed systems.

The purpose of this development is to optimize workflows and create a solution for working with and analyzing topological graphs by creating a platform for analyzing and designing topological structures of distributed systems.

Object of research: platforms for analyzing topological structures.

Subject of research: creation of a platform with extended functionality compared to analogs.

In the pre-project survey section, the selected subject area was reviewed and analyzed. We identified key concepts, methodologies, algorithmic solutions, architecture, and supporting tools, and optimized business processes using BPMN models. Finally, the development goal and tasks to achieve it are formulated.

The section on software requirements development is devoted to analyzing user needs and formulating requirements. The business processes are described, a use case diagram is created, and functional and non-functional requirements are identified. As a result, a technical specification for software development is formed.

In the section on software design and development, modeling and design were performed. The software architecture, selection of development tools, analysis of the database structure, and data protection issues are considered.

The section on software quality analysis and testing is devoted to methods of analyzing code quality. Manual testing was performed, test cases and a control example were developed, which confirmed the successful functioning of the software.

The section on software deployment and maintenance describes the process of product implementation and deployment. The support requirements are defined to ensure round-the-clock availability of the system.

KEYWORDS: PLATFORM, TOPOLOGY, VISUAL STUDIO, ASP.NET, NEO4J, ANGULAR.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0201.045440.01.91	Технічне завдання	13	
3	А4	КПІ.ІТ-0201.045440.02.81	Пояснювальна записка	93	
4	А4	КПІ.ІТ-0201.045440.03.12	Текст програми	41	
5	А4	КПІ.ІТ-0201.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-0201.045440.05.34	Керівництво користувача	19	
7	А3	КПІ.ІТ-0201.045440.06.99	Графічний матеріал	3	

КПІ.ІТ-0213.045440.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Лазанчук К.В.		
Перевір.		Волокита А.М.		
Н.контр.		Ліщук К.І.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІТ-02

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Вебзастосунок для проведення консультацій з психологом

Технічне завдання

КПІ.ІТ-0213.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Артем ВОЛОКИТА

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Костянтин ЛАЗАНЧУК

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Платформа для аналізу та проєктування топологічних організацій розподілених систем.

Галузь застосування:

Наведене технічне завдання поширюється на розробку платформи для аналізу та проєктування топологічних організацій розподілених систем “Diplom Project”, котра буде використовуватися для роботи з топологічними графами. Ця платформа буде корисною для дослідників, інженерів та розробників, які працюють з розподіленими системами, мережами, та іншими областями, де потрібний аналіз та проектування складних топологій.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки “ Diplom Project ” є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для дослідників, інженерів та розробників, які працюють з розподіленими системами, мережами, та іншими областями, де потрібний аналіз та проектування складних топологій.

Метою розробки платформи “Diplom Project” є створення інструменту для аналізу та проектування топологічних організацій розподілених систем. Ця платформа забезпечує користувачів можливістю легко та ефективно моделювати, аналізувати та оптимізувати структури складних мереж та систем. Основні завдання платформи включають

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– відображення даних у зрозумілому користувачу вигляді (Рисунок 4.1);

– базова сторінка (перша частина) (Рисунок 4.1);

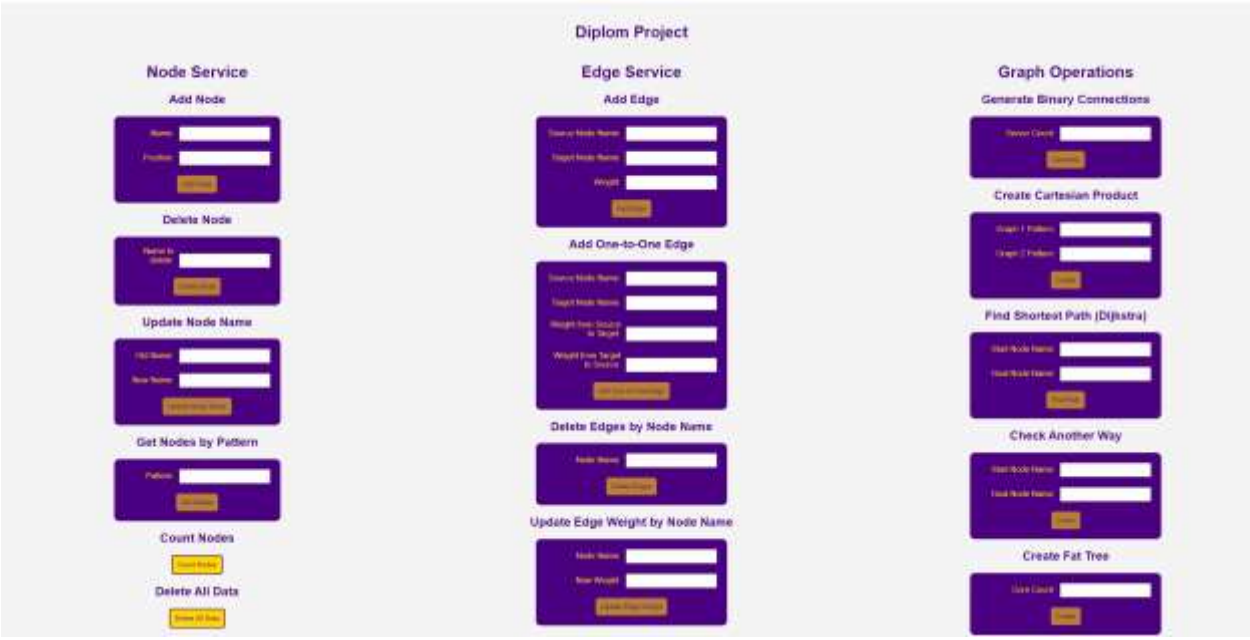


Рисунок 4.1 – Базова реєстрації (перша частина)

– базова сторінка (друга частина) (Рисунок 4.1);

Рисунок 4.2 – Базова реєстрації (друга частина)

– відображення топологічних графів (Рисунок 4.3);

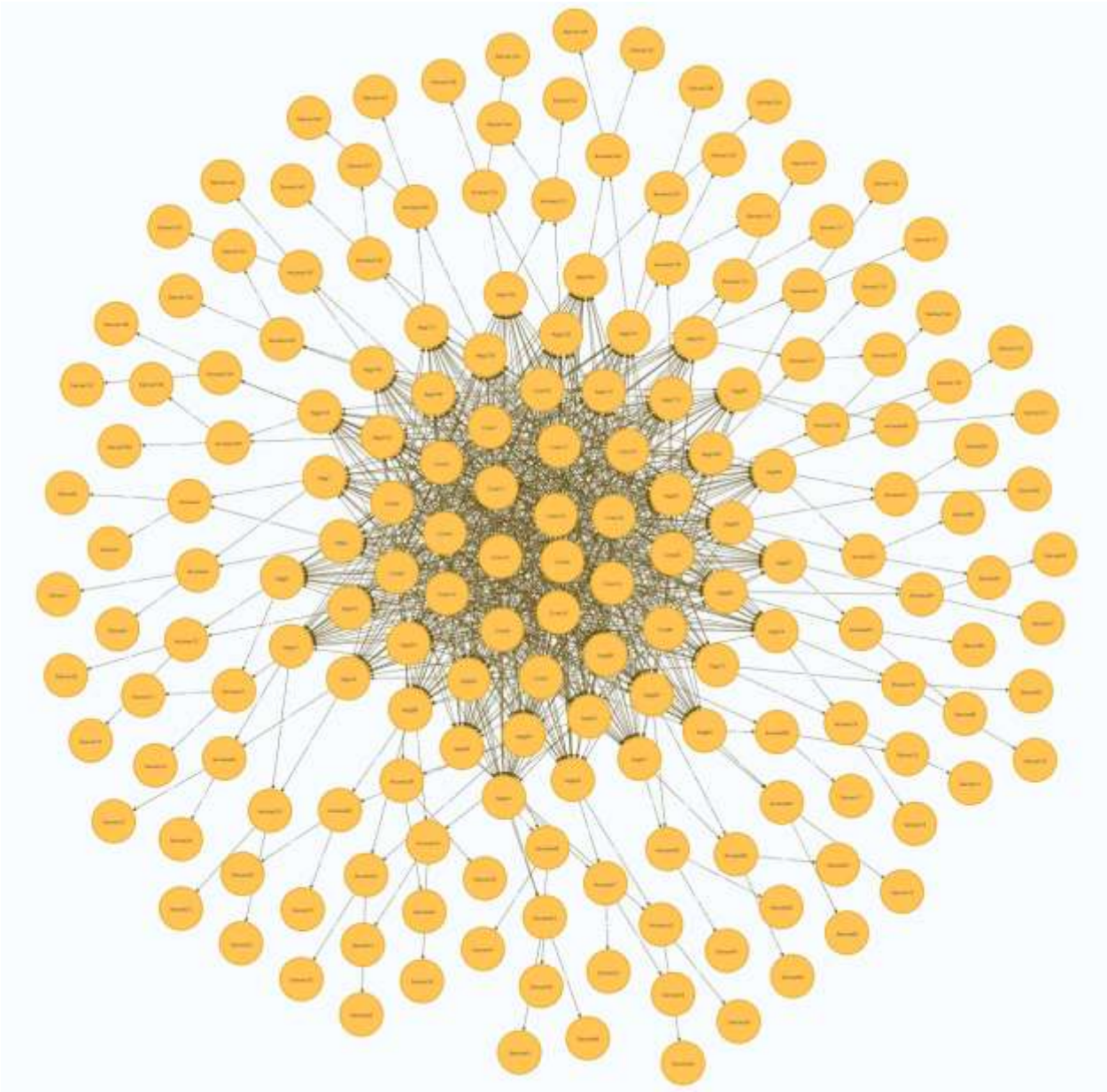


Рисунок 4.3 – Відображення топологічного графа

4.1.2 Для користувача:

– можливість створювати вузол;
– можливість редагувати вузол;
– можливість видаляти вузол;
– можливість створювати ребро;
– можливість редагувати ребро;
– можливість видаляти ребро;
– можливість згенерувати граф з певною кількістю вузлів;

– можливість видаляти графові системи;

– можливість створювати топологічний граф «Жирне дерево»;

– можливість створювати топологічний граф через кодове перетворення;

– можливість об’єднати топологічні графи вузол на вузол;

– можливість об’єднати топологічні графи використовуючи декартовий добуток топологій;

– можливість знайти шляху від одного до другого вузла використовуючи алгоритм A*;

– можливість знайти шляху від одного до другого вузла використовуючи алгоритм Дейкстри;

– можливість прорахувати альтернативний шлях при заблокування найкоротшого;

– можливість прорахувати мережу на відмово стійкість.

4.1.3 Для адміністратора системи (якщо він передбачений):

Не передбачено.

4.1.4 Додаткові вимоги:

Не висуваються.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Рекомендовані апаратні вимоги:

- процесор: Inter Core 2duo або AMD Athlon з мінімальною частотою від 1 ГГц;
- оперативна пам’ять: Мінімум 4 ГБ для стабільної роботи;
- місце на диску: Не менше ніж 1 ГБ вільного місця для тимчасових файлів та кешу;
- мережеве підключення: Інтернет з’єднання зі швидкістю не менше 5 Мбіт/с для стабільної комунікації з застосунком.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix. Програмне забезпечення повинно працювати на браузерах Google Chrome (version 72.0.3626+), Safari (version 15.6.1 +), Firefox (version 102.3esr+).

4.5.1 Вимоги до вхідних даних

Відсутні.

4.5.2 Вимоги до вихідних даних

Відсутні.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# завдяки середовищу ASP.NET Core. Для ведення бази даних використати Neo4j, для розробки клієнтської частини було використано мову програмування TypeScript з використання фреймворка Angular.

4.5.4 Вимоги до середовища розробки

Розробку за допомогою середовища Microsoft Visual Studio Community 2022 для серверної частини, для бази даних було використано Neo4j та для розробки клієнтської частини було розроблено за допомогою Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторія на платформі GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– архітектура програмного забезпечення;
– схема структурна класів.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проекту

на тему: Платформа для аналізу та проєктування топологічних організацій розподілених систем

КПІ.ІТ-0213.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
IT	–	Інформаційні технології
ER	–	Entity-Relation diagram
БД	–	База даних
LAN	–	Local Area Network
SAN	–	Мережі зберігання даних
MANET	–	Мобільні Ad Hoc мережі
ITS	–	Інтелектуальні транспортні системи
HPC	–	Високопродуктивні обчислювальні системи
VIRL	–	Cisco Virtual Internet Routing Lab
MVC	–	Модель-вид-контролер
IoT	–	Інтернету речей
JVM	–	Java Virtual Machine
SAN	–	Мережі зберігання даних
SPA	–	Single Page Application
HTTPS	–	Hypertext Transfer Protocol Secure
TLS	–	Transport Layer Security
GUI	–	Графічний інтерфейс користувача
RDF	–	Resource Description Framework

ВСТУП

У сучасному світі інформаційних технологій розподілені системи стають невід'ємною частиною інфраструктури більшості програм, організацій та підприємств. Зростаюча складність і масштаби таких систем обумовлені збільшенням обсягів даних та кількості взаємодій між вузлами мереж, що створює нові виклики для їхнього управління. Ця робота є актуальною, оскільки потребує нових підходів для полегшення проектування, аналізу та оптимізації розподілених систем для забезпечення їхньої надійності, продуктивності та масштабованості.

Світові тенденції у цій сфері свідчать про зростаючу потребу у нових інструментах для проектування та аналізу розподілених систем. Провідні IT-компанії активно займаються розробкою нових технологій і методів, спрямованих на підвищення ефективності цих систем.

Сучасний стан програм, які спеціалізуються в розподілених систем характеризується наявністю численних рішень для управління топологіями мереж, однак вони часто стикаються з різними проблемами, як масштабованість, перевірка робото стійкості та виявлення слабких місць. Провідні вчені та фахівці в галузі інформатики розробляють інноваційні підходи для подолання цих викликів, акцентуючи увагу на інтеграції різних джерел даних та реального часу аналізу.

Розробка платформи для аналізу та проектування топологічних організацій розподілених систем спрямована на створення гнучкого та масштабованого інструменту, який дозволить IT-фахівцям і інженерам моделювати, аналізувати та оптимізувати складні архітектури. Однією з ключових функцій такої платформи є можливість швидкого створення, модернізації системи та перевірка на роботу, що забезпечує актуальну інформацію про стан мережі та системи в реальному часі.

Можливі сфери застосування розробленої платформи охоплюють широкий спектр галузей – від телекомунікацій та фінансових послуг до виробництва та побудови спеціальної мережі. Ця платформа не лише спрощує

процес створення складних системам, але й сприяє підвищеному виявленню слабких місць, що в свою чергу підвищує ефективність бізнесу та задоволеність користувачів.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Данна робота визначається реалізацією платформи для аналізу та проектування топологічних організацій розподілених систем.

Розподілена система — це набір незалежних комп'ютерів, що представляється їх користувачам єдиною об'єднаною системою.

У цьому визначенні обмовляються два моменти. Перший відноситься до апаратури: всі машини автономні. Другий стосується програмного забезпечення: користувачі думають, що мають справу з єдиною системою. Важливо обидва моменти. Можливо, замість того щоб розглядати визначення, розумніше буде зосередитися на важливих характеристиках розподілених систем. Перша з таких характеристик полягає в тому, що від користувачів приховані відмінності між комп'ютерами і способи зв'язку між ними. Те ж саме відноситься і до зовнішньої організації розподілених систем. Іншою важливою характеристикою розподілених систем є спосіб, за допомогою якого користувачі і додатки одноманітно працюють в розподілених системах, незалежно від того, де і коли відбувається їх взаємодія.

Розподілені системи повинні також відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика є прямим наслідком наявності незалежних комп'ютерів, але в той же час не указує, яким чином ці комп'ютери насправді об'єднуються в єдину систему. Розподілені системи зазвичай існують постійно, проте деякі їх частини можуть тимчасово виходити з ладу. Користувачі і додатки не повинні повідомлятися про те, що ці частини замінені або полагоджені або що додані нові частини для підтримки додаткових користувачів або додатків. [1]

Для більш чіткого розумінні будемо представляти топологію у виді графів. Вони приводять її через кілька основних причин, які роблять модель зручніше та ефективними для представлення аналізу.

– природне відображення, як мережи пов’язані. Топологія розподіленої системи або мережі природно відображається як набір вузлів (вузли) і зв'язків (ребра) між ними. Вузли можуть представляти об'єкти, а ребра - взаємодії або зв'язки між ними;

– гнучкість і масштабованість. Через різноманітність структур, моделі можна представити у різному вигляді, включаючи деревоподібні або циклічні;

– ефективність алгоритмів. На сьогоднішній день існує велика кількість ефективних алгоритмів для пошуку шляхів, каутеризації, виявлення спільнот та інших задач аналізу графів. Це робить графові моделі потужним інструментом для топологічного аналізу;

– оптимізація. Багато є рішень для оптимізації, як наприклад алгоритм Дейкстри, який дає можливість знайти найкоротший шлях;

– візуалізація. Графи легко розуміти та продемонструвати, завдяки цьому їх структура можна легко зрозуміти, що дасть можливість працювати з ними більш ефективно.

Далі буде приведемо основні види графів, які існують та де їх використовуються у сфері топологічних систем.

Топологія зірка – це єдина топологія з явно виділеним центром, до якого підключаються всі інші частини. Обмін інформацією йде винятково через центральний елемент, на який лягає більше навантаження, а також він відповідає за розподілення. Про рівноправність всіх частин у цьому випадку говорити не доводиться. Звичайно, на центральний елемент покладають усі функції по керуванню обміном. Ніякі конфлікти в системі з топологією зірка в принципі неможливі, тому що керування повністю централізоване. [2]

На рисунку 1.1 продемонстровано підвиди топологія зірка

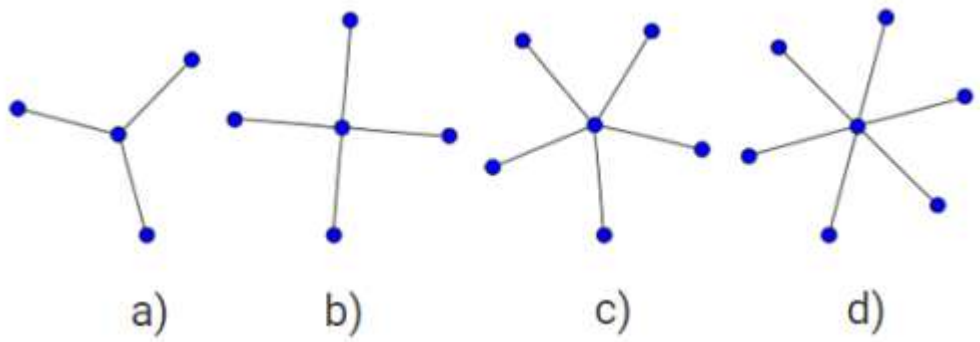


Рисунок 1.1 - Підвиди зірки : a – клешня b – із чотирма ребрами, c – із п'ятьма ребрами , d – із шістьома ребрами

Таку систему використовують в таких варіантах як: локальна мережа (LAN), телекомунікаційна мережа, центр обробки даних, система управління будівлями та розумні будинки

1.1.2 Топологія Дерево

Топологія дерево також відома як ієрархічна топологія, є розширенням топології "Зірка", де центральні вузли зіркових мереж з'єднані з одним головним центральним вузлом, формуючи структуру, що нагадує дерево. Це дозволяє організовувати великі мережі з чіткою ієрархією та забезпечує легкість розширення мережі.

На рисунку 1.2 продемонстровано топологію дерево

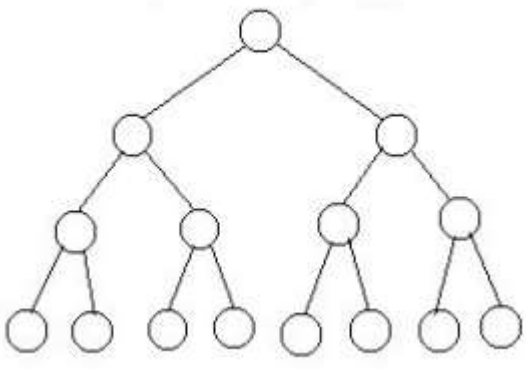


Рисунок 1.2 - Топологія дерево

Топологія "Дерево" використовується в широкомасштабних мережевих інсталяціях, таких як корпоративні мережі, університетські та інші.

1.1.3 Топологія Кільце

Дана топологія передбачає, що всі вузли в мережі з'єднані в замкнуте кільце, де кожен вузол має безпосереднє з'єднання з двома іншими вузлами: одним зліва і одним справа. В цій топології дані передаються в одному напрямку — від вузла до вузла — до тих пір, поки не досягнуть призначеного вузла.

На рисунку 1.3 продемонстровано топологію кільце

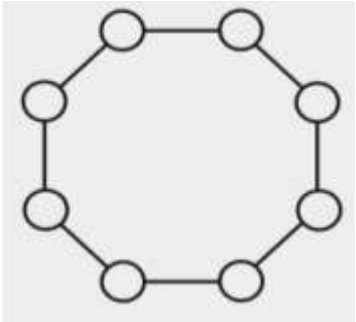


Рисунок 1.3 - Топологія кільце

Дану систему можна використовувати для сітьових мережі, особливо в оптичних кільцях. Дуже добре підходить для Мережі зберігання даних (SAN) та для промислових мереж.

1.1.4 Топологія Решітка

Топологія решітки, яка також відома як меш-топологія, є способом організації комп'ютерних мереж, де кожен вузол мережі безпосередньо з'єднаний з одним або кількома іншими вузлами. Це створює щільну мережу з'єднань, де є множинні шляхи даних між вузлами. У повній меш-топології кожен вузол з'єднаний з кожним іншим вузлом у мережі, тоді як у частковій меш-топології вузли з'єднані тільки з декількома іншими вузлами, формуючи решітку з'єднань.

На рисунку 1.4 продемонстровано топологію решітка

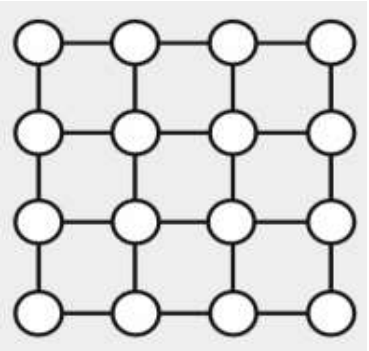


Рисунок 1.4 - Топологія решітка

Така топологія дуже добре підходить для комп’ютерних кластерів, мережі на основі мультипроцесорних систем та мобільна Ad Hoc мережа (MANET).

1.1.5 Топологія Пов’язана

Пов'язана топологія або інколи її називають «Повна» топологія є типом мережевої архітектури, де кожен вузол мережі безпосередньо з'єднаний з усіма іншими вузлами. Данна топологія забезпечує найвищий рівень надійності серед усіх мережевих топологій, оскільки втрата будь-якого одного з'єднання не впливає на загальну працездатність мережі.

На рисунку 1.5 продемонстровано топологію кільце

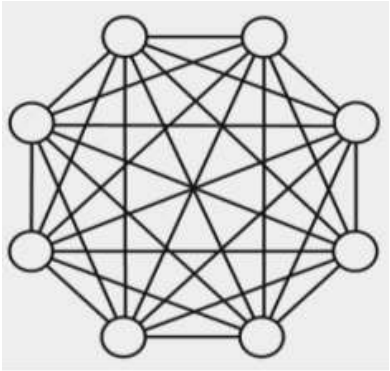


Рисунок 1.5 - Пов’язана топологія

Таку систему використовують для суперкомп’ютерів для паралельних обчислень, корпоративні мережи, яким потрібно з’єднати офіси чи філіали та фінансові установи, як банки або біржи.

1.1.6 Топологія Змішана

Змішана або гібридна топологія є композицією різних типів топологій мережі, що використовуються разом у межах однієї мережевої інфраструктури. Цей підхід дозволяє об'єднувати елементи топологій зірка, кільце, решітка, дерево та інших у єдину, більш функціональну та ефективну мережеву структуру. Гібридні топології дозволяють використовувати переваги кожної окремої топології, мінімізуючи при цьому їх недоліки та брати до увагу загальну архітектуру самої топології.

На рисунку 1.6 продемонстровано змішана топологія

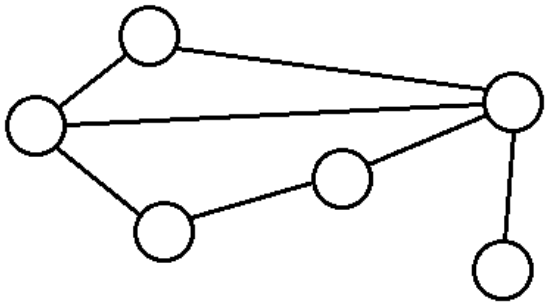


Рисунок 1.6 - Змішана топологія

Таку топологію можна використати в Інтелектуальні транспортні системи (ITS), для освітньої установи та промислових систем автоматизацій.

1.1.7 Топологія Шина

Топологія шина є одним з класичних методів організації комп'ютерних мереж. У такій топології всі комп'ютери (вузли мережі) підключені до одного спільного комунікаційного кабелю, який називається шиною. Дані, відправлені одним з вузлів, розповсюджуються по всьому кабелю і стають доступними всім іншим вузлам на мережі. Кожен вузол перевіряє адресу відправника та отримувача у пакеті даних і обробляє дані, якщо вони адресовані йому.

На рисунку 1.1.7 продемонстровано змішана топологія

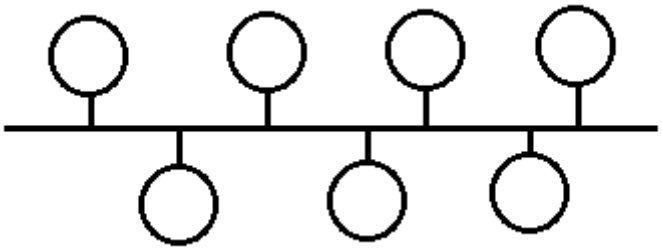


Рисунок 1.7 - Топологія шина

Така топологія підходить для інтегрованих систем та вбудованих пристроїв, для малих локальних мереж та для мережі кабельного телебачення.

1.1.8 Топологія Жирне дерево

Топологія "жирне дерево" , також відома як "товсте дерево", є варіантом ієрархічної мережевої архітектури, яка широко використовується в великомасштабних обчислювальних середовищах, таких як дата-центри та обчислювальні кластери. Основна ідея цієї топології полягає в створенні мережевої структури, де кожен рівень ієрархії має більшу пропускну здатність, ніж попередній, що дозволяє ефективно розподіляти трафік і зменшувати затори в мережі.

На рисунку 1.1.8 продемонстровано змішана топологія

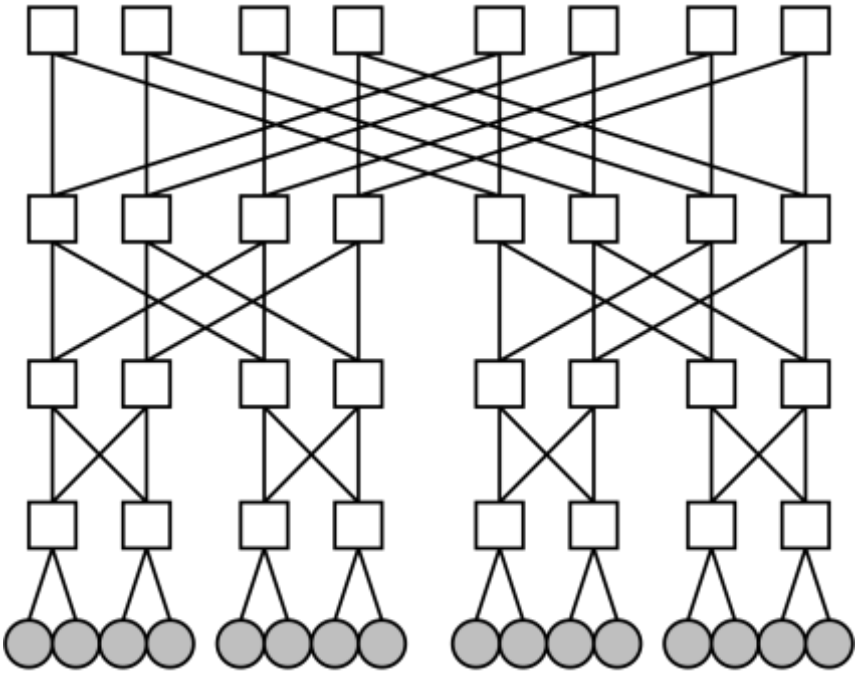


Рисунок 1.8 - Топологія жирне дерево

Дану топологію використовують для великої обробки даних, для високопродуктивних обчислювальних систем (HPC), хмарні обчислення та великі корпоративні мережі.

Дані моделі систем, які було продемонстровано використовуються майже у всіх цифрових галузях і потребують, як підтримку так і розвиток. Сучасні компанії під час створення нового високотехнологічного продукту прораховуються на перед, яка буде топологія, які будуть позитивні сторони при обиранні її так і негативні, як наприклад високе фінансування, слабке місце у певному місці або складність масштабування.

На сьогоднішній день потреба у новому інструменті, який буде створювати, аналізувати топології є дуже висока. Ось декілька конкретних потреб, які можна виділити:

– інструмент в якому можна створити новий об’єкт для роботи в топології. Для створення власної платформи потрібно, щоб було можливість створювати вершини графа та їх об’єднувати, це включає також збереження цих даних у базу даних;

– аналіз системи. Після створення певної системи, потрібно провести аналіз за певними критеріями, як: кількість вершин та їх переходи, пошук центрального вузла, пошук найкоротшого шляху і так даліє;

– перевірка на робото здатність. Після створення системи та її перевірки на базові компоненти, потрібно зробити тестування на її роботу. Це може бути стрес тест або просто перевірка чи є зв’язок між певними не суміжними вершинами;

– безпека мережи. Перевірка та дослідження вразливості, які можуть існувати в певній системі.

Потрібно підкреслити, що Топологічні системи, як і будь-які інші складні технологічні рішення, мають свої проблеми та недоліки. Ось деякі з основних:

– складність управління. До цього можна підставити саме координацію вузлів. У результаті правлінням багатьма вузлами в розподіленій системі може бути складною та високо навантаженою роботою. Це вимагає ефективної координації та синхронізації;

– проблема масштабованості. Деякі рішення у обиранні певної топології можуть стикатися з обмеженнями продуктивності при масштабуванні або не виявленні певної проблеми на початку проектування системи. Наприклад, централізовані системи можуть ставати вузьким місцем у міру зростання навантаження;

– складність моделювання та аналізу. Через велику кількість різних даних обробка та аналіз їх може призвести до потреби у великої кількості розрахунків;

– високі витрати. До цього пункту можна зазначити, як високу вартість впровадження так і підтримка самої системи. Бо після створення самої системи її підтримка та оновлення може коштувати велику кількість ресурсів, як фінансових так і людських.

Розглянувши виклики, які є в даній галузі в рамках дипломного проекту було обрану шлях розробки своєї спеціальної платформи для аналізу та проектування топологічних організацій розподілених систем, яка буде давати можливість створювати свої системи та робити аналіз на початку її проектуванні, щоб виявити її недоліки. Окремо даний проект буде мати можливість розширюватися завдяки додаванню у нього функціоналу такого, як нові алгоритми перевірки роботи системи, більш глибоке створення мережі, додавання нового функціоналу і тому інше.

1.2 Аналіз існуючих рішень

Бля більш чіткого розуміння недоліків у сфері топологічних систем та їх створення, потрібно переглянути у даній області сумчасті продукти та їх технічні рішення, що допоможуть у реалізації платформи для аналізу та

проектування топологічних організацій розподілених систем. Далі будуть розглядатися готові програмні рішення та програмні продукти.

1.2.1 Аналіз відомих програмних продуктів

На сьогоднішній день існує велика кількість різних систем, які дають можливість створити власну топологічну систему. Однак, розгляд існуючих систем показує, що у всіх них є певні негативні сторони, які не дають можливість використати, щоб задовольнити усі потреби.

Для більш чіткого порівняння буде декілька категорій, через які будуть робитися порівняння між програмами, до них входить: можливість масштабування та його складність, аналіз мережи, перевірка на робото здатність, візуалізація мережи, простота у використанні, гнучке налаштування мережи, потреба у встановленні продукту.

1.2.1.1 Cisco Packet Tracer

Cisco Packet Tracer - це кросплатформний інструмент візуального моделювання, розроблений компанією Cisco Systems, який дозволяє користувачам створювати мережеві топології та імітувати сучасні комп'ютерні мережі.[4]

Сильна перевага даного продукту, що для нього є дуже багато різної інформації у відкритому доступі. Це дає можливість дуже якісно розібратися у даному продукті та створити хорошу мережу. Проте хочу підкреслити, що даний продукт може працювати дуже повільно при великій та навантаженій мережі, що приводить до погіршення роботи з даним продуктом.

Приклад роботи даної програми на рисунку 1.9

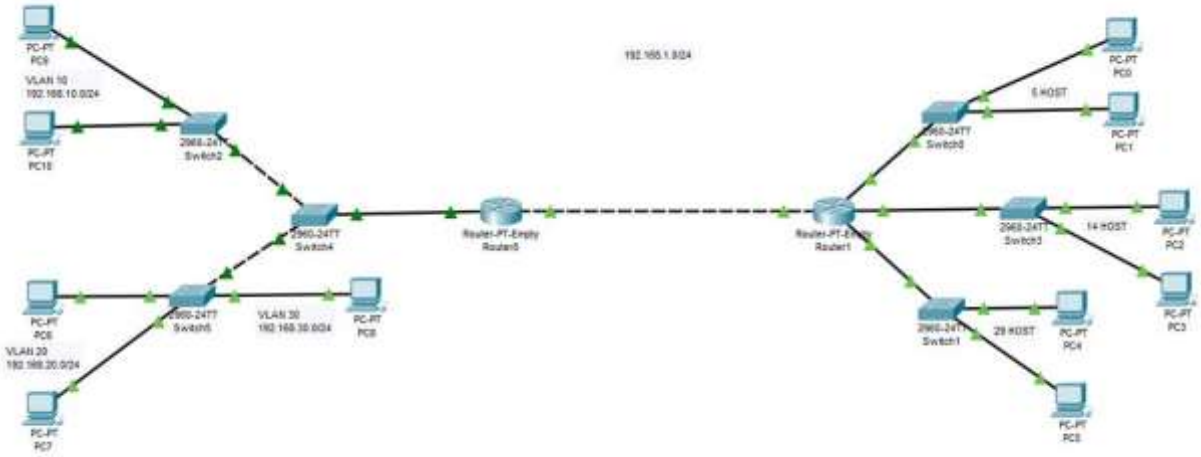


Рисунок 1.9 - Приклад системи у Cisco Packet Tracer

1.2.1.2 Boson NetSim

Boson NetSim – це спеціальна платформа для моделювання мереж, яка дає можливість користувачам створювати, налаштовувати та тестувати свої віртуальні мережі. Вона розроблена для того, щоб допомогти користувачам вивчити та практикувати навички роботи з мережами.

Переваги даного застосунку те що дана платформа дуже гнучка, дає можливість побудувати любу мережу та протестувати її, що у результаті дає можливість точно побудувати модель з мінімальною кількістю слабких місць. Також хочу зазначити, що для даної платформи також існує велика кількість різних навчальної інформації, що дає можливість глибоко вникнути у саму розробку без додаткових проблем у розумінні даного продукту. З приємних бонусів, є інтегрованість з іншими продуктами від компанії Boson, що може допомогти користувачам навчитись або покращити свої навички у роботі з мережами.

Приклад роботи даного продукту на рисунку 1.10

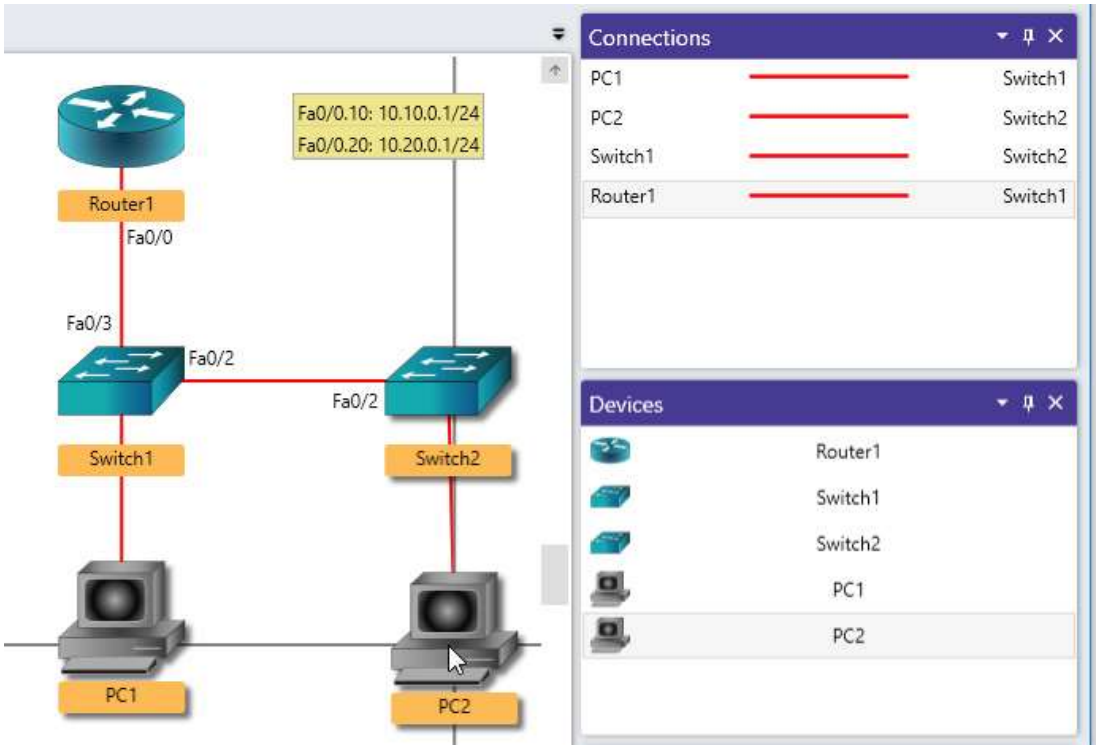


Рисунок 1.10 - Приклад системи у Boson NetSim

1.2.1.3 Cisco VIRL

Cisco Virtual Internet Routing Lab (VIRL) - це програмний інструмент для створення та запуску мережевих симуляцій без використання фізичного обладнання.

Дана програма дозволяє користувачам створювати, налаштовувати та тестувати складні мережеві топології без необхідності в дорогому фізичному обладнанні. VIRL використовується навчальними закладами, фахівцями з мережевих технологій та дослідниками для проектування, навчання, тестування та дослідження мереж.

Переваги даного продукту є що він може дуже легко масштабуватися, він дає можливість підтримувати мережу будь-якого розміру та оновлювати або доповнювати її.

Приклад системи побудованої даним продуктом на рисунку 1.11

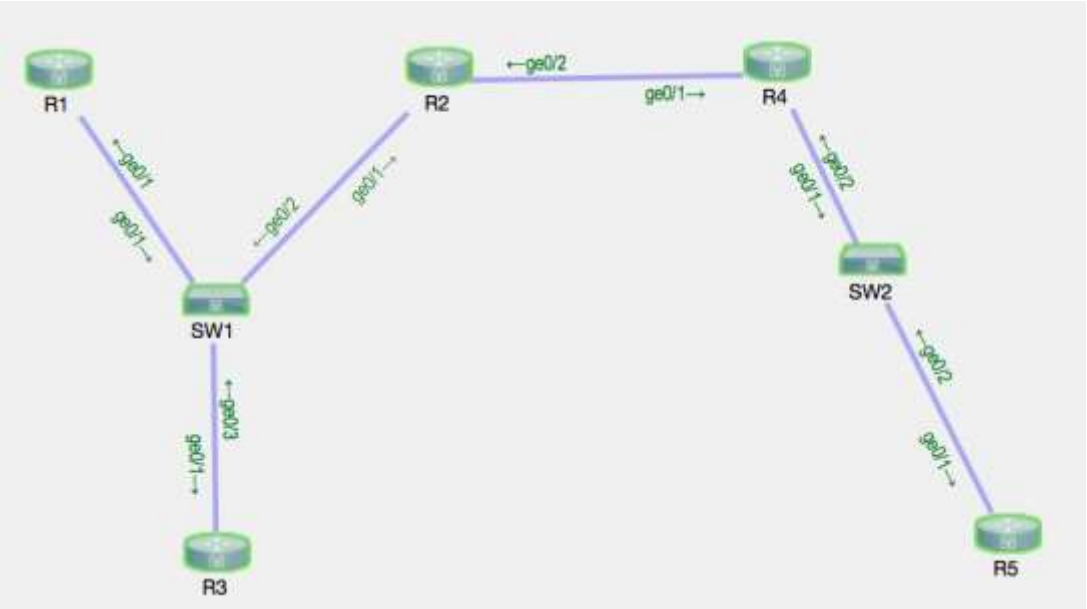


Рисунок 1.11 - Приклад системи у Cisco VIRL

Для порівняння дипломної роботи з іншими аналогами, які були описані, можна скористатись таблицею 1.1

Таблиця 1.1 – Порівняння з іншими продуктами

Функціонал	Diplom Project	Cisco Packet Tracer	Boson NetSim	Cisco VIRL	Пояснення
Можливість масштабування та складність	+	-	+	+	Можливість оновлювати мережу без додаткового сильного навантаження на неї

Продовження таблиці 1.1

Функціонал	Diplom Project	Cisco Packet Tracer	Boson NetSim	Cisco VIRL	Пояснення
Аналіз мережі	+	+	+	+	Є функціонал, який аналізує мережу, може показати слабкі сторони створеної мережи
Перевірка на працездатність	+	-	+	+	Є Можливість зробити симуляцію мережі та перевірити чи правильно вона працює
Візуалізація мережі	+	+	+	+	Є візуалізація мережі
Простота у використанні	+	+	-	-	Простота у використанні

Продовження таблиці 1.1

Функціонал	Diplom Project	Cisco Packet Tracer	Boson NetSim	Cisco VIRL	Пояснення
Гнучке налаштування мережі	+	-	+	+	Дає можливість добре налаштувати мережу
Не потреба у встановленні продукту	+	-	-	+	Для роботи з програмою, не потребується її інсталяція на власний комп’ютер
Ціна	+	+	-	-	Перед початком роботи з даним продуктом чи потрібно заплатити перед ним

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

В рамках даної роботи передбачається використання певних алгоритмів для виявлення певних параметрів. У світовій практиці в основному для роботи

з топології використовують алгоритми, які потурбовані на графах, тому будемо використовувати на їх основі.

Алгоритм Дейкстри це одним з алгоритмів, які дозволяють розв'язати задачу пошуку найкоротшого шляху у зваженому графі. Даний алгоритм дозволяє знайти найкоротший шлях від однієї фіксованої вершини графа до всіх інших його вершин, проте є значно оптимальнішим у порівнянні з іншими.[5]

Алгоритм Беллмана – Форда це ітеративний алгоритм, що не лише дозволяє знайти найкоротший шлях від однієї вершини до іншої у зваженому графі, але й знайти найкоротші шляхи від однієї вершини зваженого графа до усіх інших його вершин. Цей алгоритм є одним з найпростіших з алгоритмічної точки зору, проте й одним з найповільніших. Його перевагою над іншими алгоритмами є те, що він може відшукати найкоротший шлях у графі, що містить ребра від'ємної ваги, якщо граф не має циклів від'ємної ваги. Крім цього, алгоритм Беллмана – Форда дозволяє досить просто визначити чи містить граф цикл від'ємної ваги. [5]

Алгоритм A* подібний до алгоритму Дейкстри і фактично є його удосконаленням. У ньому також для визначення порядку обходу вершин використовується черга з пріоритетом. Проте, на відміну від алгоритму Дейкстри, у якому пріоритет кожної неопрацьованої вершини враховує лише поточну довжину шляху від стартової точки, пріоритет вершин у алгоритмі A* використовує додатково допоміжну функцію (евристику), аби скеровувати напрям пошуку. [5]

Алгоритм Монте-Карло. Даний метод працює шляхом запуску симуляцій або епізодів, коли агент взаємодіє з середовищем, поки не досягне термінального стану. Наприкінці кожного епізоду алгоритм переглядає відвідані стани та отримані винагороди, щоб обчислити так звану «віддачу» — сукупну винагороду, починаючи з певного стану до кінця епізоду. Оцінка політики за методом Монте-Карло неодноразово моделює епізоди, відстежуючи загальні винагороди за кожним станом, а потім обчислюючи

середнє значення. Ці середні значення дають оцінку вартості держави відповідно до політики, якої дотримуються.

Агрегуючи результати за багатьма епізодами, метод наближається до справжнього значення кожного стану при дотриманні політики. Ці значення корисні, оскільки вони допомагають нам зрозуміти, які стани є більш цінними, і таким чином спрямовують агента до прийняття кращих рішень у майбутньому. З часом, коли агент дізнається цінність різних станів, він може вдосконалити свою політику, віддаючи перевагу діям, які призводять до вищих винагород. [6]

Алгоритм пошук у глибину. Це алгоритм для обходу графа, у якому застосовується стратегія йти, на скільки це можливо, вглиб графа. Проте на відміну від дерев пошук в глибину має одну особливість – при обході дерев, на відміну від графів, не може трапитися ситуація при якій ми спробуємо відвідати вершину у якій ми вже були. Дійсно, якщо розглядати дерево як граф, то входу кожної вершини не може бути більшим за 1 (0 для кореня і 1 для всіх його вузлів). Останнє означає, що в кожну вершину дерева можна потрапили єдиним шляхом. Іншою суттєвою відмінністю алгоритму пошуку в глибину для графа від аналогічного алгоритму для дерев є те, що ми змушені визначати початкову вершину з якої стартує пошук. Для дерев пошук завжди стартував з кореня дерева. [5]

Алгоритм пошук у ширину. Даний алгоритм Аналогічний до пошуку в глибину. У ньому застосовується стратегія послідовного перегляду окремих рівнів графа, починаючи з заданого вузла.

Зробивши аналіз усіх даних алгоритмів для перевірки дерев на шлях, нам буде підходити два алгоритми, а саме алгоритм A* та алгоритм Дейкстри. Дані алгоритме дуже добре працюють з графами і можуть допомогти знайти найкоротший шлях від одного графа до іншого. Обидва можна використати для виявлення некоректності або слабких місць у системі. Також нам потрібен алгоритм для перевірки робото здатності. Для цього дуже добре підходить

алгоритм Монте-Карло, він допоможе перевірити нашу систему, як вона працює при різних умовах.

1.2.2.1 Аналіз відомих архітектурних рішень

Для створення даного проекту потрібно обрати вірну архітектуру. Це дасть можливість зрозуміти, як краще будувати проект, та дати можливість його масштабуванні у майбутньому.

Архітектура на основі мікро сервісів передбачає розбиття програмного коду на незалежні шматки сервісів, кожен з яких виконує конкретну функцію або дію і взаємодіє з іншими сервісами через визначені інтерфейси.

Монолітна архітектура створює цілісний застосунок, де всі його компоненти працюють у єдиному середовищі і тісно взаємодіють між собою. Такий підхід має свої переваги, зокрема, спрощену розробку та розгортання: всі частини системи знаходяться в одному пакеті, що полегшує управління залежностями та забезпечує координацію між різними модулями.

Трьох рівнева архітектура передбачає розподіл застосунку на три окремі рівні: презентаційний рівень, логічний рівень (або рівень бізнес-логіки) і рівень даних. Кожен з цих рівнів виконує свої завдання та взаємодіє з іншими рівнями через визначені інтерфейси. Цей підхід має ряд переваг, серед яких чітке розділення обов'язків та спрощення розробки та підтримки.

Модель-вид-контролер (MVC) є архітектурною схемою, що передбачає розділення застосунку на три окремі компоненти: модель, вид і контролер. Кожен компонент виконує специфічну роль і взаємодіє з іншими через визначені інтерфейси. Цей підхід має ряд переваг, серед яких чітке розділення обов'язків та покращена організація коду. Модель відповідає за управління даними і бізнес-логікою.

Після чіткого аналізу, було обрано архітектуру для дипломного проекту, а саме трьох рівнева архітектура, вона допоможе добре розділили алгоритми, дає можливість додавати окремий функціонал через модульність та дасть легко змінити певні частини проекту, які будуть потребувати у модернізації або змінні. На додачу це дасть чітке розподілення різних частин проекту, що

полегшить розробку та дасть можливість повторне використання певного шматку коду та дасть легко масштабувати окремі рівні незалежно один від одного.

1.3 Опис бізнес-процесів

Для опису певних бізнес процесів, потрібно використати BPMN модель. Схема бізнес-процесу «Створення вузла» на рисунку 1.12

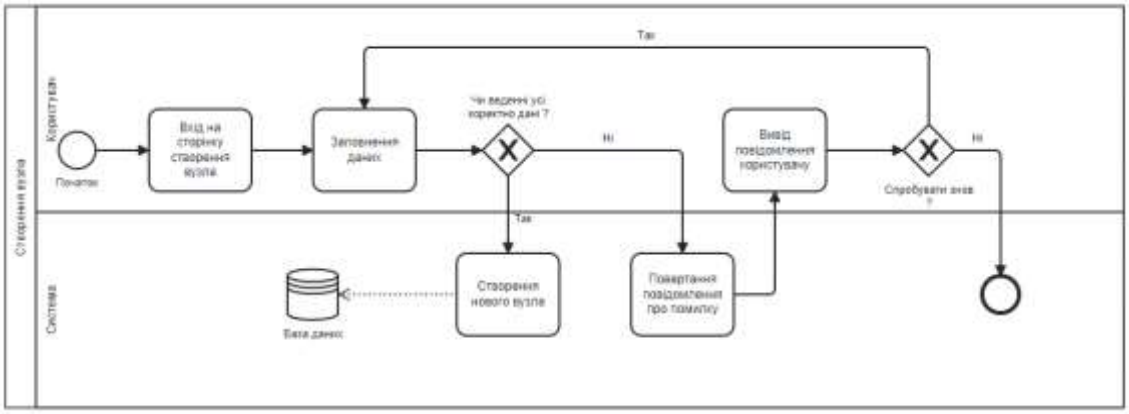


Рисунок 1.12 - Схема бізнес-процесу «Створення вузла»

Опис роботи:

– користувач заходить на сторінку, де можна створити вузол для графа;

– користувач заповнює потрібні поля для реалізації;

– якщо введені дані будуть не коректні то система напише йому об помилки і запропонує спробувати знов;

– якщо користувач не хоче пробувати знов, його повертають до початкової сторінки, процес закінчується;

– якщо користувач хоче повторити, він залишається на сторінці де потрібно вести дані для створення нового вузла;

– чкщо користувач вів правильні усі дані, то проходять оновлення у базі даних, а саме створюється новий вузол. Процес закінчується.

Схема бізнес-процесу «Додавання ребра між вузлами» на рисунку 1.13

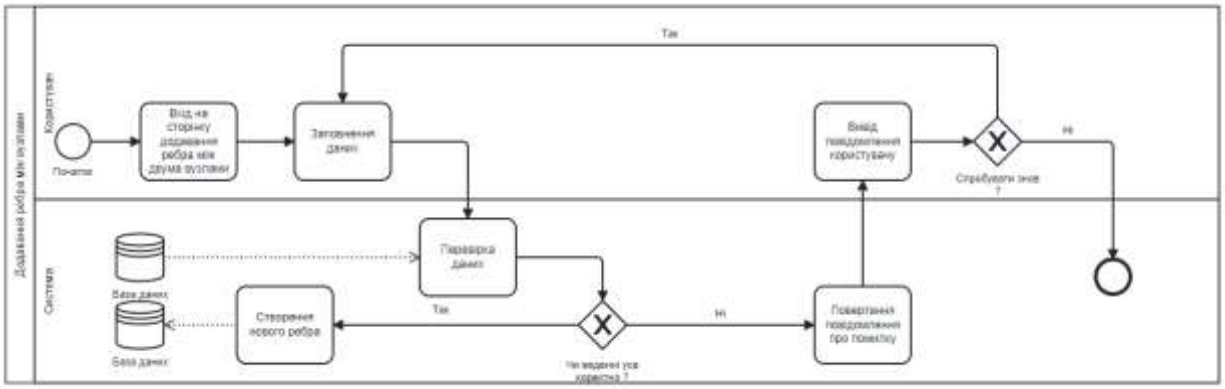


Рисунок 1.13 - Схема бізнес-процесу «Додавання ребра між вузлами»

Опис роботи:

– користувач заходить на сторінку, де можна створити ребро між двома графами;

– користувач заповнює потрібні поля для роботи;

– система шукає два вузла, які потрібно зв’язати;

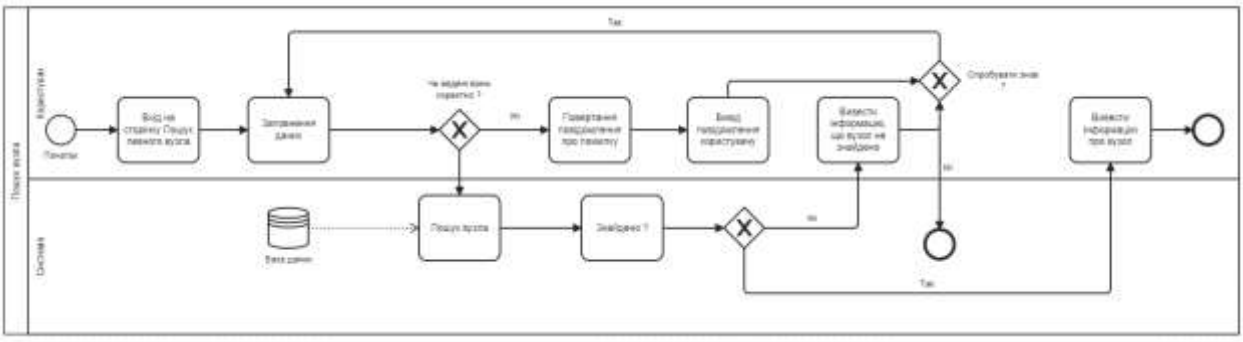
– якщо система не знаходить два вузла то вона повертає повідомлення про помилку;

– якщо користувач не хоче пробувати знов, його повертають до початкової сторінки, процес закінчується;

– якщо користувач хоче повторити, він залишається на сторінці де потрібно вести дані для створення нового вузла;

– якщо система знаходить два вузла то вона оновлює базу даних та створює ребро між ними та закінчує процес.

Схема бізнес-процесу «Пошук вузла» на рисунку 1.6



Рисунку 1.14 - Схема бізнес-процесу «Пошук вузла»

Опис роботи:

– користувач заходить на сторінку, де можна створити знайти певний вузол у системі графів;

– користувач заповнює потрібні поля для роботи;

– якщо введені дані будуть не коректні то система напише йому об помилки і запропонує спробувати знов;

– якщо користувач не хоче пробувати знов, його повертають до початкової сторінки, процес закінчується;

– якщо користувач вів правильні усі дані то переходимо до пошуку їх у базі даних;

– якщо у базі даних не було знайдено даного вузла, то система виводить повідомлення про не існуючого вузла;

– якщо користувач не хоче пробувати знов, його повертають до початкової сторінки, процес закінчується;

– якщо користувач хоче повторити, він залишається на сторінці де потрібно вести дані для створення нового вузла;

– якщо у базі даних існує такий вузол, система виводить повну інформацію про нього.

Схема бізнес-процесу «Аналіз системи» на рисунку 1.15

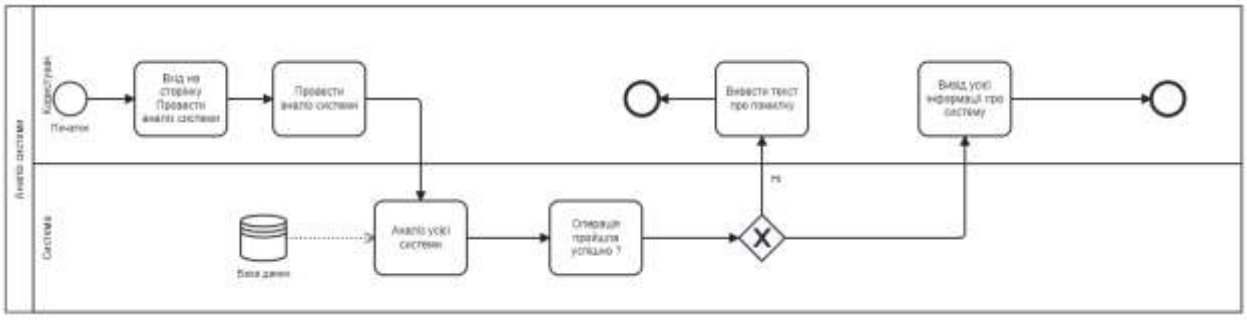


Рисунок 1.15 - Схема бізнес-процесу «Аналіз системи»

Опис роботи:

– користувач заходить на сторінку, де можна зробити аналіз системи;

– користувач починає перевірку;

– система бере дані із бази даних про кожний вузол та ребро;

– якщо перевірка пройшла успішно, користувачу виводять повну інформацію про систему;

– якщо при обробці система зіткнулося з помилкою, вона напишу користувачу про помилку та закінчить виконання роботи.

Схема бізнес-процесу «Імпорт даних» на рисунку 1.16

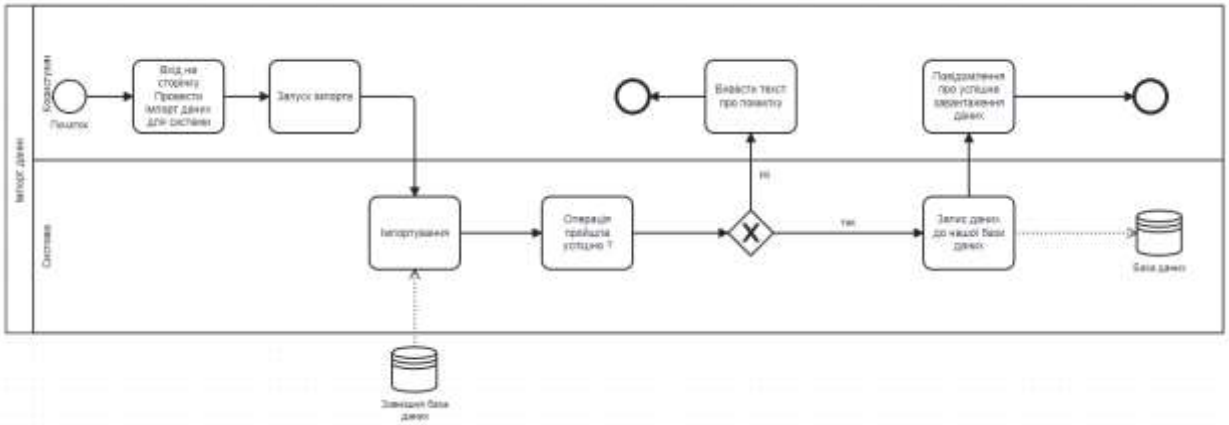


Рисунок 1.16 - Схема бізнес-процесу «Імпорт даних»

Опис роботи:

– користувач заходить на сторінку, де можна імпортувати системи;

– користувач починає процес імпортування;

– система робить запит через API до зовнішнього середовища та получає нову систему;

– якщо десь сталася помилка, система повідомляє користувачу та завершує роботу;

– якщо система вдалося вона зібрати дані з зовнішнього середовища то потім вона їх записує у свою базу даних, потім повідомляє про успішну роботу та завершує її.

Схема бізнес-процесу «Експорт даних» на рисунку 1.17

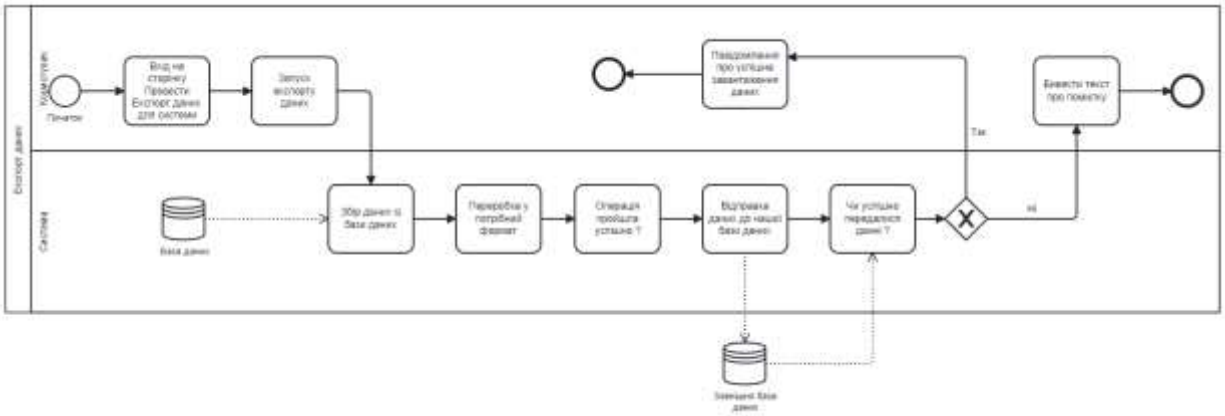


Рисунок 1.17 - Схема бізнес-процесу «Експорт даних»

Опис роботи:

– користувач заходить на сторінку, де можна експортувати системи до зовнішньої бази даних;

– користувач починає процес експортування даних;

– збирає їх із внутрішньої бази даних;

– обробка даних;

– відправляє їх до зовнішньої бази даних;

– получає відповідь про успішне експортування чи з помилкою;

– якщо вивело помилку, то система повідомляє про це користувачу та закінчує роботу;

– якщо вивело про успішну відправку, то система також повідомить про це користувачу та закінчить роботу.

1.4 Постановка задачі

Головною метою є створення спеціальної платформи, яка буде спеціалізованим веб-додатком, який буде надавати можливість користувачам будувати свою власну топологічну систему та перевіряти її на робото здатність. Така платформа дуже сильно полегшить розробку системи для різних користувачів, а саме від компанії до звичайних користувачів, які хочу спробувати побудувати не велику систему. Також завдяки створеній системі багато галузей, які побудовані на цьому, дати можливість приймати рішення набагато швидше.

Основні завданні, які потрібно виконати в рамках даної роботи:

– розробка системи в якій буде базовий функціонал робити з вузлами та ребрами;

– реалізація алгоритмів для перевірки системи, пошуку шляху від одного вузла до іншого;

– розробка інтерфейси для користувача, який буде взаємодіяти з даною платформою, та буде зручним у використанні;

– реалізація перевірки системи на робото здатність, а саме перевірка системи на певній симуляції, яка може протестувати створену систему користувачем.

Висновки до розділу

У цьому розділі даного дипломного проекту було проведено детальний аналіз в якому було розписано загальні поняття, що пов’язано з темою розробки даного продукту. До нього входить загальна інформація про топологію та спрощений варіант, як працювати з ними через графи. Було описано кожний відомий із них та було наведено приклади, де така кожна система може використовуватися у нашому житті.

Було зроблено порівняння декількох інших продуктів, було розписано їх, та виявлено сильні сторони кожного з них. На основі цих даних, було створено спеціальну таблицю, яка демонструю сильні та слабкі сторони кожного продукту в певних базових параметрів, так як функціонал, складність роботи з ними, можливість масштабувати систему.

Наступна частина було аналіз алгоритмів, було розписано базове їх призначення, для чого вони використовуються. Завдяки цьому було обрано декілька з них.

На додачу був розгляд відомих архітектурних рішень, було описано, їх сильні та слабкі сторони, як потрібно реалізовувати програмний продукт, при дотриманні даних архітектурних рішень.

Також було створено базові BPMN моделі для функціональних задач, яка потім у плані реалізувати їх. Із базових BPMN моделей увійшло: створення вузла, додавання ребра між вузлами, пошук вузла, аналіз системи, імпорт даних та експорт даних це одні із найголовніших моделей, які потрібно реалізувати в даного проекті, щоб він міг функціонувати та бути сходим на платформу з роботи топологічних графів, але у планах в функціональних ролях розписати більш детально з усіма можливими нюансами.

На основі всіх отриманих даних було визначено мету розробки, та компоненти, які потрібно реалізувати.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

В даному програмному забезпеченні існує велика кількість функції тому було створено декілька діаграм.

Діаграми будуть продемонстровано на рисунках 2.1 – 2.5

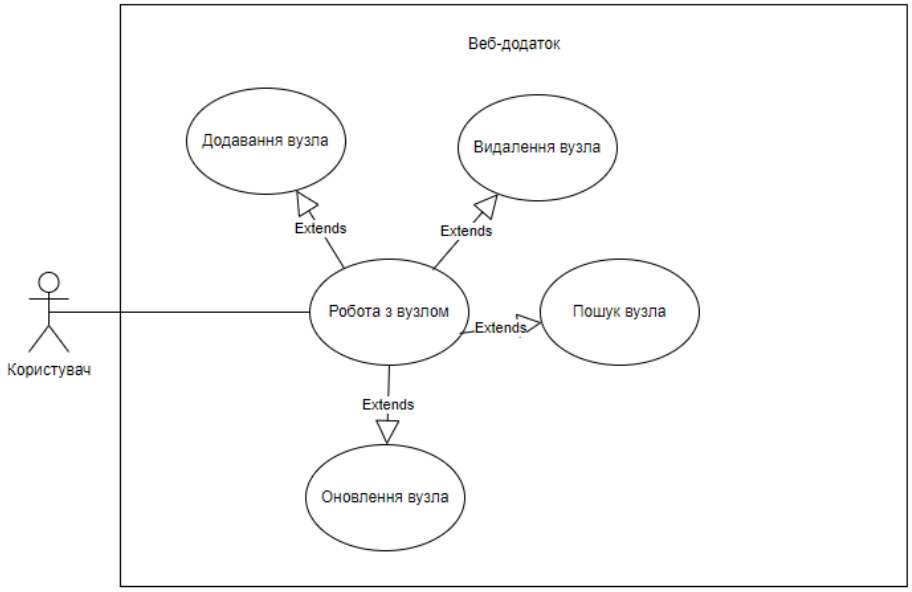


Рисунок 2.1 - Діаграма варіантів роботи з вузлом

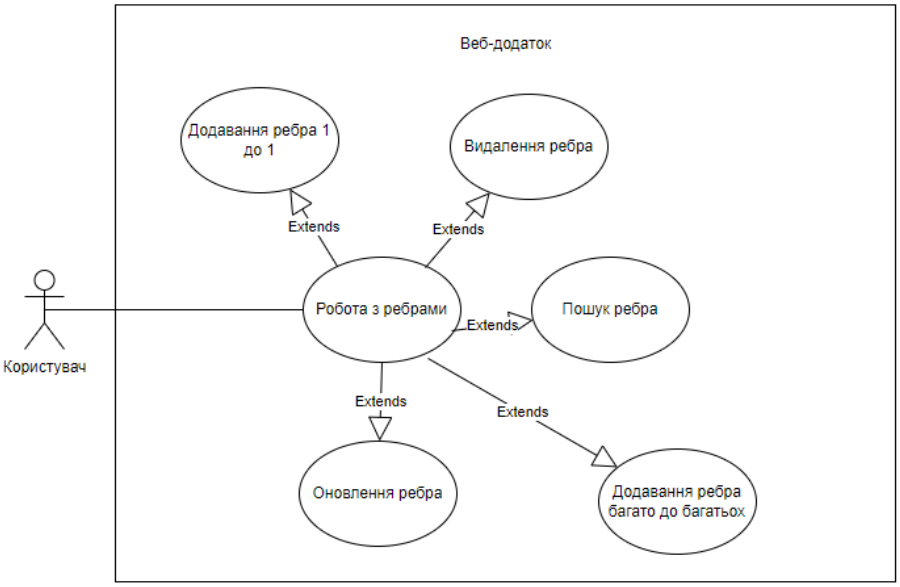


Рисунок 2.2 - Діаграма роботи з ребрами

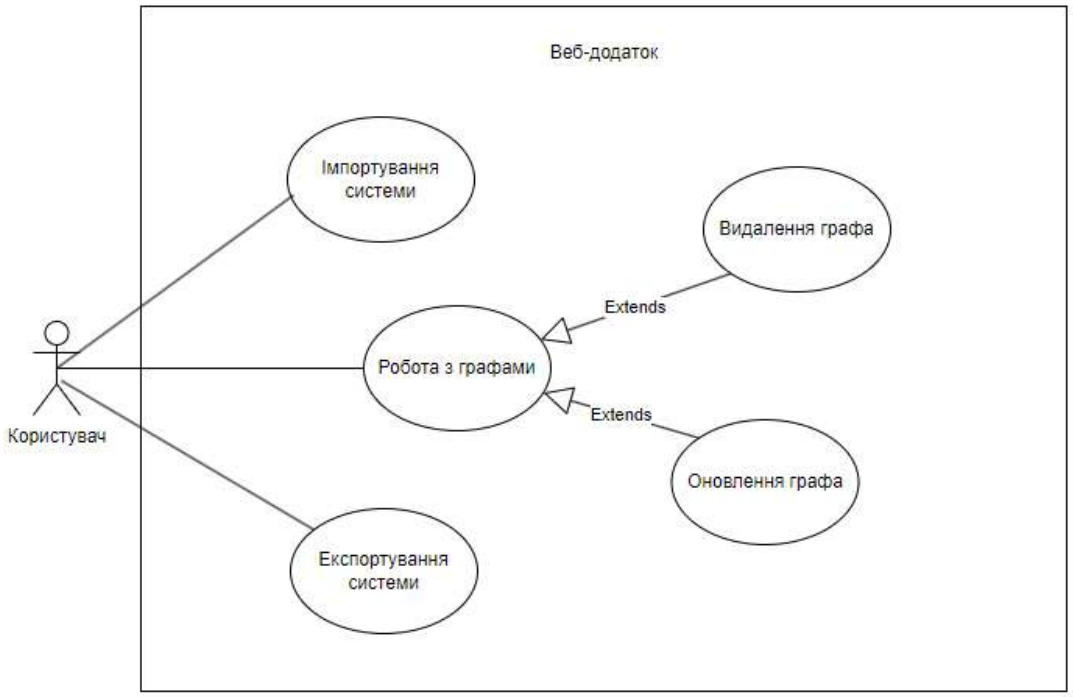


Рисунок 2.3 - Діаграма варіантів роботи з готовою системою графів

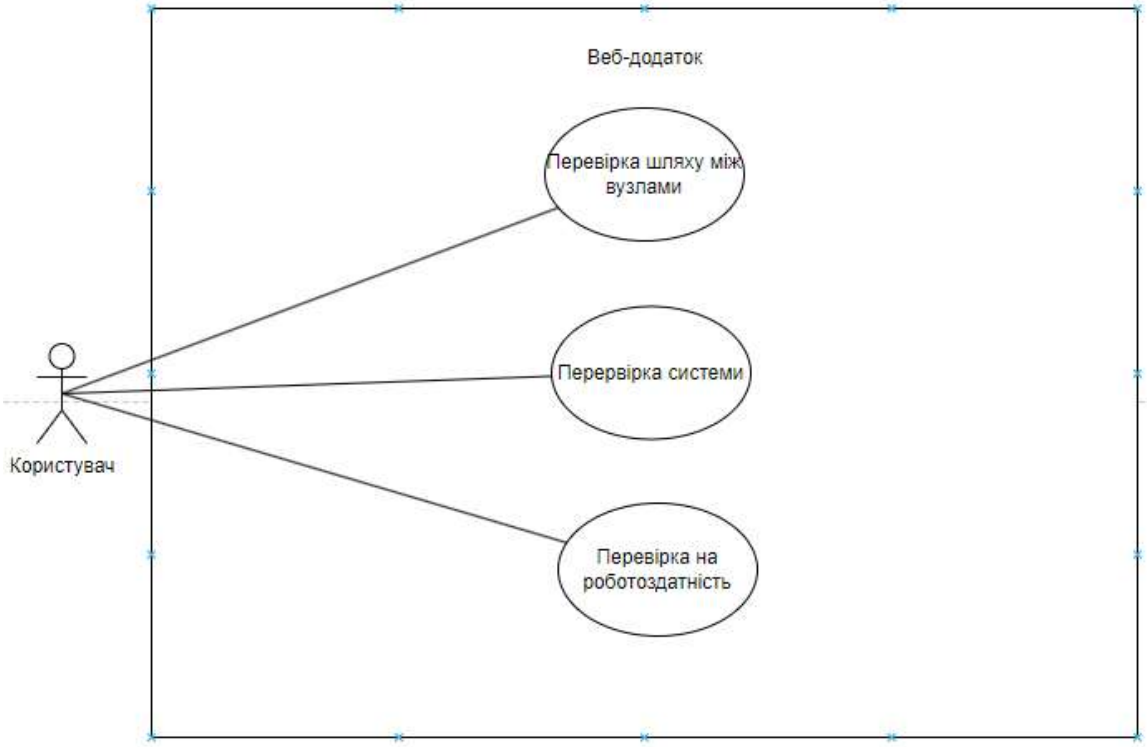


Рисунок 2.4 - Діаграма варіантів роботи аналізу системи

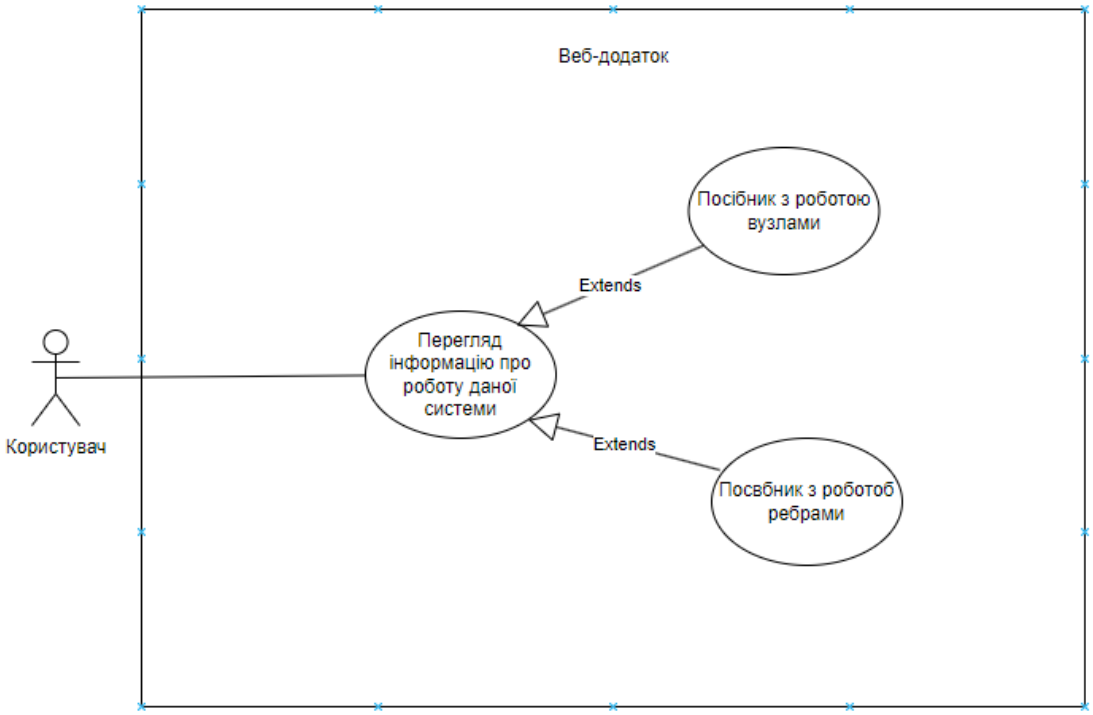


Рисунок 2.5 - Діаграма варіантів посібника у системі

В таблицях 2.1 – 2.20 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Створення вузла
Use case ID	UC-01
Goals	Створення вузла у системі
Actors	Користувач
Trigger	Користувач бажає створити новий вузол у системі
Pre-conditions	-
Flow of Events	Користувач обирає функцію створити вузол, він переходить на іншу сторінку. Там він вводить усі потрібні параметри, а саме: назву вузла, номер його позиції, може створити сам унікальний номер або обрати, щоб створило йому автоматично, у результаті, чого воно записує його до бази даних та пише про успішне створення нового вузла

Продовження таблиці 2.1

Use case name	Створення вузла
Extension	В випадку введення не коректних даних або не заповнення всіх потрібних параметрів, користувач побачить повідомлення про помилку
Post-Condition	Створення нового вузла

Таблиця 2.2 - Варіант використання UC-2

Use case name	Створення ребра (1 до 1)
Use case ID	UC-02
Goals	Створення ребра у системі
Actors	Користувач
Trigger	Користувач бажає створити ребро для об’єднання двох вузлів
Pre-conditions	Користувач вже має, як мінімум два створених вузла
Flow of Events	Користувач обирає функцію створює ребро. Він переходить на сторінку де може вести усі потрібні параметри, а саме, вагу ребра та з якого та до якого вузла буде з’єднувати у результаті, чого воно записує його до бази даних та пише про успішне створення нового ребра та об’єднання двох вузлів
Extension	У випадку введення не коректних даних, побачить повідомлення про некоректне введення даних
Post-Condition	Користувач об’єднає два вузла створивши зв’язок між ними від одного вузла до іншого

Таблиця 2.3 - Варіант використання UC-3

Use case name	Створення ребра багато до багатьох
Use case ID	UC-03
Goals	Створення ребра у системі
Actors	Користувач
Trigger	Користувач бажає створити ребро для об’єднання двох вузлів

Продовження таблиці 2.3

Use case name	Створення ребра багато до багатьох
Pre-conditions	Користувач вже має, як мінімум два створених вузла
Flow of Events	Користувач обирає функцію створює ребро. Переходить на сторінку, де потрібно вести усі потрібні параметри та обирає варіант ввести «багато до багатьох», а саме, вагу ребра та з якого та до якого вузла буде з'єднувати у результаті, чого воно записує його до бази даних та пише про успішне створення нового ребра та об'єднання двох вузлів
Extension	У випадку введення не коректних даних, побачить повідомлення про некоректне введення даних
Post-Condition	Користувач об'єднає два вузла створивши зв'язок між ними або вузла мають зв'язок між собою

Таблиця 2.4 - Варіант використання UC-4

Use case name	Оновити вузол
Use case ID	UC-04
Goals	Оновити параметри вузла
Actors	Користувач
Trigger	Користувач бажає оновити вузол
Pre-conditions	Користувач має створений вузол
Flow of Events	Користувач обирає функцію оновити вузол, спершу він повинен обрати, який вузол він хоче оновити, для цього йому потрібно вести як наприклад назву даного вузла або його унікальний номер змінити назву або позицію вузла, як змінити назву вузла або його параметри
Extension	У випадку введення не коректних даних, побачить повідомлення про некоректне введення даних або вже неможливе зміна ім'я даного вузла через вже існуючий вузол з таким ім'ям
Post-Condition	Користувач оновить вузол, у нього будуть оновлені дані, йому прийде повідомлення про успішне оновлення

Таблиця 2.5 - Варіант використання UC-5

Use case name	Оновлення ребра
Use case ID	UC-05
Goals	Оновити параметри ребра
Actors	Користувач
Trigger	Користувач бажає оновити параметри ребра
Pre-conditions	Користувач має створене ребро
Flow of Events	Користувач обирає функцію оновити ребро, для цього користувач повинен вести дані, а саме унікальний номер, щоб знайти ребро, далі він водить внутрішні параметри, як наприклад вагу ребра або з’єднання між вузлами та оновлює їх
Extension	У випадку введення не коректних даних, побачить повідомлення про не правильне заповнення даних
Post-Condition	Користувач оновить ребро, у нього будуть оновлені дані, йому прийде повідомлення про успішне оновлення

Таблиця 2.6 - Варіант використання UC-6

Use case name	Видалення вузла
Use case ID	UC-06
Goals	Видалення вузла
Actors	Користувач
Trigger	Користувач бажає видалити вузол
Pre-conditions	Користувач має створений вузол
Flow of Events	Користувач обирає функцію видалення вузла, для видалення спочатку потрібно ввести параметри, а саме ім’я або унікальний номер вузла, щоб знайти його далі буде можливість видалити його.
Extension	У випадку не знаходження даного вузла, користувач побачить повідомлення
Post-Condition	Користувач видалить вузол із бази даних

Таблиця 2.7 - Варіант використання UC-7

Use case name	Видалення ребра
Use case ID	UC-07
Goals	Видалення ребра
Actors	Користувач
Trigger	Користувач бажає видалити ребро
Pre-conditions	Користувач має створене ребро
Flow of Events	Користувач обирає функцію видалення ребра, для видалення спочатку потрібно ввести унікальний номер ребра, щоб знайти його, а далі буде можливість видалити його.
Extension	У випадку не знаходження даного ребра, користувач побачить повідомлення про помилку
Post-Condition	Користувач видалить вузол із бази даних

Таблиця 2.8 - Варіант використання UC-8

Use case name	Пошук вузла
Use case ID	UC-08
Goals	Пошук конкретного вузла
Actors	Користувач
Trigger	Користувач бажає знайти конкретний вузол
Pre-conditions	Користувач має вже створений вузол у базі даних
Flow of Events	Користувач обирає функцію пошук вузла, для його знаходження потрібно вести параметри, як наприклад назву вузла або унікальний номер, після чого йому виводить на екран повну інформацію про вузол
Extension	У випадку не знаходження даного вузла, користувач побачить, що данні були введені не правильно.
Post-Condition	Користувач побачить інформацію про вузол

Таблиця 2.9 - Варіант використання UC-9

Use case name	Пошук ребра
Use case ID	UC-09
Goals	Пошук конкретного ребра
Actors	Користувач
Trigger	Користувач бажає знайти конкретний ребро у системі
Pre-conditions	Користувач має вже створене ребро, яке об’єднує вузли у базі даних
Flow of Events	Користувач обирає функцію пошук ребро, він переходить до сторінки де йому потрібно вести унікальний номер.
Extension	У випадку не знаходження даного ребра, користувач побачить, що данні були введені не правильно.
Post-Condition	Користувач побачить інформацію про ребро

Таблиця 2.10 - Варіант використання UC-10

Use case name	Видалення усієї системи
Use case ID	UC-10
Goals	Видалення усієї системи
Actors	Користувач
Trigger	Користувач видалити усю систему із бази даних
Pre-conditions	Користувач має вже створену систему із вузлів та ребр, але вирішив її видалити
Flow of Events	Користувач натискає на кнопку видалити систему, йому відкривається спеціальне вікно, яке попереджає його та просить підтвердити його дії, якщо користувач відмовляється, ніякі дії не виконуються, але якщо користувач погоджується на виконання даних дій то система, яка була у базі даних буде видалена. Після завершення цієї процедури, буде показано повідомлення про успішне видалення
Extension	-
Post-Condition	Користувач видаляє усю систему із бази даних

Таблиця 2.11 - Варіант використання UC-11

Use case name	Генерація випадкової системи
Use case ID	UC-11
Goals	Створення нової системи
Actors	Користувач
Trigger	Користувач створює нову систему
Pre-conditions	-
Flow of Events	Користувач натискає на функцію згенерувати систему. Після натискання він переходить на нову сторінку де буде потрібно вести певні параметри. Після заповнення цих параметрів, він повинен натиснути кнопку «Створити»
Extension	У випадку не коректного заповнення даних, програма повідомить про помилку
Post-Condition	Користувач має щойно створену систему, з якою він може працювати

Таблиця 2.12 - Варіант використання UC-12

Use case name	Огляд системи
Use case ID	UC-12
Goals	Перегляд усіх вузлів та ребр
Actors	Користувач
Trigger	Користувач хоче переглянути повну інформацію про вузли та ребра
Pre-conditions	Користувач має вже створену систему із вузлів та ребр
Flow of Events	Користувач обирає функцію огляд системи, система продемонструє усю інформацію про дану систему, яку він має, а саме усі параметри вузлів та ребра
Extension	-
Post-Condition	Користувачу виводять повну інформацію про систему

Таблиця 2.13 - Варіант використання UC-13

Use case name	Перевірка на об’єднання усіх вузлів
Use case ID	UC-13
Goals	Перегляд чи всі вузли об’єднанні
Actors	Користувач
Trigger	Користувач хоче переглянути чи усі вузли об’єднанні
Pre-conditions	Користувач має вже створену систему із вузлів та ребр
Flow of Events	Користувач обирає функцію перевірка на об’єднання усіх вузлів, програма робить аналіз системи та під час перевірки шукає вузли, які не мають зв’язок з іншими
Extension	У випадку знаходження вузла, який не має з’єднання, програма повинно вивести повну інформацію про даний вузол
Post-Condition	Користувачу повідомляють про результат перевірки

Таблиця 2.14 - Варіант використання UC-14

Use case name	Аналіз системи
Use case ID	UC-14
Goals	Аналіз системи
Actors	Користувач
Trigger	Користувач хоче перевірити систему та получити основну інформацію про неї
Pre-conditions	Користувач має вже створену систему із вузлів та ребр
Flow of Events	Користувач обирає функцію аналіз системи, система перевіряє граф та виводить інформацію, на кількість вузлів та ребр у неї
Extension	-
Post-Condition	Користувачу виводять результат аналізу системи

Таблиця 2.15 - Варіант використання UC-15

Use case name	Перевірка шляху між вузлами
Use case ID	UC-15
Goals	Пошук найкоротшого шляху між двома вузлами
Actors	Користувач
Trigger	Користувач хоче дізнатися, який найкращий шлях від одного вузла до іншого
Pre-conditions	Користувач має вже створену систему із вузлів та ребр
Flow of Events	Користувач обирає функцію перевірка шляху між вузлами, Користувач повинен вести дані про два вузла, які він хоче дізнатися найкоротший шлях. Для цього йому потрібно буде заповнити поле або з унікальними номерами або вести назву вузлів
Extension	При не знаходженні одного із вузлів або не можливість провести шлях, система виведе повідомлення про помилку
Post-Condition	Користувачу виведе інформацію найкоротшого шляху від одного вузла до другого

Таблиця 2.16 - Варіант використання UC-16

Use case name	Перевірка на робото здатність системи
Use case ID	UC-16
Goals	Перевірка системи на робото здатність
Actors	Користувач
Trigger	Користувач хоче перевірити систему на робото здатність
Pre-conditions	Користувач має вже створену систему із вузлів та ребр
Flow of Events	Користувач обирає функцію симуляція, Користувач чекає перевірки програми поки вона про симулює систему на робото здатність.
Extension	-
Post-Condition	Користувачу виведе результат перевірки системи на робото здатність

Таблиця 2.17 - Варіант використання UC-17

Use case name	Підказка по створенню вузла
Use case ID	UC-17
Goals	Продемонструвати вказівки, як створити вузол
Actors	Користувач
Trigger	Користувач не знає, як створити вузол
Pre-conditions	-
Flow of Events	Користувач обирає розділ «підказка», де він переходить на нову сторінку, в якій він натискає на кнопку «Як створити вузол ?»
Extension	-
Post-Condition	Користувачу виводить посібник, як створити вузол, які у нього є параметри і які данні туби можна вводити

Таблиця 2.18 - Варіант використання UC-18

Use case name	Підказка по створенню ребра
Use case ID	UC-18
Goals	Продемонструвати вказівки, як створити ребро у системі
Actors	Користувач
Trigger	Користувач не знає, як створити ребро
Pre-conditions	-
Flow of Events	Користувач обирає розділ «підказка», де він переходить на нову сторінку, в якій він натискає на кнопку «Як створити ребро у системі?»
Extension	-
Post-Condition	Користувачу виводить посібник, як створити ребро, які у нього є параметри і які данні туби можна вводити

Таблиця 2.19 - Варіант використання UC-19

Use case name	Опис призначення програми
Use case ID	UC-19
Goals	Продемонструвати опис, що вміє програма
Actors	Користувач
Trigger	Користувач не знає, що це за програма і він хочу ознайомитися
Pre-conditions	-
Flow of Events	Користувач обирає розділ «підказка», де він переходить на нову сторінку, в якій він натискає на кнопку «Що це за програма ?»
Extension	-
Post-Condition	Користувачу виводить посібник про дану програму, має опис усього функціоналу та, що він робить

Таблиця 2.20 - Варіант використання UC-20

Use case name	Імпорт системи з json формату
Use case ID	UC-20
Goals	Імпорт даних, до яких входить інформація про систему
Actors	Користувач
Trigger	Користувач хоче з іншого додатку імпортувати дані в які вже входить створена мережа
Pre-conditions	-
Flow of Events	Користувач обирає функцію «імпорт дпних» або з іншого сервісу надходить запит про передачу даних. Програма приймає дані у json форматі та обробляє, а саме створює мережу та записує її до бази даних
Extension	При не вірному заповнені json формату, програма виведе повідомлення про помилку
Post-Condition	У програмі оновлюється дані, а саме записується нова мережа з якою можна тепер працювати

Таблиця 2.21 - Варіант використання UC-21

Use case name	Експорт системи з json формату
Use case ID	UC-21
Goals	Експорт даних, до яких входить інформація про систему
Actors	Користувач
Trigger	Користувач хоче з додатку Експортувати дані до іншого застосунку
Pre-conditions	-
Flow of Events	Користувач обирає функцію «Екпорт даних» до іншого сервісу надходить запит про передачу даних. Інша програма повинна приймати дані у json форматі та обробляти їх
Extension	При не вірному заповнені json формату, програма виведе повідомлення про помилку
Post-Condition	У програмі оновлюється дані, а саме записується нова мережа з якою можна тепер працювати

2.1 Аналіз системних вимог

Для успішного функціонування даного додатку, який буде розташований, як веб застосунок. Нам потрібно мати апаратні вимоги, які зможуть виконати увесь функціонал за нормальний час.

У таблиці 2.22 описані системні вимоги.

Таблиця 2.22 – Системні вимоги

Назва вимог	Рекомендовані системні вимоги
Процесор	Inter Core 2duo або AMD Athlon з мінімальною частотою від 1 ГГц
Об’єм оперативної пам’яті	2 ГБ
Швидкість підключення до інтернету	Інтернет з’єднання зі швидкістю не менше 5 Мбіт/с

Продовження таблиці 2.22

Назва вимог	Рекомендовані системні вимоги
Браузер	Google Chrome від версії 90,
Операційна система	Windows
Місце на диску	Як мінімум 1 Гб вільного місця для роботи з ним

2.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.23 наведено загальну модель вимог, а в таблиці 2.24 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.23 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Робота з вузлом	FR-1	Високий	Високий
1.1	Додавання вузла	FR-2	Високий	Середній
1.2	Видалення вузла	FR-3	Високий	Середній
1.3	Оновлення вузла	FR-4	Високий	Середній
1.4	Пошук вузла	FR-5	Високий	Середній
2	Робота з ребром	FR-6	Високий	Високий
2.1	Додавання ребра	FR-7	Високий	Середній
2.2	Видалення ребра	FR-8	Високий	Середній
2.3	Оновлення ребра	FR-9	Високий	Середній
2.4	Пошук ребра	FR-10	Високий	Середній
3	Робота з топологічним графом	FR-11	Високий	Високий
3.1	Видалення топологічного графу	FR-12	Низький	Високий

Продовження таблиці 2.23

№	Назва	ID Вимоги	Пріоритети	Ризики
3.2	Оновлення топологічного графу	FR-13	Середній	Високий
4	Експортування системи з json файлу	FR-14	Середній	Середній
5	Імпортування системи з json файлу	FR-15	Середній	Середній
6	Перевірка системи	FR-16	Високий	Низький
6.1	Перевірка вузлів на можливість знаходження шляху	FR-17	Високий	Високий
6.2	Перевірка на роботоздатність	FR-18	Високий	Високий
6.3	Перевірка на загальну інформацію системи	FR-19	Середній	Середній
7	Можливість переглянути посібник	FR-20	Низький	Низький
7.1	Можливість переглянути посібник з роботою вузлами	FR-21	Середній	Низький
7.2	Можливість переглянути посібник з роботою ребрами	FR-22	Середній	Низький

Таблиця 2.28 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Робота з вузлом. Користувач повинен мати можливість працювати через інтерфейс з вузлами.

Продовження таблиці 2.28

ID вимоги	Назва та опис
FR-2	Додавання вузла. Користувач повинен мати можливість додавати новий вузол до топологічного графа через інтерфейс платформи. Вузол повинен мати унікальний ідентифікатор та набір атрибутів, які можуть бути налаштовані користувачем. Інтерфейс повинен підтримувати валідацію введених даних.
FR-3	Видалення вузла. Користувач повинен мати можливість видаляти існуючий вузол з топологічного графа. Платформа повинна перевіряти, чи є у вузла зв'язки з іншими вузлами, та попереджати користувача про наслідки видалення.
FR-4	Оновлення вузла. Користувач повинен мати можливість оновлювати дані існуючого вузла, змінюючи його атрибути. Інтерфейс повинен надавати зручні засоби для редагування вузла та відображати поточні значення атрибутів.
FR-5	Пошук вузла. Користувач повинен мати можливість здійснювати пошук вузлів за різними критеріями, такими як ім'я, ID або атрибути. Платформа повинна підтримувати швидкий та точний пошук навіть у великих графах.
FR-6	Робота з ребром. Користувач повинен мати можливість працювати через інтерфейс з ребром.

Продовження таблиці 2.28

ID вимоги	Назва та опис
FR-7	Додавання ребра. Користувач повинен мати можливість додавати нове ребро між двома вузлами. Ребро повинно мати унікальний ідентифікатор та набір атрибутів, які можна налаштовувати. Інтерфейс повинен дозволяти вибір вузлів для створення зв'язку.
FR-8	Видалення ребра. Користувач повинен мати можливість видаляти існуюче ребро між двома вузлами. Платформа повинна попереджати користувача про наслідки видалення зв'язку між вузлами.
FR-9	Оновлення ребра. Користувач повинен мати можливість оновлювати дані існуючого ребра, змінюючи його атрибути. Інтерфейс повинен надавати зручні засоби для редагування ребра та відображати поточні значення атрибутів.
FR-10	Пошук ребра. Користувач повинен мати можливість здійснювати пошук ребер за різними критеріями, такими як ID або атрибути. Платформа повинна підтримувати швидкий та точний пошук навіть у великих графах.
FR-11	Робота з топологічним графом. Користувач повинен мати можливість працювати з топологічними графами, мати доступ до інтерфейсу, щоб з ними працювати.

Продовження таблиці 2.28

ID вимоги	Назва та опис
FR-12	Видалення топологічного графа. Користувач повинен мати можливість видаляти весь топологічний граф. Платформа повинна попереджати користувача про наслідки видалення графа та надавати можливість підтвердження дії.
FR-13	Оновлення топологічного графа. Користувач повинен мати можливість оновлювати дані всього топологічного графу. Інтерфейс повинен надавати зручні засоби для редагування графа та відображати поточні значення атрибутів.
FR-14	Експортування системи. Користувач повинен мати можливість експортувати дані системи у форматі JSON. Інтерфейс повинен надавати можливість вибору елементів для експорту та забезпечувати коректне формування файлу.
FR-15	Імпортування системи. Користувач повинен мати можливість імпортувати дані системи з JSON файлу. Інтерфейс повинен забезпечувати коректне зчитування файлу та інтеграцію даних у систему.
FR-16	Перевірка системи. Користувач повинен мати можливість через інтерфейс робити перевірку системи, яка буде давати йому можливість більш чітко розуміти сильні та слабкі сторони створеною їх системи.
FR-17	Перевірка вузлів на можливість знаходження шляху Користувач повинен мати можливість перевіряти, чи існує шлях між двома вузлами у графі. Платформа повинна надавати зручні засоби для вибору вузлів та відображати результати перевірки.

Продовження таблиці 2.28

ID вимоги	Назва та опис
FR-18	Перевірка на працездатність. Користувач повинен мати можливість перевіряти працездатність системи, включаючи виявлення можливих помилок та проблем у графі.
FR-19	Перевірка загальної інформації системи. Користувач повинен мати можливість отримувати загальну інформацію про систему, включаючи статистику по вузлам, ребрам та інші важливі показники.
FR-20	Можливість переглянути інформація про використання даної програми. Користувач має мати можливість через інтерфейс переглянути базу інформацію, щодо роботи базового функціоналу програми.
FR-21	Можливість переглянути посібник з роботою вузлами. Користувач повинен мати можливість переглядати посібник з роботи з вузлами, включаючи інструкції та рекомендації.
FR-22	Можливість переглянути посібник з роботою ребрами. Користувач повинен мати можливість переглядати посібник з роботи з ребрами, включаючи інструкції та рекомендації.

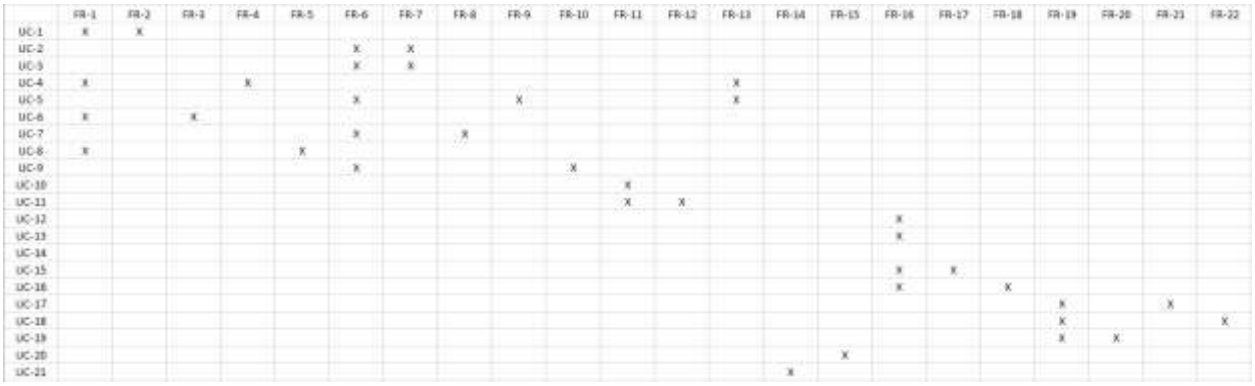


Рисунок 2.6 – Матриця трасування вимог

2.2 Розроблення нефункціональних вимог

Нефункціональні вимоги нам потрібні, що визначити загальні характеристики та обмеження системи, які впливають на її розробку та використання у звичайному режимі роботи. Тому вони є одним ключових для забезпечення якості та ефективності даного застосунку.

Пріоритетними нефункціональними вимогами є:

– можливість у масштабуванні. Система має можливість легко масштабуватися, при додаванні нового функціоналу або інтеграції з іншими додатками або сервісами;

– вивід помилки при некоректної роботи програми. Система повинна реагувати на некоректність виконанні роботи, як наприклад на не правильне введення даних або при виконанні певного функціоналу при не правильній обробці, вона повинно виводити користувачу помилку про це;

– зручний інтерфейс. Інтерфейс користувача має бути зрозумілим та не надлишковим у роботі, головною цілю, щоб користувачі могли легко взаємодіяти з системою та ефективно використовувати її весь функціонал;

– доступний запуск. Програма повинна давати можливість запускатися з усіх відомих браузерів.

Висновки до розділу

В цьому розділі було розроблено use-case діаграму для проекту. В зв’язку, що в дному проекту було вирішено розбити загальну діаграм, в які буде розписано весь функціонал. Діаграми містять в собі інформацію про функціонал проекту.

На основі діаграми було створено таблиці з детальним роз’ясненням функцій, таких як задача, тригер, початкові умови, результат, виняток та інше. Після цього було створено загальну таблицю з функціональними вимогами до проекту, а також таблицю детальним роз’ясненням функціональних вимог до проекту.

Після того, як було розписано функціонал та функціональні вимоги було створено матрицю трасування вимог, дивлячись на яку видно, що всі функціональні вимоги виконуються відповідним функціоналом. Окрім цього було описано приблизні рекомендовані апаратні вимоги користувача а також нефункціональні вимоги.

За результатами даного розділу, було сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для даного веб застосунку, що розробляється, було обрано трьох шарову архітектуру, як основну. Трьох шарова архітектура є однією з найпоширеніших моделей проектування програмного забезпечення. Вона розділяє додаток на три окремі шари: презентаційний шар, логічний шар і шар доступу до даних. Це розділення сприяє покращенню підтримки, масштабованості та керованості додатка.

Презентаційний шар (Presentation Layer)

– функція: Відповідає за взаємодію з користувачем. Це може бути графічний інтерфейс користувача (GUI) у веб-додатку, мобільному додатку або десктопному додатку;

– приклад: У веб-додатку цей шар включає веб-сторінки, стилі CSS та скрипти JavaScript для створення динамічного інтерфейсу користувача.

Логічний шар (Business Logic Layer)

– функція: Відповідає за обробку бізнес-логіки додатка. Логічний шар обробляє запити від презентаційного шару, взаємодіє з шаром доступу до даних та виконує основні бізнес-процеси;

– приклад: Якщо користувач запитує інформацію про продукт, логічний шар обробляє цей запит, виконує бізнес-логіку та взаємодіє з базою даних для отримання необхідної інформації.

Шар доступу до даних (Data Access Layer)

– функція: Цей шар забезпечує доступ до бази даних. Він містить методи для збереження, отримання, оновлення та видалення даних;

– приклад: Шар доступу до даних включає SQL-запити або ORM-конфігурації для взаємодії з базою даних, наприклад, для збереження нової інформації про продукт.

Переваги трьох шарової архітектури

– модульність та підтримка: Розділення додатка на шари дозволяє легше оновлювати, підтримувати та масштабувати кожен окремий компонент. Це забезпечує кращу керованість проектом та можливість незалежного розвитку кожного шару;

– повторне використання коду: Компоненти кожного шару можуть використовуватися повторно в інших додатках або частинах системи, що знижує витрати на розробку та тестування;

– забезпечення безпеки: Розділення бізнес-логіки та доступу до даних дозволяє ефективніше захищати додаток та дані, оскільки кожен шар може мати свої власні механізми безпеки;

– легке тестування: Кожен шар можна тестувати окремо, що спрощує процес відлагодження та забезпечення якості програмного забезпечення.

Приклад, як виглядає трьох шарова архітектура на рисунку 3.1

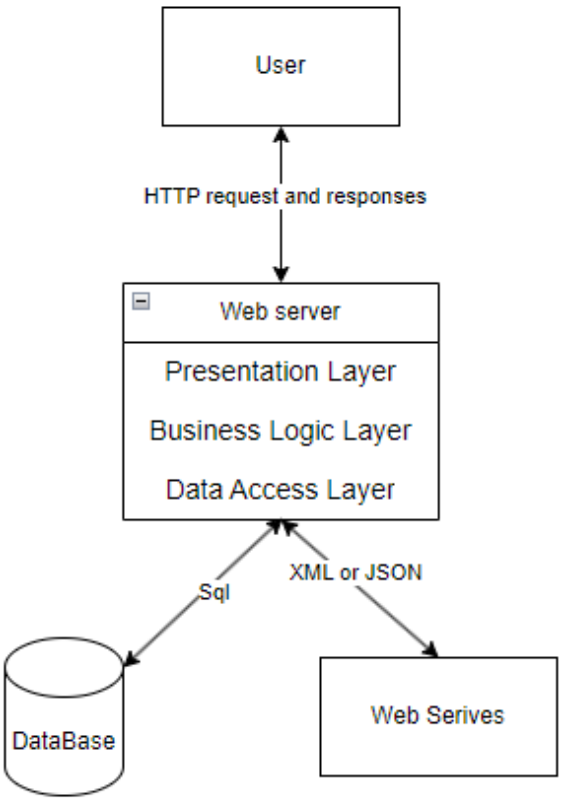


Рисунок 3.1 - Приклад трьох шаровій архітектурі

Для роботи з даними я обрав графову базу даних Neo4j. Neo4j — це потужна графова база даних, яка спеціалізується на збереженні та аналізі даних з складними взаємозв'язками. Вона базується на графовій моделі даних, де дані представлені у вигляді вузлів (nodes), ребер (relationships) та властивостей (properties). Переваги даної бази даних.

Висока продуктивність. Neo4j забезпечує швидке виконання запитів на графах, особливо при обробці складних та великих взаємозв'язків. Це досягається завдяки прямому доступу до пов'язаних вузлів без необхідності складних об'єднань та великого проектування самих моделей перед цим.

Гнучкість та природна модель даних. Графова модель даних є інтуїтивно зрозумілою і добре підходить для моделювання реальних об'єктів та їх взаємозв'язків. Це дозволяє легко адаптувати модель даних до змін в бізнес-вимогах без складних міграцій. На додачу даний веб-застосунок має основну роботу через графи тому у результаті у нас є можливість мати візуальне відображення створеного графа через функціонал бази даних.

Приклад графової бази даних на рисунку 3.2

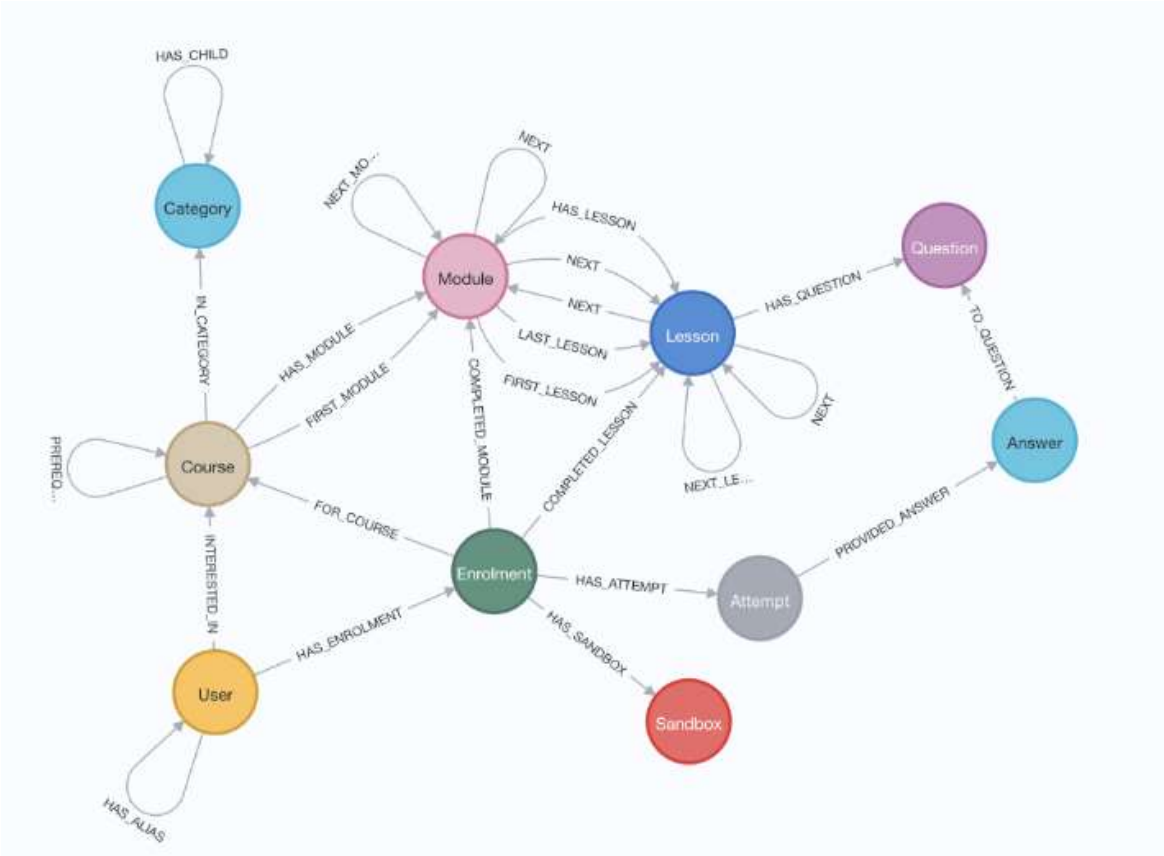


Рисунок 3.2 - Демонстрація графової бази даних Neo4j

Давайте порівняємо з іншими рішеннями:

Amazon Neptune - це керована графова база даних, яка підтримує дві основні моделі графів: Property Graph та RDF (Resource Description Framework). Neptune надає повністю кероване середовище з автоматичним резервним копіюванням, відновленням, масштабуванням і оновленнями без зупинки.

Основні особливості:

– висока доступність. Розподіляє дані в трьох доступних зонах для забезпечення високої доступності та стійкості до відмов;

– підтримка декількох моделей графів. Підтримує Property Graph через інші інфраструктурні проекти;

– безпека. Інтеграція з AWS Identity and Access Management (IAM), шифрування даних під час передачі та на зберіганні, контроль доступу на основі ролей.

PostgreSQL — це потужна, відкрита реляційна база даних з підтримкою SQL. Вона відома своєю надійністю, потужними можливостями та підтримкою стандартів SQL.

Основні особливості:

– розширюваність. Підтримка користувацьких типів даних, функцій, операцій та індексів;

– сумісність зі стандартами. Висока відповідність стандартам SQL, включаючи підтримку складних запитів та транзакцій;

– безпека. Вбудовані механізми автентифікації, авторизації та шифрування даних.

Не дивлячись на те що інші бази даних мають багатий функціонал та можливості роботи з ним, саме Neo4j забезпечує як високу продуктивність, масштабованість та безпеку даних, що є дуже важливим для даного веб-застосунку.

3.2 Обґрунтування вибору засобів розробки

На сьогоднішні день існує дуже багато різних мов програмування, які можуть виконати весь функціонал. Основними мовами програмування для серверної частини програми є: C#, Java, Node.js. Для клієнтської частини даного проекту було використано HTML, CSS, JavaScript та TypeScript.

Мова програмування C#. Дана мова є найпопулярнішою мовою для платформи .NET - вільного, крос-платформного, відкритого середовища розробки. Програми на C# можуть працювати на багатьох різних пристроях, від пристроїв Інтернету речей (IoT) до хмарних сервісів і скрізь, де тільки можна. Ви можете писати програми для телефонів, настільних і портативних комп'ютерів та серверів. [7]

Плюси:

- простота вивчення;
- універсальність;
- безпека;
- інтеграція з Microsoft.

Мінуси:

- швидкість;
- складність для початківців.

Мова програмування Java – це об'єктно-орієнтована мова програмування загального призначення. Її широко використовують для розробки різноманітного програмного забезпечення, включаючи веб-додатки, серверні програми, мобільні додатки, вбудовані системи та багато іншого. [8]

Плюси:

- простота вивчення;
- кроссплатформність;
- об'єктно-орієнтованість;
- безпека.

Мінуси:

- швидкість;

– складність;
– витрата великого обсягу пам'яті.

Node.js – це середовище виконання JavaScript з відкритим кодом, засноване на двигуні V8 від Chrome. Воно дозволяє розробникам писати серверні програми на JavaScript, що робить його популярним вибором для створення динамічних веб-застосунків та мережевих сервісів. [9]

Плюси:

– швидкість та масштабованість;
– простота вивчення;
– багато різних бібліотек.

Мінуси:

– одно потоковість;
– необхідність у зовнішніх бібліотеках;
– може бути складним у пошуку помилки.

Для даного проекту для сервісної частини було зроблено вибір на мову програмування C# обумовлений використанням таких технологій, як ASP.NET. Вона дає потужний інструментарій для розробки веб-додатків, інтеграцію з іншими сервісами, та підтримує сучасні стандарти безпеки.

Для клієнтської частини було обрана мова програмування JavaScript з використання фреймворку під назвою Angular.

Angular є потужним фреймворком для розробки SPA (Single Page Application) – додатків, які функціонують без перезавантаження сторінки. Основні можливості та переваги Angular включають:

Компонентна архітектура: Додаток складається з компонентів – невеликих незалежних блоків коду, які можна повторно використовувати та легко керувати.

Двостороннє зв’язування даних: Angular забезпечує можливість зв’язування даних між компонентами та шаблонами, що дозволяє динамічно змінювати дані в застосунку при зміні моделі.

Сервіси та ін’єкція залежностей: Сервіси надають загальну логіку та функціональність для додатка, а ін’єкція залежностей дозволяє компонентам отримувати доступ до сервісів без необхідності створення їхніх екземплярів вручну.

Маршрутизація: Angular включає механізм маршрутизації, який пов’язує компоненти з різними URL, що забезпечує переходи між різними сторінками без перезавантаження всієї програми.

Цикл життя компонентів: Кожен компонент проходить через певні стадії життєвого циклу, що надає хуки для виконання дій на різних етапах існування компонента. [10]

Також потрібно не забувати про середовище розробки, в якому вона буде розроблятися. Для мови програмування існує декілька відомих середовищ розробки. Із найвідоміших це Visual Studio, Rider, Visual Studio Code.

Visual Studio.

Переваги:

– повний набір інструментів. Включає все необхідне для розробки великих проектів, включаючи редактор коду, дебагер, профілювання, аналіз коду та багато іншого;

– підтримка багатьох мов: Підтримує C#, C++, F#, Python, JavaScript та інші;

– інтеграція з Azure: Глибока інтеграція з хмарними сервісами Azure;

– вбудований дебагер: Потужний інструмент для налагодження програм;

– розширення: Широкий вибір розширень та плагінів.

Недоліки:

– важкий: Велика інсталяція та важка у потребі ресурсів;

– вартість: Платна ліцензія для професійної та корпоративної версій.

Rider.

Переваги:

– продуктивність: Висока швидкість роботи та швидке завантаження;

– кросплатформеність: Працює на Windows, macOS та Linux;

– інтелектуальний редактор: Потужні можливості для рефакторингу та навігації по коду;

– інтеграція з іншими продуктами JetBrains: Хороша інтеграція з іншими інструментами JetBrains, такими як ReSharper.

Недоліки:

– вартість: Платна ліцензія, хоча є безкоштовна версія для студентів та відкритих проектів;

– споживання ресурсів: Може бути дуже сильно потреба у ресурсів, особливо на великих проектах.

Visual Studio Code

Переваги:

– легкий та швидкий: Невеликий розмір інсталяції та швидке завантаження;

– безкоштовний: Повністю безкоштовний з відкритим кодом;

– кросплатформеність: Працює на Windows, macOS та Linux;

– розширення: Величезна кількість доступних розширень для підтримки різних мов та інструментів;

– гнучкість: Можливість налаштування середовища під власні потреби.

Недоліки:

– обмежений функціонал: Не включає деякі функції, які є у повноцінних IDE, таких як Visual Studio;

– плагіни: Деякі функції доступні тільки через сторонні плагіни, що може вимагати додаткової конфігурації;

– інтеграція: Менша інтеграція з деякими інструментами порівняно з Visual Studio.

Для розробки веб-додатку було обрано середовище розробки, як Visual Studio, оскільки це потужне інтегроване середовище розробки (IDE), яке надає широкий спектр інструментів та підтримку різних мов програмування. Visual Studio вирізняється зручним інтерфейсом, вбудованим хорошим функціоналом, можливість легко підключати бібліотеки, зручний дебагер, завдяки якому, можна відстежити кожну строчку роботи програми.

Для розробки даного веб-застосунку доведеться використати декілька потрібних сторонніх бібліотек та фреймворків. До них входить Asp.Net Core, Neo4j.Driver, FluentValidation.

ASP.NET Core — це кросплатформенний фреймворк для розробки веб-додатків, який створено компанією Microsoft. Він є частиною .NET Core платформи.

Основні особливості:

– кросплатформеність: Працює на Windows, macOS і Linux;

– висока продуктивність: Оптимізований для швидкої роботи та низького споживання ресурсів;

– модульна архітектура: Легка конфігурація та розширення завдяки системі модулів;

– інтеграція з хмарою: Добре інтегрується з хмарними сервісами, такими як Azure;

– розробка API: Зручні засоби для створення RESTful API.

Neo4j.Driver — це офіційний клієнтський драйвер для роботи з Neo4j, графовою базою даних. Цей драйвер дозволяє підключатися до бази даних Neo4j з використанням різних мов програмування, включаючи C#.

Основні особливості:

– підтримка Cypher: Виконання Cypher-запитів до бази даних;

– багатопоточність: Підтримка багатопоточних операцій для підвищення продуктивності;

– інтеграція з .NET: Легка інтеграція з .NET додатками.

FluentValidation — це бібліотека для .NET, яка надає зручний та інтуїтивно зрозумілий спосіб валідації об'єктів, використовуючи вирази C#.
Основні особливості:

– чіткий та зручний синтаксис: Використання лямбда-виразів для визначення правил валідації;

– можливість налаштування: Легко налаштовується та розширюється під специфічні вимоги;

– інтеграція з ASP.NET Core: Можливість інтеграції з ASP.NET Core для автоматичної валідації моделей.

3.3 Конструювання програмного забезпечення

Для роботи з бази даних на потрібно створити моделі, які будуть зберігати базові данні для топологічних графів. Результат на таблиці 3.2

Таблиця 3.2 – Моделі

Назва	Опис	Контекст
Node	Модель вузла	Контекст вузла
Relationship	Модель ребра	Контекст ребра

Далі, нам потрібно розписати презентаційний шар в якому буде розписано усі базові файли.

Таблиця 3.3 – Презентаційний шар

Назва	Опис
DataConst.cs	Константи зі значеннями
Neo4JService.cs	Сервіс підключення до бази даних
Node.cs	Модель вузла
Relationship.cs	Модель ребра
ICommonService.cs	Інтерфейс базових методів для графа сервіс
INodeService.cs	Інтерфейс базових методів сервісу вузла
IRelationshipService.cs	Інтерфейс базових методів сервісу ребра
CoommonService.cs	Виклик логіки базових методів для графа
NodeService.cs	Виклик логіки вузла для графа
RelationshipService.cs	Виклик логіки вузла для ребра
CommonRepository.cs	Реалізація логіки базових методів для графа
NodeRepository.cs	Реалізація логіки методів для вузла
RelationshipRepository.cs	Реалізація логіки методів для ребра
AnalysisAlgorithm.cs	Реалізація аналітичного методу
AStarAlgorithm.cs	Реалізація A* алгоритму
DijkstraAlgorithm.cs	Реалізація алгоритму Дейкстри
RelationshipCheckAlgorithm.cs	Реалізація перевірки ребр

Далі зробимо опис таблиць для бази даних. Нам потрібно зберігати данні для роботи з графом, тому нам потрібно реалізувати таблиця для вузла та ребра. Опис таблиць буде в таблицях в 3.4 та 3.5.

Таблиця 3.4 – Опис таблиці Node

Назва поля	Тип даних	Опис
Id	Guid	Ідентифікаційний номер користувача
Name	String	Назва вузла
CreatedOn	DateTime	Дата створення

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
Position	Int	Позиція вузла
Edge	List< Edge>	Список ребр, які належать до вузла

Таблиця 3.5 – Опис таблиці Edge

Назва поля	Тип даних	Опис
Id	Guid	Ідентифікаційний номер користувача
Weight	Int	Вага ребра
EndNode	Guid	Посилання на вузол до якого він переходить

Далі розпишемо увесь перелік бібліотек та інших сторонніх програм,Ю що були використані під час розробки даного веб-застосунку. Результат на таблиці 3.6 – 3.8

Таблиця 3.6 – Опис утиліт, які були використані

Назва утиліти	Опис застосування
Visual Studio 2022	Головне середовище розробки програмного забезпечення.
Visual Studio Code	Допоміжне середовище розробки програмного забезпечення
Google Chrome	Програмне забезпечення, в якому буде працювати веб-застосунком.
Draw.io	Додаток в якому було реалізовано діаграми

Продовження таблиці 3.6

Назва утиліти	Опис застосування
Postman	Спеціальний інструмент для розробників, який використовується для тестування API
BPMN.io	Додаток, в якому було реалізовано BPMN моделі

Таблиця 3.16 – Опис бібліотек

Назва бібліотек	Опис застосування
Neo4j.Driver	Клієнтський драйвер для роботи з графовою базою даних Neo4j
FluentValidation	Бібліотека, яка дозволяє визначати правила валідації для різних моделей
Microsoft.AspNetCore	Спеціальна бібліотека, яка є частиною фреймворку ASP.NET Core. Вона призначена для створення веб-додатків і API

Таблиця 3.8 – Опис фреймворків, які були використані

Назва утиліти	Опис застосування
ASP.NET Core	кросплатформенний фреймворк для розробки веб-додатків, розроблений компанією Microsoft.
Angular	фреймворк для розробки фронтенд додатків

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Одним із ключових засобів захисту даних для веб-застосунків є використання протоколу HTTPS для шифрування даних, що передаються між користувачем та сервером. HTTPS (Hypertext Transfer Protocol Secure) забезпечує конфіденційність, цілісність і автентичність даних завдяки використанню протоколу TLS (Transport Layer Security).

Конфіденційність даних. При використанні HTTPS дані, що передаються між клієнтом (браузером) та сервером, шифруються. Це означає, що навіть якщо зловмисник перехопить ці дані, він не зможе їх прочитати або використовувати без відповідного ключа дешифрування. Шифрування захищає особисту інформацію користувачів, таку як паролі, номери кредитних карток та інші конфіденційні дані, від крадіжки під час передачі.

Цілісність даних. HTTPS забезпечує цілісність даних, тобто захист від їх несанкціонованої модифікації під час передачі. Якщо зловмисник спробує змінити дані, що передаються між клієнтом та сервером, це буде виявлено, і дані будуть відкинуті. Це гарантує, що інформація, яку отримує користувач, є автентичною і не була змінена під час передачі.

На додачу при встановленні HTTPS-з'єднання, сервер представляє клієнту цифровий сертифікат, який підтверджує його автентичність. Цей сертифікат видається довіреними сертифікаційними центрами і дозволяє клієнту переконатися, що він з'єднується саме з тим сервером, з яким мав намір. Це захищає користувачів від атак типу людини, які знаходяться посередині, де зловмисник може підробити сервер і перехопити всі дані, що передаються між клієнтом і справжнім сервером.

На останнє протокол HTTPS також захищає від проміжних атак, при яких зловмисник може спробувати перехопити або змінити дані, що передаються між клієнтом та сервером. Оскільки дані шифруються, зловмисник не може їх розшифрувати або змінити без відповідних ключів шифрування. Крім того, використання сертифікатів автентичності допомагає виявляти спроби атак на підміну серверів.

Хочу підкреслити, що захист з бази даних також дуже важливий. Neo4j забезпечує комплексний підхід до безпеки бази даних, включаючи різні рівні контролю доступу, шифрування та автентифікації. Основні аспекти безпеки Neo4j включають:

– контроль доступу. Рольова модель безпеки: Neo4j використовує рольову модель для управління доступом до даних. Адміністратори можуть створювати ролі з визначеними правами доступу та призначати їх користувачам. Це дозволяє контролювати, хто має доступ до певних частин бази даних. Права доступу: Користувачі можуть мати різні рівні доступу, такі як читання, запис, створення або видалення даних. Це дозволяє налаштовувати гнучкі політики безпеки відповідно до потреб організації;

– автентифікація та авторизація. Вбудована автентифікація: Neo4j підтримує автентифікацію користувачів за допомогою вбудованих механізмів або інтеграцію з зовнішніми системами автентифікації, такими як LDAP. Це забезпечує безпечний вхід у систему та управління доступом. Авторизація: Після автентифікації користувачів, система визначає їхні права доступу на основі призначених ролей і політик безпеки. Це дозволяє забезпечити, що користувачі мають доступ лише до тих ресурсів, які їм дозволені;

– шифрування. Шифрування на транспортному рівні (TLS): Neo4j підтримує шифрування даних під час передачі між клієнтом та сервером за допомогою протоколу TLS (Transport Layer Security). Це забезпечує захист даних від перехоплення та несанкціонованого доступу під час їх передачі по мережі. Шифрування на рівні зберігання: Neo4j також підтримує шифрування даних на рівні зберігання. Це означає, що дані шифруються перед записом на диск, забезпечуючи їх захист навіть у випадку фізичного доступу до носія даних;

– Логування та вивід інформації про всі дії програми. Аудит та журналювання: Neo4j забезпечує можливість ведення журналів активності користувачів та системи. Це дозволяє відстежувати доступ до даних, зміни в

базі даних та інші важливі події. Логування та аудит допомагають виявляти та реагувати на потенційні загрози безпеці;

– Захист від атак. Запобігання ін'єкціям: Використання Cypher, спеціальної мови запитів для графових баз даних, знижує ризик ін'єкцій, характерних для SQL-базованих систем. Проте, важливо дотримуватися найкращих практик з написання безпечного коду для запобігання іншим типам атак.

Висновки до розділу

В даному розділі дипломної роботи було виконано конструювання програмного забезпечення. Було детально розроблено архітектуру програмного забезпечення, пояснення обраної багаторівневої архітектури.

Далі було детально розписано можливі технології, які можна використати для реалізації даного веб-застосунку. Було описано види мови програмування, які найбільше всього підходять для реалізації даного застосунку. Було коротко розписано про такі мови програмування для серверної частини, до них входили як: C#, Java та Node.js. Для реалізації даного застосунку було обрано мову програмування C#. Також було розписано короткий опис мови програмування для клієнтської частини через стандартизацію та популярність мови програмування JavaScript через його зручність та велику кількість технологій написані для нього, було обрано саме його.

На додачу було зроблено короткий опис та написано плюсів та мінусів середовищ розробки для реалізації даного застосунку. Для реалізації було обрано середовище розробки Visual Studio.

Далі було детально розписано кожен шлях архітектури, було створено спеціальні таблиці, які описували функціонал програми, структуру класів та розписано, які моделі було створено для бази даних.

Також було проведено глибокий аналіз безпеки застосунку. Безпека досягається шляхом використання протоколу HTTPS та особливості захисту

бази даних Neo4j. Завдяки цьому дозволяє безпечно зберігати дані користувача.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Процес розробки програмного забезпечення обов'язково включає етап аналізу якості, що дозволяє оцінити відповідність програми встановленим стандартам якості. Цей етап є критично важливим, оскільки забезпечує впевненість у надійності та ефективності роботи продукту.

Основні цілі тестування включають:

– перевірку правильності функціонування програмного забезпечення згідно з визначеними функціональними вимогами;

– перевірку цілісності та безпеки даних;

– оцінку сумісності веб-додатку з актуальними версіями сучасних браузерів (таких як Chrome, Opera, Firefox тощо);

– виявлення дефектів, помилок та інших недоліків з метою їх оперативного усунення;

– оцінку зручності та ергономічності графічного інтерфейсу користувача.

Для оцінки якості програмного забезпечення використовуються такі метрики:

– функціональність. Аналіз відповідності реальних функцій додатку до заявлених вимог, включаючи коректність виконання операцій та точність результатів;

– надійність. Оцінка стабільності роботи ПЗ в умовах нормальної та підвищеної завантаженості, включаючи відсоток відмов та час відновлення після збоїв;

– продуктивність. Вимірювання швидкості реакції системи на запити користувачів, час завантаження сторінок, а також ефективність використання ресурсів;

– сумісність. Перевірка коректності роботи ПЗ на різних платформах, пристроях та у різних середовищах, включаючи різні веб-браузери та операційні системи;

– зручність використання. Оцінка зручності та інтуїтивності інтерфейсу користувача, включаючи простоту навігації, доступність функцій та задоволеність користувачів;

– безпека. Аналіз здатності ПЗ захищати дані та підтримувати конфіденційність користувачів, включаючи виявлення та усунення можливих вразливості;

– мобільність. Перевірка адаптивності інтерфейсу та функціональності ПЗ на різних мобільних пристроях, включаючи смартфони та планшети.

Для більш детальної оцінки якості програмного забезпечення був застосований сервіс embold.io [12]. Даний сервіс має можливість зробити аналіз коду. Він є досить відомим інструментом для аналізу коду, оскільки виявляє помилки, вразливості та архітектурні проблеми. На рисунку 4.1 демонстрація аналізу програмного забезпечення за допомогою даного сервісу.

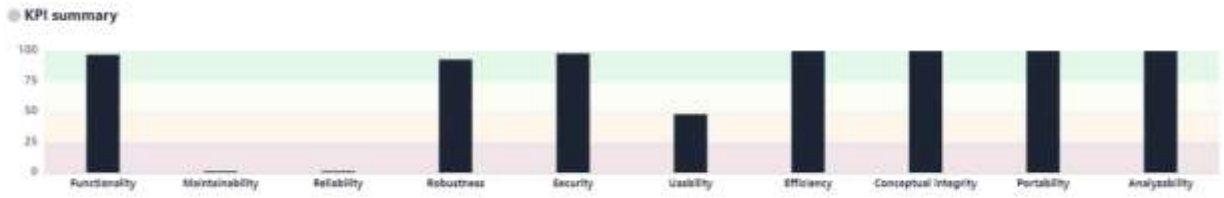


Рисунок 4.1 - Результат аналізу якості програмного забезпечення.

Тут розбито на декілька компонентів. Пройдемося по кожному з них.

– функціональність. Проект має високий рівень функціональності, що означає, що всі необхідні функції та можливості реалізовані та працюють належним чином;

– підтримка. Рівень підтримка дуже низька. Це свідчить про те, що проект важко підтримувати, оновлювати або модифікувати через складний код або недостатню документацію. Це можна підкреслити через не сильні відомість графової бази даних neo4j;

– надійність. Аналогічна ситуація, надійність проекту також дуже низька. Можливо через велику кількість неочікуваного результату роботи програми в різних ситуаціях;

– міцність. Проект демонструє високий рівень міцності, що означає здатність системи протистояти та відновлюватися від помилок або несправності;

– безпека. Безпека проекту також на високому рівні, що свідчить про ефективний захист від потенційних загроз;

– зручність використання. Зручність використання має середній показник. Це може означати, що інтерфейс користувача або досвід роботи з системою може бути покращений для більшої зручності користувачів;

– ефективність. Проект має високий рівень ефективності, що вказує на добру продуктивність системи та оптимальне використання ресурсів;

– концептуальна цілісність. Концептуальна цілісність також на високому рівні, що свідчить про добре продуману та узгоджену архітектуру проекту;

– переносимось. Переносимось проекту висока, що означає легкість перенесення системи на інші платформи або середовища;

– паралізованість. Дана категорія також на високому рівні, що вказує на легкість аналізу та оцінки системи для виявлення проблем або можливостей для покращення.

Загалом, даний проект демонструє високу функціональність, міцність, безпеку, ефективність, концептуальну цілісність, переносимось та паралізованість, але має низькі показники підтримки та надійності, а також середній рівень зручності використання. Це може вказувати на необхідність покращення деяких пунктів системи, а також зручності використання для кінцевих користувачів.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 - 4.10

Таблиця 4.1 – Текст 1.1 Додавання вузла

Початковий стан системи	Користувач знаходиться на сторінці створення вузла
Вхідні дані	Назва вузла, позиція вузла
Опис проведення тесту	У відповідні поля вводяться назва вузла та його позиція. Після перевірки, якщо не було знайдено такого вузла, система створює його та додає до бази даних
Очікуваний результат	Система додає вузол до бази даних та пише про успішне створення вузла у системі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Текст 1.2 Створення ребра

Початковий стан системи	Користувач знаходиться на сторінці створення ребра
Вхідні дані	Назва вузла з якого починається шлях, Назва вузла, яким закінчується шлях
Опис проведення тесту	У відповідні поля вводяться назві першого вузла та назва другого вузла. Після перевірки, якщо не було знайдено помилок, система створює зв’язок між вузлами та додає до бази даних
Очікуваний результат	Система додає ребро між двома вузлами та пише про успішне об’єднання двох вузлів у системі.

Продовження таблиці 4.2

Початковий стан системи	Користувач знаходиться на сторінці створення ребра
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Текст 1.3 Об’єднання граф на граф

Початковий стан системи	Користувач знаходиться на сторінці об’єднання граф на граф
Вхідні дані	Назва першого графа та назва другого графа
Опис проведення тесту	У відповідні система об’єднує два графа, а саме кожний вузол першого графа створює зв’язок з кожним вузлом у другого графа
Очікуваний результат	Система об’єднає два графа
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Текст 1.4 Імпорт графа

Початковий стан системи	Користувач знаходиться на сторінці імпорту графа
Вхідні дані	
Опис проведення тесту	Користувач обирає імпортування графа із зовнішньої системи, йому із зовнішнього середовище приходять дані про граф, який буде записаний у нього в базу даних
Очікуваний результат	Користувач імпортував систему та може працювати вже з нею у себе на середовищі
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Текст 1.5 Експорт графа

Початковий стан системи	Користувач знаходиться на сторінці експорт графа
Вхідні дані	
Опис проведення тесту	Користувач обирає експортування графа із своєї системи, дана система передає із локального середовища у зовнішні будуть відправлятися дані про граф.
Очікуваний результат	Користувач експортував систему у зовнішню среду
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Текст 1.6 Об’єднання графів

Початковий стан системи	Користувач хочу об’єднати два графа
Вхідні дані	Назва першого графа, назва другого графа
Опис проведення тесту	У відповідні поля водяться: назва вузлів першого графа та назва вузлів другого графа, після ведення натискає на кнопку «Об’єднати»
Очікуваний результат	Користувач об’єднує два графи. Користувач може переглянути, вже перетворену систему.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Текст 1.7 Створення топологічного графа «жирне дерево»

Початковий стан системи	Користувач хоче створити топологічне дерево «жирне дерево»
Вхідні дані	Кількість головних вузлів

Продовження таблиці 4.7

Початковий стан системи	Користувач хоче створити топологічне дерево «жирне дерево»
Опис проведення тесту	У відповідне поле водиться: кількість вузлів, після даного ведення, натискається кнопка «створити граф»
Очікуваний результат	Система створює користувачу топологічне дерево «жирне дерево»
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Текст 1.8 Створення топологічного графа на основі кодових перетворень

Початковий стан системи	Користувач хоче створити топологічний граф на основі кодових перетворень
Вхідні дані	Кількість позицій
Опис проведення тесту	У відповідне поле водиться: кількість позицій, після даного ведення натискається кнопка «Створити граф»
Очікуваний результат	Система створює користувачу топологічне дерево на основі кодових перетворень
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Текст 1.9 Об’єднання графів використовуючи «Декартовий добуток топологій»

Початковий стан системи	Користувач хоче об’єднати графові системи, використовуючи метод «Декартовий добуток топологій»
Вхідні дані	Назва першого графа та назва другого графа

Продовження таблиці 4.9

Початковий стан системи	Користувач хоче об’єднати графові системи, використовуючи метод «Декартовий добуток топологій»
Опис проведення тесту	У відповідне поле водиться: назва першого графа та назва другого графа, після даного ведення натискається кнопка «Об’єднати граф»
Очікуваний результат	Система створює користувачу топологічне дерево на основі «Декартовий добуток топологій»
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Текст 1.10 Створення випадкової топологічної системи

Початковий стан системи	Користувач хоче створити нову топологічну систему з певною кількістю вузлів
Вхідні дані	Кількість вузлів
Опис проведення тесту	У відповідне поле водиться: певна кількість вузлів, після даного воду натискається кнопка «Згенерувати граф»
Очікуваний результат	Система створює користувачу випадкову збудовано топологічну графову систему
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Для кожного тестування буде продемонстровано окремий результат роботи.

Користувач створив вузол. Результат на рисунку 4.2

Рисунок 4.2 - Створення вузла

Далі перейдемо до створення об’єднання, а саме ребр між вузлами. Результат створення на рисунку 4.3

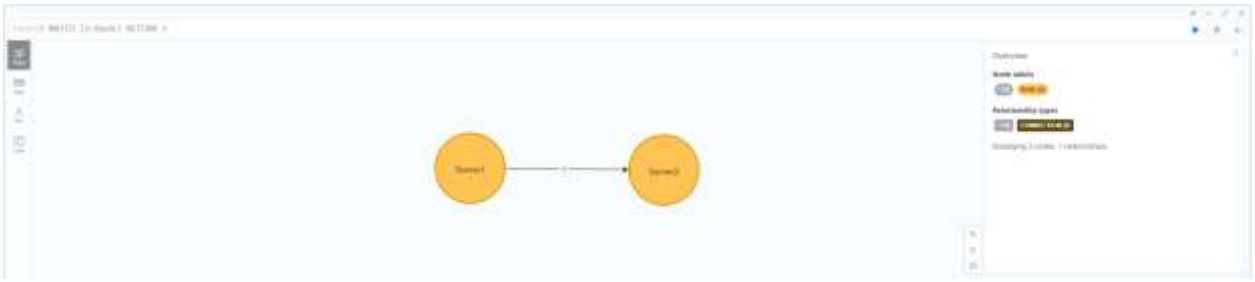


Рисунок 4.3 - Створення ребра для об’єднання вузлів

Далі створимо приклад для об’єднання графів через кореневий добуток. На рисунку 4.4 було створено дві не великі графові системи.

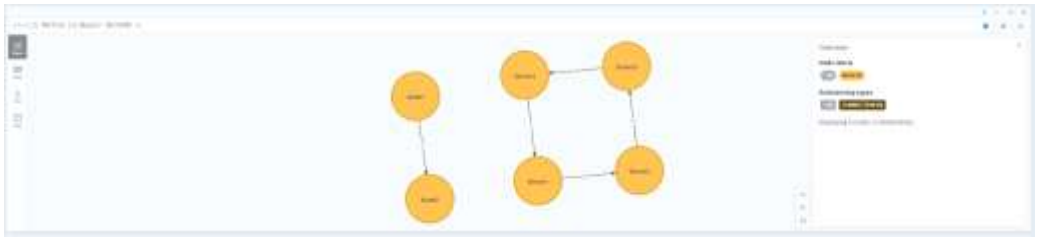


Рисунок 4.4 - Дві графові системи

Результат їх об’єднання через кореневий добуток на рисунок4.5

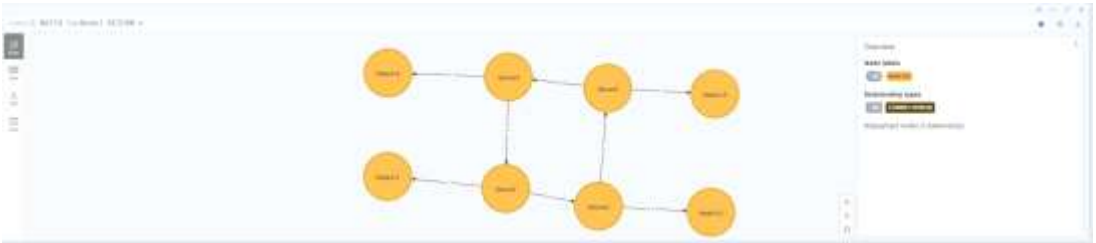


Рисунок 4.5 - Об’єднання через кореневий добуток

Далі давайте імпортуємо систему через зовніше середовище, результат на рисунку 4.6

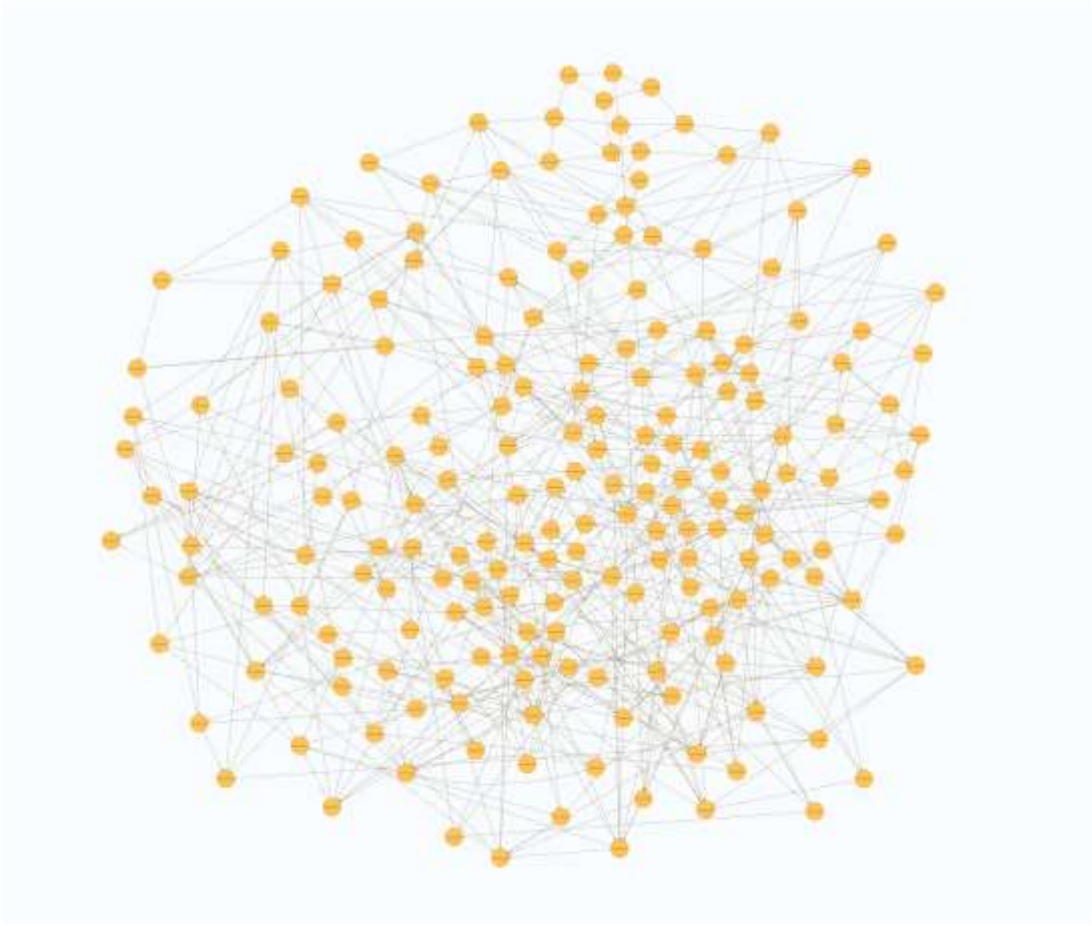


Рисунок 4.6 - Імпортована графова система

Давайте туж саму систему давайте відправимо, вона відправляється через json файл, продемонструю один вузол з’єднання. Результат на рисунку 4.7


```
[
  {
    "id": "b17b7663-dff7-42d3-ad15-fb08864bfb36",
    "name": "Server1,Node1",
    "createdOn": "2024-06-03T22:32:36.457+03:00",
    "position": -1,
    "color": "",
    "edge": [
      {
        "id": "7a139098-fe64-40c6-bedb-c180224d1067",
        "weight": 1,
        "endNode": "7a139098-fe64-40c6-bedb-c180224d1067"
      },
      {
        "id": "81e2ab63-e883-478a-a49b-153dfa87d099",
        "weight": 1,
        "endNode": "81e2ab63-e883-478a-a49b-153dfa87d099"
      },
      {
        "id": "62803db2-0c37-4e65-9164-a2fe479da286",
        "weight": 1,
        "endNode": "62803db2-0c37-4e65-9164-a2fe479da286"
      }
    ]
  },
  {
    "id": "b4c28e84-2346-4b7d-b486-a35d676d7924",
    "name": "Server1,Node2",
```


Рисунок 4.7 - Інформація про один вузол

Далі давайте об’єднаємо декілька графів вузлами, щоб кожен вузол одного графа мав зв’язок до іншого вузла, результат на рисунку 4.8

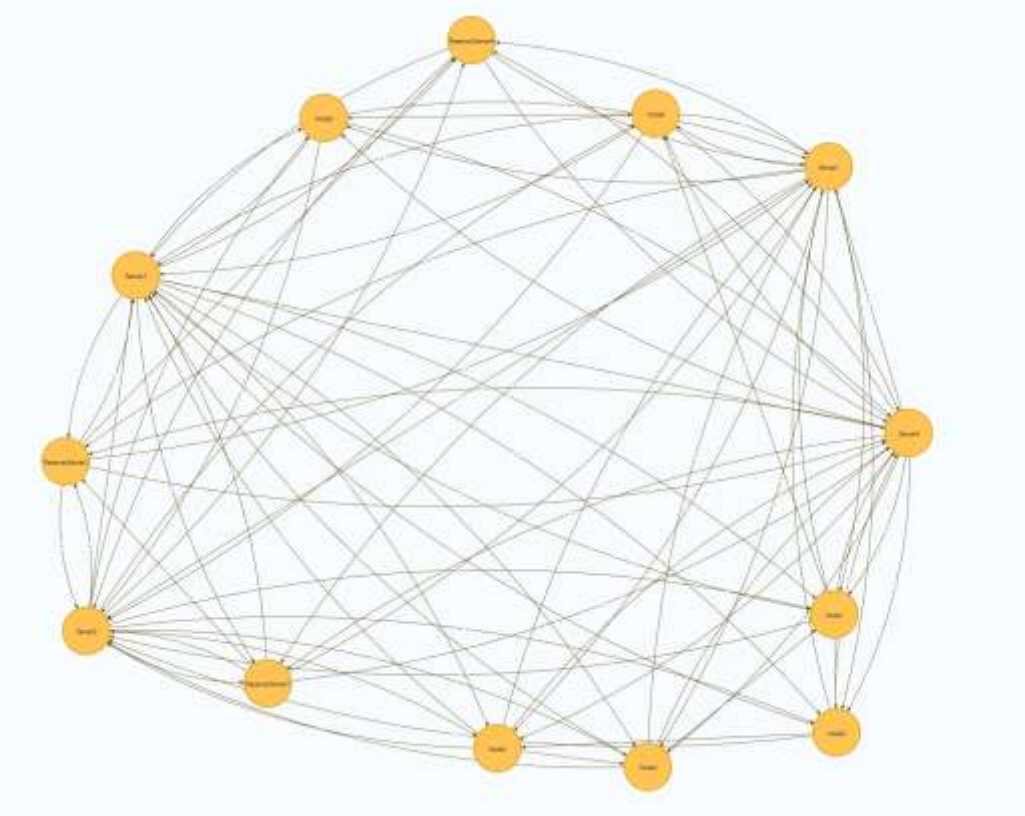


Рисунок 4.8 - Результат об’єднання декількох маленьких графів

Наступним кроком давайте згенеруємо жирне дерево, для цього давайте ведемо, щоб вузлів «Core» було 10 штук, результат на рисунку 4.9

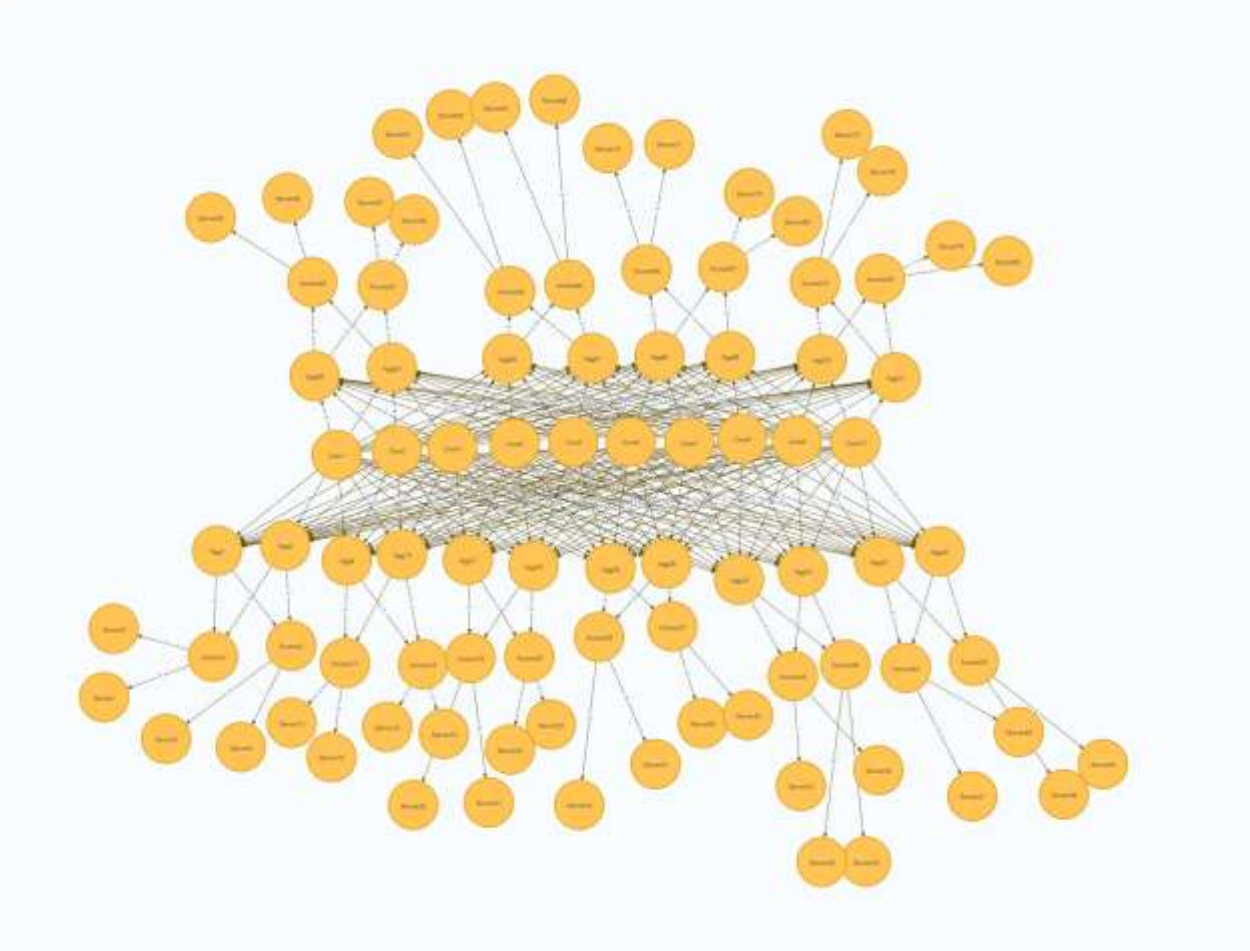


Рисунок 4.9 - Побудовано жирне дерево

Наступним кроком буде об’єднання графів через кодове переміщення. Результат на рисунку 4.10

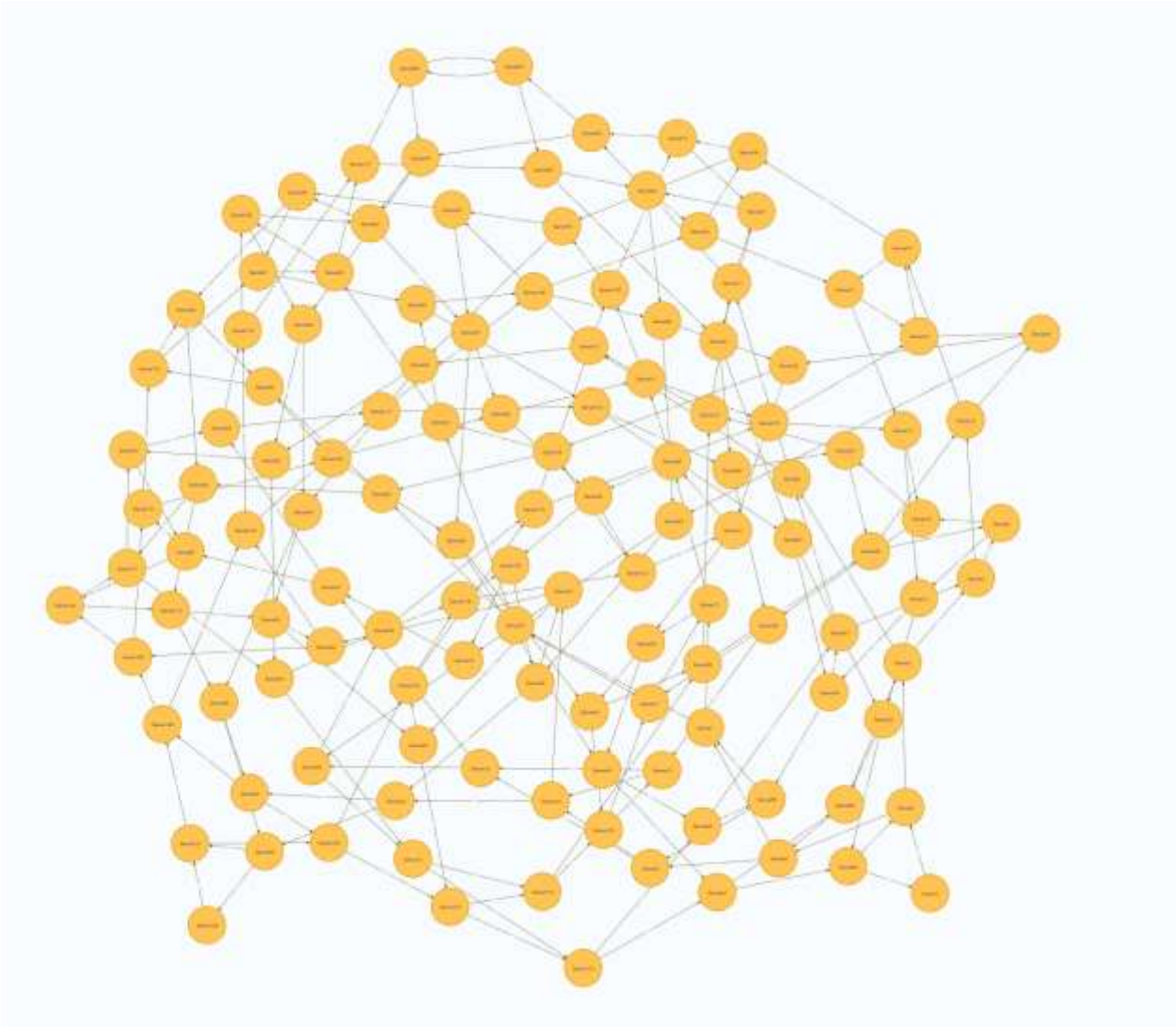


Рисунок 4.10 - Результат об’єднання графів через кодове переміщення

Наступним кроком буде об’єднання графів через декартовий добуток топологій. Результат на рисунку 4.11

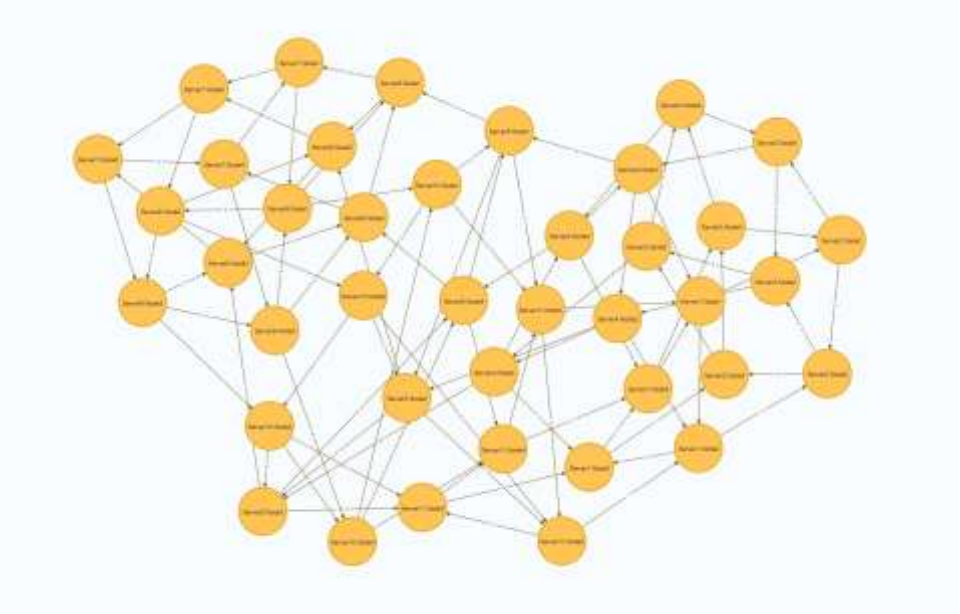


Рисунок 4.11 - Результат об’єднання графів через декартовий добуток топологій

Останнім кроком, буде випадкова генерація системи, було задано 300 вузлів. Результат на рисунку 4.12

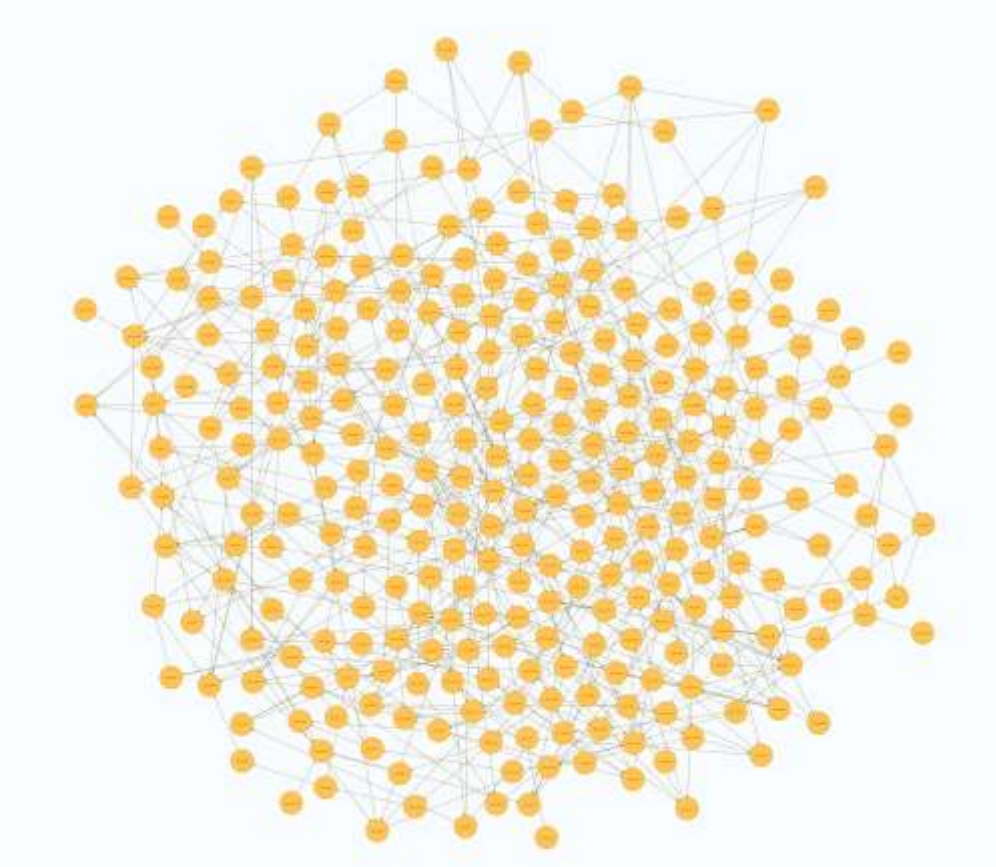


Рисунок 4.12 - Випадково створена система

Висновки до розділу

У цьому розділі детально розглянуто методи і засоби для аналізу якості програмного забезпечення. Для аналізу коду використовувався інструмент embold.io, що застосовує технології, які надають широкий спектр метрик для оцінки якості вихідного коду. Результати аналізу показали, як високі сторони та і слабкі даного програмного застосунку. Попередній аналіз виявив певні потенційні ризики у коді, але вже були усунуті після детального аналізу та виправлення даного додатку. Було також розроблено набір тест-кейсів, які охоплюють різні можливі сценарії використання програмного забезпечення. Проведено ручне тестування, яке дозволило оцінити продукт з точки зору користувача та адаптувати його до реальних умов експлуатації.

В останньому підрозділі створено контрольний приклад з демонстрацію його роботи. Цей приклад включав перевірку основного функціоналу даного продукту. Після успішного відтворення контрольного прикладу програмне забезпечення пройшло повну перевірку.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання програмного забезпечення включає кілька кроків, що охоплюють підготовку середовища, налаштування сервісів та забезпечення коректної роботи всіх компонентів системи. У даному розділі буде розглянуто процес розгортання проекту, що включає сервер баз даних Neo4j, .NET Web API та клієнтський додаток на Angular.

Нам потрібно підготувати середовище бази даних. Для цього нам потрібно його встановити та потім налаштувати. Результат роботи даної бази даних на рисунку 5.1

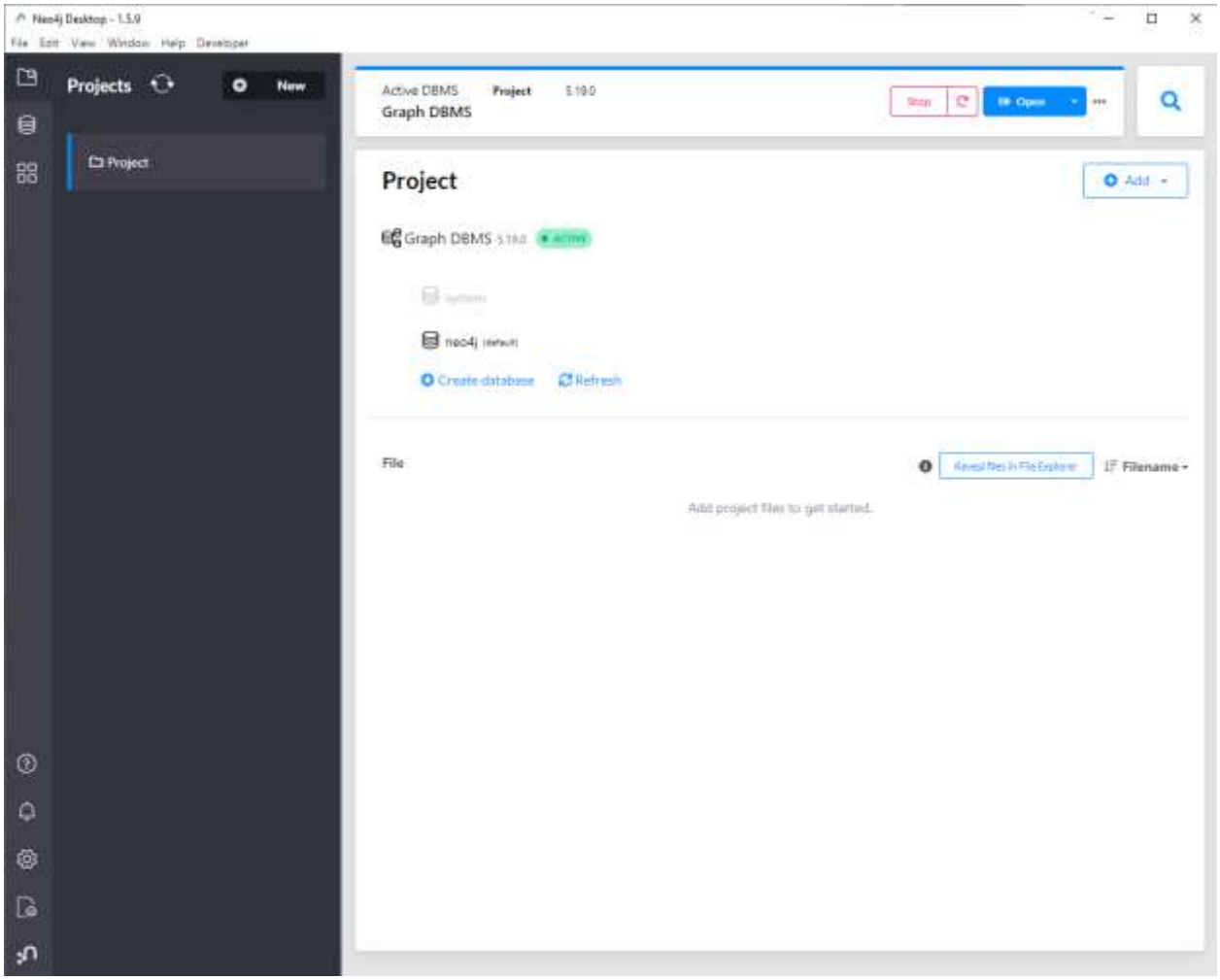


Рисунок 5.1 - Запущена база даних Neo4j

Далі потрібно розгорнути .Net Web API. Нам потрібно встановити налаштування для роботи з neo4j. Демонстрація на рисунку 5.2

```
var builder = Microsoft.AspNetCore.Builder.WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Add Neo4j driver
builder.Services.AddSingleton<IDriver>(provider =>
{
    var neo4jService = GraphDatabase.Driver(
        DataConst.ConnectionData.url,
        AuthTokens.Basic(DataConst.ConnectionData.user, DataConst.ConnectionData.password));
    return neo4jService;
});
```

Рисунок 5.2 - Налаштування підключення

Наступним кроком нас потрібно створення проекту та сервісів на Angular. Ось які команди потрібно для реалізації. «ng new your-angular-project» та «ng generate service api»

Далі потрібно розгорнути на сервері, для цього було обрано Apache. Треба встановити Apache HTTP Server, виконавши команди: «sudo apt update» та «sudo apt install apache2».

5.2 Супровід програмного забезпечення

Супровід програмного забезпечення включає підтримку, моніторинг та оновлення системи для забезпечення її стабільної роботи та відповідності вимогам користувачів.

Моніторинг системи. Для кращої продуктивності, потрібно налаштувати збір логів, як наприклад ELK Stack та моніторинг системи, як наприклад Prometheus.

Також краще використати інструменти для резервного копіювання файлів та конфігурацій вашого .NET Web API та Angular додатка.

Висновки до розділу

Розгортання та супровід програмного забезпечення є критично важливими етапами у життєвому циклі будь-якого проекту. Правильне налаштування середовища, належна документація та використання найкращих практик забезпечують стабільну та безпечну роботу системи. Супровід,

включаючи моніторинг, резервне копіювання та оновлення, гарантує, що програмне забезпечення залишатиметься ефективним та відповідатиме потребам користувачів. У цьому розділі було розглянуто основні аспекти розгортання та супроводу проекту, що використовує стек технологій Neo4j, .NET Web API та Angular.

ВИСНОВКИ

Метою дипломного проекту є створення платформи для аналізу та проектування топологічних організацій розподілених систем, що дозволяє ефективно працювати з графовими структурами та забезпечує наочну візуалізацію складних мережевих зав’язків. Це допомагає приймати обґрунтовані рішення на основі аналізу даних.

На етапі передпроєктного обстеження було досліджено сучасні тенденції та існуючі рішення в галузі аналізу та візуалізації графів, що дозволило визначити основні напрямки розробки та адаптувати найкращі практики для задоволення вимог проекту. Було сформульовано системні та функціональні вимоги, що охоплюють широкий спектр необхідної функціональності від користувацького інтерфейсу до алгоритмічного забезпечення.

Конструювання та розробка програмного забезпечення зосереджувались на створенні стійкої та масштабованої тришарової архітектури, яка забезпечує гнучкість та легкість у супроводі системи. Структура архітектури спроектована так, щоб підтримувати незалежність компонентів, дозволяючи ізолювати помилки окремих сервісів та оптимізувати їх обслуговування.

База даних, реалізована на Neo4j, була інтегрована для використання її можливостей у зберіганні та обробці графових даних. Використання Neo4j дозволило ефективно керувати складними мережевими структурами та забезпечити швидку обробку запитів.

Користувацький інтерфейс, розроблений на Angular, забезпечує зручний та інтуїтивно зрозумілий доступ до функціональності платформи. Він дозволяє користувачам взаємодіяти з графовими структурами, виконувати аналіз даних та відображати результати у зрозумілому форматі.

Особлива увага в проекті приділена розробці та інтеграції функціоналу для аналізу та проектування топологічних структур, що забезпечує точне та ефективне моделювання складних мережевих зв'язків. Використання

передових технологій дозволило досягти високої продуктивності та надійності системи, що підтверджено результатами тестувань та експлуатації.

Завершальні етапи роботи включали опис процесу розгортання системи, що охоплював налаштування серверного середовища, інтеграцію всіх компонентів та забезпечення їх стабільної роботи. Також було розглянуто підходи до супроводу програмного забезпечення, включаючи моніторинг, резервне копіювання та оновлення системи.

Розроблена платформа демонструє здатність аналізувати, проектувати та відображати топологічні структури розподілених систем, надаючи користувачам потужні інструменти для ефективної роботи з графовими даними. Це сприяє глибшому розумінню складних мережевих взаємодій та прийняттю обґрунтованих рішень на основі аналізу даних.

Враховуючи відкритість даного проекту для спільноти та актуальність аналізу топологічних структур, дана робота може сприяти розвитку наукових досліджень в області інформатики та інженерії, де аналіз мережевих даних є ключовим для розуміння складних систем. Платформа дозволяє науковцям, інженерам та студентам доступатися до потужних інструментів для обробки графових даних та участі у вдосконаленні системи, що стимулює інноваційні підходи та методики в аналізі топологічної інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Різновиди комп'ютерних мереж. Локальні та глобальні мережі. Їхні відмінності. Розподілені інформаційні системи. Регіональні та корпоративні мережі. *StudFiles*. URL: https://studfile.net/preview/7543085/page:6/

2) РОЗРОБКА СТРУКТУРИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОРГАНІЗАЦІЇ ОСОБИСТОГО ТАЙМ-МЕНЕДЖМЕНТУ | Вісник Вінницького політехнічного інституту. *Home Page*. URL: https://doi.org/10.31649/1997-9266-2021-154-1-70-76

3) *DSpace :: ELAKPI :: Репозитарій КПІ ім. Ігоря Сікорського*. URL: https://ela.kpi.ua/server/api/core/bitstreams/e0a0c843-a57d-4d82-8f42-0eba294bef1f/content

4) Using Cisco Packet Tracer to simulate Smart Home – IJERT. *Home Page*. URL: https://doi.org/10.17577/ijertv8is120211

5) Андрій Креневич АЛГОРИТМИ І СТРУКТУРИ ДАНИХ. 2021 URL: https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf

6) Monte Carlo Policy Evaluation - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/monte-carlo-policy-evaluation/

7) ASP.NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/ru-ru/aspnet/overview (date of access: 06.06.2024).

8) Java [Електронний ресурс]. – Базове пояснення. URL: https://www.java.com/en/.

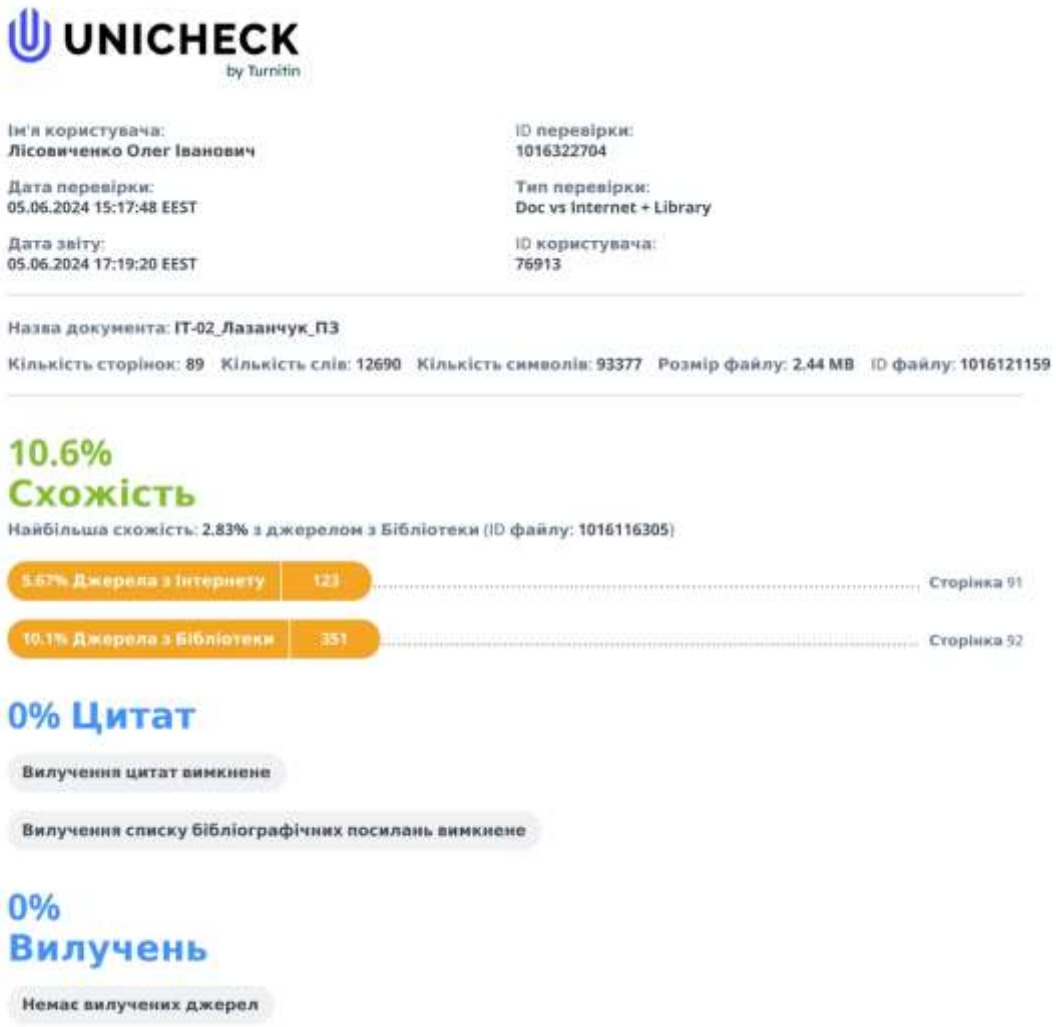
9) Index | Node.js v22.2.0 Documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/docs/latest/api/

10) Що таке Angular: шлях до кар'єри у веб-розробці. *FoxmindEd*. URL: https://foxminded.ua/angular/ (дата звернення: 06.06.2024).

11) Web Architecture [Електронний ресурс]. Загальна архітектура веб-додатку.NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures/

12) Embold | Static Code Analysis Platform. *Embold | Static Code Analysis Platform*. URL: https://embold.io.

ДОДАТКИ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016322704

Дата перевірки:
05.06.2024 15:17:48 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2024 17:19:20 EEST

ID користувача:
76913

Назва документа: ІТ-02_Лазанчук_ПЗ

Кількість сторінок: 89 Кількість слів: 12690 Кількість символів: 93377 Розмір файлу: 2.44 MB ID файлу: 1016121159

10.6%
Схожість

Найбільша схожість: 2.83% з джерелом з Бібліотеки (ID файлу: 1016116305)

5.67% Джерела з Інтернету 123 Сторінка 91

10.1% Джерела з Бібліотеки 351 Сторінка 52

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Платформа для аналізу та проєктування топологічних організацій розподілених систем

Текст програми

КПІ.ІТ-0213.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

Артем ВОЛОКИТА

Нормоконтроль:
Катерина ЛІЩУК

Виконавець:
Костянтин ЛАЗАНЧУК

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/KostyaLazanchuk/DiplomProject

Файл BinaryConnectionGenerator.cs

Реалізація функціональної вимоги «кодового перетворення»

```
using BusinessLogic.Interface;
using BusinessLogic.Service;
using Diplom.Core.Models;

namespace BusinessLogic.Graph
{
    public class BinaryConnectionGenerator
    {
        private INodeService _nodeService;
        private IEdgeService _edgeService;

        public BinaryConnectionGenerator(INodeService nodeService,
IEdgeService edgeService)
        {
            _nodeService = nodeService;
            _edgeService = edgeService;
        }

        public async Task GenerateBinaryConnections(int serverCount)
        {
            var nodes = new List<Node>();
            var binaryStrings = new List<string>();

            for (var i = 0; i < serverCount; i++)
            {
                var node = new Node
                {
                    Id = Guid.NewGuid(),
                    Name = $"Server{i + 1}",
                    CreatedOn = DateTime.UtcNow,
                    Position = i
                };
                await _nodeService.CreateNode(node);
                nodes.Add(node);

                string binary = Convert.ToString(i,
2).PadLeft((int)Math.Ceiling(Math.Log2(serverCount)), '0');
                binaryStrings.Add(binary);
            }

            for (int i = 0; i < serverCount; i++)
            {
                string binary = binaryStrings[i];

                string shift0 = ShiftBit(binary, '0');
                string shift1 = ShiftBit(binary, '1');

                int newPosition0 = Convert.ToInt32(shift0, 2);
                int newPosition1 = Convert.ToInt32(shift1, 2);


                if (newPosition0 < serverCount && newPosition0 != i)
                {
                    var startNode = nodes.First(n => n.Position == i);
```

```
                        var endNode = nodes.First(n => n.Position == newPosition0);

                        if (!await _edgeService.IsEdgeExists(startNode.Id, endNode.Id))
                        {
                            await CreateEdge(startNode.Id, endNode.Id);
                        }
                    }

                    if (newPosition1 < serverCount && newPosition1 != i)
                    {
                        var startNode = nodes.First(n => n.Position == i);
                        var endNode = nodes.First(n => n.Position == newPosition1);

                        if (!await _edgeService.IsEdgeExists(startNode.Id, endNode.Id))
                        {
                            await CreateEdge(startNode.Id, endNode.Id);
                        }
                    }
                }
            }

            private async Task CreateEdge(Guid startNodeId, Guid endNodeId)
            {
                var edge = new Edge
                {
                    Id = Guid.NewGuid(),
                    Weight = 1,
                    EndNode = endNodeId
                };
                await _edgeService.CreateRelationshipOneWay(startNodeId, endNodeId, edge.Weight);
            }

            private string ShiftBit(string binary, char newBit)
            {
                return binary.Substring(1) + newBit;
            }
        }
    }
```

Файл CartesianProduct.cs
Реалізація функціональної вимоги об’єднання графа

```
    using BusinessLogic.Interface;
    using Diplom.Core.Models;

    namespace BusinessLogic.Graph
    {
        public class CartesianProduct
        {
            private readonly ICommonService _commonService;
            private readonly IEdgeService _edgeService;
            public CartesianProduct(ICommonService commonService, IEdgeService edgeService)
            {
                _commonService = commonService;
                _edgeService = edgeService;
            }
```

```
            public async Task CartesianProductExecution(string nodenName1, string
nodeName2)
            {
                var            getNodesByNamePattern            =            await
GetNodesByNamePattern(nodenName1, nodeName2);
                await    GetCartesianProduct(getNodesByNamePattern.serverNodesOne,
getNodesByNamePattern.serverNodesTwo);
            }

            private  async  Task  GetCartesianProduct(List<Node>  list1,  List<Node>
list2)
            {
                foreach (var node1 in list1)
                {
                    foreach (var node2 in list2)
                    {
                        await    _edgeService.CreateRelationshipOneToOne(node1.Id,
node2.Id, 2, 2);
                    }
                }
            }
            private    async    Task<(List<Node>    serverNodesOne,    List<Node>
serverNodesTwo)> GetNodesByNamePattern(string nodenName1, string nodeName2)
            {
                var                allNodes                =                await
_commonService.GetAllNodesWithRelationships();

                var serverNodes = new List<Node>();
                var otherNodes = new List<Node>();

                foreach (var node in allNodes)
                {
                    if (node.Name.StartsWith(nodenName1))
                    {
                        serverNodes.Add(node);
                    }
                    else if (node.Name.StartsWith(nodeName2))
                    {
                        otherNodes.Add(node);
                    }
                }

                return (serverNodes, otherNodes);
            }
        }
    }
```

Файл CartesianProductOfGraphs.cs

Реалізація функціональної декартовий добуток

```
    using BusinessLogic.Interface;
    using Diplom.Core.Models;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

    namespace BusinessLogic.Graph
    {
        public class CartesianProductOfGraphs
        {
            private readonly INodeService _nodeService;
```

```
        private readonly IEdgeService _edgeService;

        public       CartesianProductOfGraphs(INodeService       nodeService,
IEdgeService edgeService)
        {
            _nodeService = nodeService;
            _edgeService = edgeService;
        }

        public  async  Task  GenerateCartesianProduct(List<Node>  graph1Nodes,
List<Node> graph2Nodes)
        {
            var newNodes = new Dictionary<(Guid, Guid), Node>();

            foreach (var node1 in graph1Nodes)
            {
                foreach (var node2 in graph2Nodes)
                {
                    var newNode = new Node
                    {
                        Id = Guid.NewGuid(),
                        Name = $"{node1.Name},{node2.Name}",
                        CreatedOn = DateTime.UtcNow,
                        Position = -1,
                        Edge = new List<Edge>()
                    };

                    await _nodeService.CreateNode(newNode);
                    newNodes[(node1.Id, node2.Id)] = newNode;
                }
            }

            foreach (var node1 in graph1Nodes)
            {
                foreach (var edge1 in node1.Edge)
                {
                    var   targetNode1   =   graph1Nodes.First(n   =>   n.Id   ==
edge1.EndNode.Value);

                    foreach (var node2 in graph2Nodes)
                    {
                        var newNode1 = newNodes[(node1.Id, node2.Id)];
                        var newNode2 = newNodes[(targetNode1.Id, node2.Id)];

                        var newEdge = new Edge
                        {
                            Id = Guid.NewGuid(),
                            Weight = edge1.Weight,
                            EndNode = newNode2.Id
                        };

                        await
_edgeService.CreateRelationshipOneWay(newNode1.Id, newNode2.Id, newEdge.Weight);
                    }
                }
            }

            foreach (var node2 in graph2Nodes)
            {
                foreach (var edge2 in node2.Edge)
                {
                    var   targetNode2   =   graph2Nodes.First(n   =>   n.Id   ==
edge2.EndNode.Value);

                    foreach (var node1 in graph1Nodes)
                    {
```

```
                                var newNode1 = newNodes[(node1.Id, node2.Id)];
                                var newNode2 = newNodes[(node1.Id, targetNode2.Id)];

                                var newEdge = new Edge
                                {
                                    Id = Guid.NewGuid(),
                                    Weight = edge2.Weight,
                                    EndNode = newNode2.Id
                                };

                                await
_edgeService.CreateRelationshipOneWay(newNode1.Id, newNode2.Id, newEdge.Weight);
                            }
                        }
                    }
                }
            }
        }
```

Файл FatTreeGenerator.cs
Реалізація функціоналу генерація графа «Жирного дерева»

```
        using BusinessLogic.Interface;
        using BusinessLogic.Service;
        using Diplom.Core.Models;
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Text;
        using System.Threading.Tasks;

        namespace BusinessLogic.Graph
        {
            public class FatTreeGenerator
            {
                private INodeService _nodeService;
                private IEdgeService _edgeService;
                private int _index;

                public    FatTreeGenerator(INodeService    nodeService,    IEdgeService
edgeService)
                {
                    _nodeService = nodeService;
                    _edgeService = edgeService;
                    _index = 1;
                }

                public async Task GenerateFatTree(int coreCount)
                {
                    int pods = coreCount;
                    var coreNodes = new List<Node>();
                    var aggNodes = new List<Node>();
                    var accessNodes = new List<Node>();
                    var serverNodes = new List<Node>();

                    for (int i = 0; i < coreCount; i++)
                    {
                        var coreNode = await CreateNode($"Core{i + 1}");
                        coreNodes.Add(coreNode);
                    }

                    for (int pod = 0; pod < pods; pod++)
                    {
```

```
                    var aggNode1 = await CreateNode($"Agg{_index++}");
                    var aggNode2 = await CreateNode($"Agg{_index++}");
                    aggNodes.Add(aggNode1);
                    aggNodes.Add(aggNode2);

                    var accessNode1 = await CreateNode($"Access{_index++}");
                    var accessNode2 = await CreateNode($"Access{_index++}");
                    accessNodes.Add(accessNode1);
                    accessNodes.Add(accessNode2);

                    for (int i = 0; i < coreCount; i++)
                    {
                        await CreateEdge(coreNodes[i].Id, aggNode1.Id);
                        await CreateEdge(coreNodes[i].Id, aggNode2.Id);
                    }

                    await CreateEdge(aggNode1.Id, accessNode1.Id);
                    await CreateEdge(aggNode1.Id, accessNode2.Id);
                    await CreateEdge(aggNode2.Id, accessNode1.Id);
                    await CreateEdge(aggNode2.Id, accessNode2.Id);

                    var serverNode1 = await CreateNode($"Server{_index++}");
                    var serverNode2 = await CreateNode($"Server{_index++}");
                    serverNodes.Add(serverNode1);
                    serverNodes.Add(serverNode2);
                    await CreateEdge(accessNode1.Id, serverNode1.Id);
                    await CreateEdge(accessNode1.Id, serverNode2.Id);

                    serverNode1 = await CreateNode($"Server{_index++}");
                    serverNode2 = await CreateNode($"Server{_index++}");
                    serverNodes.Add(serverNode1);
                    serverNodes.Add(serverNode2);
                    await CreateEdge(accessNode2.Id, serverNode1.Id);
                    await CreateEdge(accessNode2.Id, serverNode2.Id);
                }
            }

            private async Task<Node> CreateNode(string name)
            {
                var node = new Node
                {
                    Id = Guid.NewGuid(),
                    Name = name,
                    CreatedOn = DateTime.UtcNow
                };
                await _nodeService.CreateNode(node);
                return node;
            }

            private async Task CreateEdge(Guid startNodeId, Guid endNodeId)
            {
                var edge = new Edge
                {
                    Id = Guid.NewGuid(),
                    Weight = 1,
                    EndNode = endNodeId
                };
                await          _edgeService.CreateRelationshipOneWay(startNodeId,
endNodeId, edge.Weight);
            }
        }
    }
```

Файл NodeAndEdgeGenerator.cs

Реалізація функціоналу генерація випадкового графа

```
using BusinessLogic.Interface;
using BusinessLogic.Service;
using Diplom.Core.Models;

namespace BusinessLogic.Graph
{
    public class NodeAndEdgeGenerator
    {
        private readonly INodeService _nodeService;
        private readonly IEdgeService _edgeService;

        public NodeAndEdgeGenerator(INodeService nodeService, IEdgeService
edgeService)
        {
            _nodeService = nodeService;
            _edgeService = edgeService;
        }
        public async Task<List<Node>> CreateRandomNodesAndEdges(int
nodeCount, string nameNode)
        {
            var random = new Random();
            var nodes = new List<Node>();

            for (int i = 1; i <= nodeCount; i++)
            {
                var nodeId = Guid.NewGuid();
                var node = new Node
                {
                    Id = nodeId,
                    Name = $"{nameNode}{i}",
                    Position = i,
                    CreatedOn = DateTime.UtcNow,
                    Edge = new List<Edge>()
                };

                await CreateNode(node);
                nodes.Add(node);
            }

            for (int i = 0; i < nodeCount; i++)
            {
                int edgeCount = random.Next(1, 4);
                for (int j = 0; j < edgeCount; j++)
                {
                    var sourceNode = nodes[i];
                    var targetNode = nodes[random.Next(nodeCount)];

                    if (sourceNode.Id != targetNode.Id)
                    {
                        var edgeId = Guid.NewGuid();
                        int weight = random.Next(1, 10);

                        await CreateEdgeOneWay(sourceNode.Id, targetNode.Id,
weight);

                        sourceNode.Edge.Add(new Edge
                        {
```

```
                                Id = edgeId,
                                Weight = weight,
                                EndNode = targetNode.Id
                            });
                        }
                    }
                }

                return nodes;
            }

            private async Task CreateNode(Node node)
            {
                await _nodeService.CreateNode(node);
            }

            private async Task CreateEdgeOneWay(Guid sourceNodeId, Guid
targerNodeId, int weight)
            {
                await _edgeService.CreateRelationshipOneWay(sourceNodeId,
targerNodeId, weight);
            }
        }
    }
```

Файл RootedProduct.cs

Реалізація функціональної кореневий добуток

```
    using BusinessLogic.Interface;
    using BusinessLogic.Service;
    using Diplom.Core.Models;

    namespace BusinessLogic.Graph
    {
        public class RootedProduct
        {
            private readonly INodeService _nodeService;
            private readonly ICommonService _commonService;
            private readonly IEdgeService _edgeService;
            private static readonly SemaphoreSlim semaphore = new
SemaphoreSlim(1, 1);
            public RootedProduct(INodeService nodeService, ICommonService
commonService, IEdgeService edgeService)
            {
                _nodeService = nodeService;
                _commonService = commonService;
                _edgeService = edgeService;
            }
            public async Task RootedProductExecution(string baseGraphName, string
rootGraphName)
            {
                var starterRootGraph = await
_commonService.GetNodesByPattern(rootGraphName);
                var baseNodes = await
_commonService.GetNodesByPattern(baseGraphName);
                var index = 1;

                foreach (var baseNode in baseNodes)
                {
                    var rootNode = await
_commonService.GetNodeByPatternWithMinIndex(rootGraphName);
```

```
                    await          _commonService.CreateRootedNodeCopies(baseNode,
rootNode, index);
                    index++;
                }
                foreach (var starteRoot in starterRootGraph)
                {
                    await _nodeService.DeleteNode(starteRoot.Id);
                }
            }
        }
    }
```

Файл AStarAlgorithm.cs

Реалізація функціональної вимоги алгоритм A*

```
    using BusinessLogic.Interface;
    using DataAccess.Repositories;
    using Diplom.Core.Models;
    using Neo4j.Driver;

    namespace BusinessLogic.Algorithms
    {
        public class AStarAlgorithm
        {
            private readonly INodeService _nodeService;

            public   AStarAlgorithm(ICommonService   commonService,   INodeService
nodeService)
            {
                _nodeService = nodeService;
            }

            public  async  Task<List<Node>>  FindPathByAStar(Guid  startId,  Guid
goalId, Func<Node, Node, double> heuristic)
            {
                var openSet = new HashSet<Node>();
                var cameFrom = new Dictionary<Node, Node>();

                var gScore = new Dictionary<Node, double>();
                var fScore = new Dictionary<Node, double>();

                var start = await GetNodeById(startId);
                var goal = await GetNodeById(goalId);

                openSet.Add(start);
                gScore[start] = 0;
                fScore[start] = heuristic(start, goal);

                while (openSet.Count > 0)
                {
                    var current = GetNodeWithLowestFScore(openSet, fScore);

                    if (current.Id == goal.Id)
                    {
                        return ReconstructPath(cameFrom, current);
                    }

                    openSet.Remove(current);

                    foreach (var edge in current.Edge)
                    {
```

```
                        var targetNode = await GetNodeById(edge.EndNode.Value);
                        var tentativeGScore = gScore[current] + edge.Weight;

                        if (!gScore.ContainsKey(targetNode) || tentativeGScore < gScore[targetNode])
                        {
                            cameFrom[targetNode] = current;
                            gScore[targetNode] = tentativeGScore;
                            fScore[targetNode] = gScore[targetNode] + heuristic(targetNode, goal);

                            if (!openSet.Contains(targetNode))
                            {
                                openSet.Add(targetNode);
                            }
                        }
                    }
                }

                return new List<Node>();
            }

            private Node GetNodeWithLowestFScore(HashSet<Node> openSet, Dictionary<Node, double> fScore)
            {
                Node lowest = null;
                double lowestScore = double.PositiveInfinity;

                foreach (var node in openSet)
                {
                    if (fScore.TryGetValue(node, out double score) && score < lowestScore)
                    {
                        lowestScore = score;
                        lowest = node;
                    }
                }

                return lowest;
            }

            private List<Node> ReconstructPath(Dictionary<Node, Node> cameFrom, Node current)
            {
                var totalPath = new List<Node> { current };

                while (cameFrom.ContainsKey(current))
                {
                    current = cameFrom[current];
                    totalPath.Add(current);
                }

                totalPath.Reverse();
                return totalPath;
            }

            private async Task<Node> GetNodeById(Guid id)
            {
                return await _nodeService.GetNodeById(id);
            }
        }
    }
```

Файл CheckAnotherWay.cs

Перевірка альтернативного шляху

```
using BusinessLogic.Interface;
using BusinessLogic.Service;
using Diplom.Core.Models;

namespace BusinessLogic.Algorithms
{
    public class CheckAnotherWay
    {
        private readonly INodeService _nodeService;
        private readonly ICommonService _commonService;
        private readonly IDijkstraAlgorithm _dijkstraAlgorithm;

        public CheckAnotherWay(INodeService nodeService, ICommonService
commonService, IDijkstraAlgorithm dijkstraAlgorithm)
        {
            _nodeService = nodeService;
            _commonService = commonService;
            _dijkstraAlgorithm = dijkstraAlgorithm;
        }

        public async Task<List<Node>>
CheckAnotherWayAfterDijkstraExecute(Guid startId, Guid goalId)
        {
            var allNodesList = await
_commonService.GetAllNodesWithRelationships();
            var dijkstraAlgorithmList = await
_dijkstraAlgorithm.FindPathByDijkstra(startId, goalId);
            var elementsToRemove =
dijkstraAlgorithmList.Skip(1).Take(dijkstraAlgorithmList.Count - 2).Select(n =>
n.Id);
            allNodesList.RemoveAll(item =>
elementsToRemove.Contains(item.Id));
            return await DijkstraAlgorithmLogic(startId, goalId,
allNodesList);
        }

        private async Task<List<Node>> DijkstraAlgorithmLogic(Guid startId,
Guid goalId, List<Node> nodesList)
        {
            var startNode = await GetNodeById(startId);
            var goalNode = await GetNodeById(goalId);

            if (startNode == null || goalNode == null)
            {
                return new List<Node>();
            }

            var distances = new Dictionary<Guid, int>();
            var previous = new Dictionary<Guid, Guid?>();
            var nodes = new Dictionary<Guid, Node>();

            foreach (var node in nodesList)
            {
                distances[node.Id] = int.MaxValue;
                previous[node.Id] = null;
                nodes[node.Id] = node;
            }

            distances[startNode.Id] = 0;

            var priorityQueue = new List<Node>(nodes.Values.OrderBy(n =>
distances[n.Id]));
```

```
                while (priorityQueue.Count > 0)
                {
                    var current = priorityQueue.First();
                    priorityQueue.Remove(current);

                    if (current.Id == goalNode.Id)
                    {
                        var path = await ReconstructPath(previous, goalNode.Id);
                        return path;
                    }

                    if (current.Edge != null)
                    {
                        foreach (var edge in current.Edge)
                        {
                            if (nodes.ContainsKey(edge.EndNode.Value))
                            {
                                var neighbor = nodes[edge.EndNode.Value];
                                var alt = distances[current.Id] + edge.Weight;

                                if (alt < distances[neighbor.Id])
                                {
                                    distances[neighbor.Id] = alt;
                                    previous[neighbor.Id] = current.Id;
                                }
                            }
                        }

                        priorityQueue = priorityQueue.OrderBy(n => distances[n.Id]).ToList();
                    }
                }

                return new List<Node>();
            }

            private async Task<Node> GetNodeById(Guid id)
            {
                return await _nodeService.GetNodeById(id);
            }

            private async Task<List<Node>> ReconstructPath(Dictionary<Guid, Guid?> previous, Guid goalId)
            {
                var path = new List<Node>();
                var currentNodeId = goalId;

                while (currentNodeId != Guid.Empty && previous.ContainsKey(currentNodeId))
                {
                    var currentNode = await GetNodeById(currentNodeId);
                    path.Add(currentNode);
                    if (previous[currentNodeId].HasValue)
                    {
                        currentNodeId = previous[currentNodeId].Value;
                    }
                    else
                    {
                        break;
                    }
                }

                path.Reverse();
                return path;
            }
        }
```

```
}
```

Файл DijkstraAlgorithm.cs

Реалізація кодової вимоги «Алгоритм Дейкстри»

```
using BusinessLogic.Interface;
using BusinessLogic.Service;
using Diplom.Core.Models;
using Neo4j.Driver;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BusinessLogic.Algorithms
{
    public class DijkstraAlgorithm
    {
        private readonly INodeService _nodeService;
        private readonly ICommonService _commonService;

        public DijkstraAlgorithm(INodeService nodeService, ICommonService
commonService)
        {
            _nodeService = nodeService;
            _commonService = commonService;
        }

        public async Task<List<Node>> FindPathByDijkstra(Guid startId, Guid
goalId)
        {
            var startNode = await GetNodeById(startId);
            var goalNode = await GetNodeById(goalId);

            if (startNode == null || goalNode == null)
            {
                return new List<Node>();
            }

            var distances = new Dictionary<Guid, int>();
            var previous = new Dictionary<Guid, Guid?>();
            var nodes = new Dictionary<Guid, Node>();

            foreach (var node in await GetAllNodesWithRelationships())
            {
                distances[node.Id] = int.MaxValue;
                previous[node.Id] = null;
                nodes[node.Id] = node;
            }

            distances[startNode.Id] = 0;

            var priorityQueue = new List<Node>(nodes.Values.OrderBy(n =>
distances[n.Id]));

            while (priorityQueue.Count > 0)
            {
                var current = priorityQueue.First();
                priorityQueue.Remove(current);

                if (current.Id == goalNode.Id)
                {
```

```
                        var path = await ReconstructPath(previous, goalNode.Id);
                        return path;
                    }

                    if (current.Edge != null)
                    {
                        foreach (var edge in current.Edge)
                        {
                            if (nodes.ContainsKey(edge.EndNode.Value))
                            {
                                var neighbor = nodes[edge.EndNode.Value];
                                var alt = distances[current.Id] + edge.Weight;

                                if (alt < distances[neighbor.Id])
                                {
                                    distances[neighbor.Id] = alt;
                                    previous[neighbor.Id] = current.Id;
                                }
                            }
                        }

                        priorityQueue = priorityQueue.OrderBy(n => distances[n.Id]).ToList();
                    }
                }

                return new List<Node>();
            }


            private async Task<Node> GetNodeById(Guid id)
            {
                return await _nodeService.GetNodeById(id);
            }

            private async Task<List<Node>> GetAllNodesWithRelationships()
            {
                return await _commonService.GetAllNodesWithRelationships();
            }

            private async Task<List<Node>> ReconstructPath(Dictionary<Guid, Guid?> previous, Guid goalId)
            {
                var path = new List<Node>();
                var currentNodeId = goalId;

                while (currentNodeId != Guid.Empty && previous.ContainsKey(currentNodeId))
                {
                    var currentNode = await GetNodeById(currentNodeId);
                    path.Add(currentNode);
                    if (previous[currentNodeId].HasValue)
                    {
                        currentNodeId = previous[currentNodeId].Value;
                    }
                    else
                    {
                        break;
                    }
                }

                path.Reverse();
                return path;
            }
        }
    }
```

Файл MonteCarloSimulation.cs

Реалізація кодової вимоги «Алгоритму Монте Карло»

```
using Diplom.Core.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BusinessLogic.Algorithms
{
    public class MonteCarloSimulation
    {
        private readonly Random _random = new Random();
        private readonly Dictionary<Guid, double> _failureProbabilities;

        public MonteCarloSimulation(Dictionary<Guid, double>
failureProbabilities)
        {
            _failureProbabilities = failureProbabilities;
        }

        public double EvaluateNetworkReliability(List<Node> nodes, int
iterations)
        {
            int successfulIterations = 0;

            for (int i = 0; i < iterations; i++)
            {
                if (IsNetworkOperational(nodes))
                {
                    successfulIterations++;
                }
            }

            return (double)successfulIterations / iterations;
        }

        private bool IsNetworkOperational(List<Node> nodes)
        {
            var nodesCopy = CreateNetworkCopyWithFailures(nodes);
            return IsNetworkConnected(nodesCopy);
        }

        private List<Node> CreateNetworkCopyWithFailures(List<Node> nodes)
        {
            var nodesCopy = new List<Node>();

            foreach (var node in nodes)
            {
                var nodeCopy = new Node
                {
                    Id = node.Id,
                    Name = node.Name,
                    CreatedOn = node.CreatedOn,
                    Position = node.Position,
                    Edge = new List<Edge>()
                };

                foreach (var edge in node.Edge)
                {
```

```
                        if (_failureProbabilities.TryGetValue(edge.Id, out double
failureProbability))
                        {
                            if (_random.NextDouble() > failureProbability)
                            {
                                var edgeCopy = new Edge
                                {
                                    Id = edge.Id,
                                    Weight = edge.Weight,
                                    EndNode = edge.EndNode
                                };

                                nodeCopy.Edge.Add(edgeCopy);
                            }
                        }
                    }

                    nodesCopy.Add(nodeCopy);
                }

                return nodesCopy;
            }

            private bool IsNetworkConnected(List<Node> nodes)
            {
                if (nodes.Count == 0)
                    return false;

                var visited = new HashSet<Guid>();
                var components = new List<HashSet<Guid>>();

                foreach (var node in nodes)
                {
                    if (!visited.Contains(node.Id))
                    {
                        var component = new HashSet<Guid>();
                        DFS(node, nodes, component, visited);
                        components.Add(component);
                    }
                }

                return components.Count == 1;
            }

            private   void   DFS(Node   node,   List<Node>   nodes,   HashSet<Guid>
component, HashSet<Guid> visited)
            {
                visited.Add(node.Id);
                component.Add(node.Id);

                foreach (var edge in node.Edge)
                {
                    var   neighbor   =   nodes.FirstOrDefault(n   =>   n.Id   ==
edge.EndNode);
                    if (neighbor != null && !visited.Contains(neighbor.Id))
                    {
                        DFS(neighbor, nodes, component, visited);
                    }
                }
            }
        }
    }
```

Файл NodeRepository.cs

Реалізація кодової вимоги «Робото з базою даних та робото з Вузлами»

```
using Diplom.Core.Models;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Neo4j.Driver;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace DataAccess.Repositories
{
    public class NodeRepository
    {
        private readonly IDriver _driver;

        public NodeRepository(IDriver driver)
        {
            _driver = driver;
        }

        public async Task<Node> GetNodeById(Guid id)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node) WHERE n.id = $id OPTIONAL MATCH (n)-[r]->(m)
RETURN n, collect({ id: m.id, weight: r.weight, end: m.id }) as relationships",
                        new { id = id.ToString() });
                    var record = await reader.SingleAsync();
```

```
                    var nodeProperties = record["n"].As<INode>().Properties;
                    var relationships =
record["relationships"].As<List<Dictionary<string, object>>>();

                    var nodeObject = new Node
                    {
                        Id = Guid.Parse(nodeProperties["id"].As<string>()),
                        Name = nodeProperties["name"].As<string>(),
                        Position = Convert.ToInt32(nodeProperties["position"]),
                        CreatedOn =
DateTime.Parse(nodeProperties["createdOn"].As<string>()),
                        Color = nodeProperties.ContainsKey("color") ?
nodeProperties["color"].As<string>() : string.Empty,
                        Edge = new List<Edge>()
                    };

                    foreach (var relationshipData in relationships)
                    {
                        if (relationshipData["id"] != null)
                        {
                            var relationship = new Edge
                            {
                                Id = Guid.Parse(relationshipData["id"].ToString()),
                                Weight =
Convert.ToInt32(relationshipData["weight"]),
                                EndNode =
Guid.Parse(relationshipData["end"].ToString())
                            };
                            nodeObject.Edge.Add(relationship);
                        }
                    }
```

```
                    return nodeObject;
                });
                return result;
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<Node> GetNodeByName(string name)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node { name: $name }) OPTIONAL MATCH (n)-[r]->(m)
RETURN n, collect({ id: r.id, weight: r.weight, endNode: m.id }) as edges",
                        new { name });

                    var record = await reader.SingleAsync();

                    var nodeProperties = record["n"].As<INode>().Properties;
                    var edgesData = record["edges"].As<List<Dictionary<string,
object>>>();

                    var nodeObject = new Node
                    {
                        Id = nodeProperties.ContainsKey("id") ?
Guid.Parse(nodeProperties["id"].As<string>()) : Guid.Empty,
```

```
                        Name = nodeProperties.ContainsKey("name") ?
nodeProperties["name"].As<string>() : string.Empty,
                        Position = nodeProperties.ContainsKey("position") ?
int.Parse(nodeProperties["position"].As<string>()) : 0,
                        CreatedOn = nodeProperties.ContainsKey("createdOn") ?
DateTime.Parse(nodeProperties["createdOn"].As<string>()) : DateTime.MinValue,
                        Edge = new List<Edge>()
                    };

                    if (edgesData != null)
                    {
                        foreach (var edgeData in edgesData)
                        {
                            var edge = new Edge
                            {
                                Id = edgeData.ContainsKey("id") && edgeData["id"] !=
null ? Guid.Parse(edgeData["id"].ToString()) : Guid.Empty,
                                Weight = edgeData.ContainsKey("weight") &&
edgeData["weight"] != null ? Convert.ToInt32(edgeData["weight"]) : 0,
                                EndNode = edgeData.ContainsKey("endNode") &&
edgeData["endNode"] != null ? Guid.Parse(edgeData["endNode"].ToString()) :
(Guid?)null
                            };
                            nodeObject.Edge.Add(edge);
                        }
                    }

                    return nodeObject;
                });

                return result;
            }
            finally
```

```
            {
                await session.CloseAsync();
            }
        }

        public async Task<Node> CreateNode(Node node)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteWriteAsync(async tx =>
                {
                    var createdOn = DateTime.Now.ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");

                    var reader = await tx.RunAsync(
                        "CREATE (n:Node {id: $id, name: $name, position: $position,
createdOn: $createdOn, color: $color}) RETURN n",
                        new { id = node.Id.ToString(), name = node.Name, position =
node.Position, createdOn, color = node.Color });
                    var record = await reader.SingleAsync();
                    var createdNode = record["n"].As<INode>();
                    return new Node
                    {
                        Id = Guid.Parse(createdNode.Properties["id"].As<string>()),
                        Name = createdNode.Properties["name"].As<string>(),
                        Position = createdNode.Properties["position"].As<int>(),
                        CreatedOn = DateTime.Now,
                        //Color = createdNode.Properties["color"].As<string>(),
                        Edge = new List<Edge>()
                    };
                });
                return result;
```

```
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<Node> UpdateNode(Guid id, string newNodeName)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteWriteAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node {id: $id}) SET n.name = $newName RETURN n",
                        new { id = id.ToString(), newName = newNodeName });
                    var record = await reader.SingleAsync();
                    var updatedNode = record["n"].As<INode>();
                    return new Node
                    {
                        Id = Guid.Parse(updatedNode.Properties["id"].As<string>()),
                        Name = updatedNode.Properties["name"].As<string>(),
                        Position =
int.Parse(updatedNode.Properties["position"].As<string>()),
                        CreatedOn =
DateTime.Parse(updatedNode.Properties["createdOn"].As<string>()),
                        Color = updatedNode.Properties.ContainsKey("color") ?
updatedNode.Properties["color"].As<string>() : string.Empty,
                        Edge = new List<Edge>()
                    };
                });
                return result;
```

```
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<bool> DeleteNode(Guid id)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteWriteAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node {id: $id}) DETACH DELETE n RETURN COUNT(n) as
deletedCount",
                        new { id = id.ToString() });

                    var record = await reader.SingleAsync();
                    var deletedCount = record["deletedCount"].As<int>();

                    return deletedCount > 0;
                });

                return result;
            }
            finally
            {
                await session.CloseAsync();
            }
        }
```

```
        public async Task<List<Node>> GetNeighbors(Guid id)
        {
            var nodes = new List<Node>();

            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var cursor = await tx.RunAsync(
                        "MATCH (n:Node { id: $id })-[r]->(m:Node) RETURN m",
                        new { id = id.ToString() });

                    return await cursor.ToListAsync();
                });

                foreach (var record in result)
                {
                    var mNode = record["m"].As<INode>();
                    var node = new Node
                    {
                        Id = Guid.Parse(mNode.Properties["id"].As<string>()),
                        Name = mNode.Properties["name"].As<string>(),
                        CreatedOn =
DateTime.Parse(mNode.Properties["createdOn"].As<string>()),
                        Position =
int.Parse(mNode.Properties["position"].As<string>()),
                        Color = mNode.Properties.ContainsKey("color") ?
mNode.Properties["color"].As<string>() : string.Empty,
                        Edge = new List<Edge>()
                    };
                    nodes.Add(node);
                }
```

```
            }
            finally
            {
                await session.CloseAsync();
            }

            return nodes;
        }

        public async Task<int> GetDistance(Guid currentNodeId, Guid neighborNodeId)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var cursor = await tx.RunAsync(
                        "MATCH (current:Node { id: $currentNodeId })-[r]-
>(neighbor:Node { id: $neighborNodeId }) RETURN r.weight AS weight",
                        new { currentNodeId = currentNodeId.ToString(),
neighborNodeId = neighborNodeId.ToString() });

                    if (await cursor.FetchAsync())
                    {
                        return cursor.Current["weight"].As<int>();
                    }
                    return int.MaxValue;
                });

                return result;
            }
            finally
            {
```

```
                await session.CloseAsync();
            }
        }

        public async Task<List<Node>> GetAllNodes()
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync("MATCH (n:Node) RETURN n");

                    var nodes = new List<Node>();

                    while (await reader.FetchAsync())
                    {
                        var nodeProperties =
reader.Current["n"].As<INode>().Properties;

                        nodes.Add(new Node
                        {
                            Id = Guid.Parse(nodeProperties["id"].As<string>()),
                            Name = nodeProperties["name"].As<string>(),
                            Position = nodeProperties.ContainsKey("position") ?
int.Parse(nodeProperties["position"].As<string>()) : 0,
                            CreatedOn = nodeProperties.ContainsKey("createdOn") ?
DateTime.Parse(nodeProperties["createdOn"].As<string>()) : DateTime.MinValue,
                            Color = nodeProperties.ContainsKey("color") ?
nodeProperties["color"].As<string>() : string.Empty,
                            Edge = new List<Edge>()
                        });
                    }
```

```
                return nodes;
            });

            return result;
        }
        finally
        {
            await session.CloseAsync();
        }
    }

    public async Task<int> CountNodes()
    {
        var session = _driver.AsyncSession();
        try
        {
            var result = await session.ExecuteReadAsync(async tx =>
            {
                var reader = await tx.RunAsync("MATCH (n:Node) RETURN COUNT(n)
as nodeCount");
                var record = await reader.SingleAsync();
                return record["nodeCount"].As<int>();
            });
            return result;
        }
        finally
        {
            await session.CloseAsync();
        }
    }

    public async Task<int> CountNodesByName(string name)
```

```
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node) WHERE n.name STARTS WITH $name RETURN
count(n) as nodeCount",
                        new { name });

                    var record = await reader.SingleAsync();
                    return record["nodeCount"].As<int>();
                });

                return result;
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<Guid> GetNodeIdByName(string name)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var cursor = await tx.RunAsync(
                        "MATCH (n:Node { name: $name }) RETURN n.id AS id",
                        new { name });
```

```
            var records = await cursor.ToListAsync();

            if (records.Any())
            {
                return Guid.Parse(records.First()["id"].As<string>());
            }

            return Guid.Empty;
        });

        return result;
    }
    finally
    {
        await session.CloseAsync();
    }
}

public async Task SetNodeColor(Guid nodeId, string color)
{
    var session = _driver.AsyncSession();
    try
    {
        await session.ExecuteWriteAsync(async tx =>
        {
            await tx.RunAsync(
                "MATCH (n:Node { id: $nodeId }) SET n.color = $color",
                new { nodeId = nodeId.ToString(), color });
        });
    }
    finally
    {
```

```
                await session.CloseAsync();
            }
        }

        public async Task<List<Node>> GetNodesByColorAsync(string color)
        {
            var session = _driver.AsyncSession();
            var query = @"
            MATCH (n:Node {Color: $color})
            RETURN n.Id as Id, n.Name as Name, n.Position as Position, n.CreatedOn
as CreatedOn, n.Color as Color";

            var result = await session.RunAsync(query, new { color });
            var nodes = new List<Node>();

            await result.ForEachAsync(record =>
            {
                nodes.Add(new Node
                {
                    Id = record["Id"].As<Guid>(),
                    Name = record["Name"].As<string>(),
                    Position = record["Position"].As<int>(),
                    CreatedOn = record["CreatedOn"].As<DateTime>(),
                    Color = record["Color"].As<string>(),
                    Edge = new List<Edge>()
                });
            });

            return nodes;
        }

        public async Task<Node> GetNodeByPosition(int position)
        {
```

```
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node { position: $position }) RETURN n",
                        new { position });

                    var record = await reader.SingleAsync();

                    var nodeProperties = record["n"].As<INode>().Properties;

                    var nodeObject = new Node
                    {
                        Id = Guid.Parse(nodeProperties["id"].As<string>()),
                        Name = nodeProperties["name"].As<string>(),
                        Position = nodeProperties["position"].As<int>(),
                        CreatedOn =
DateTime.Parse(nodeProperties["createdOn"].As<string>()),
                        Edge = new List<Edge>()
                    };

                    var edgeReader = await tx.RunAsync(
                        "MATCH (n:Node { position: $position })-[r]->(m) RETURN r,
m",
                        new { position });

                    await edgeReader.ForEachAsync(edgeRecord =>
                    {
                        var edge = new Edge
                        {
```

```
                            Id =
Guid.Parse(edgeRecord["r"].As<IRelationship>().Properties["id"].As<string>()),
                            Weight =
edgeRecord["r"].As<IRelationship>().Properties["weight"].As<int>(),
                            EndNode =
Guid.Parse(edgeRecord["m"].As<INode>().Properties["id"].As<string>())
                        };
                        nodeObject.Edge.Add(edge);
                    });

                    return nodeObject;
                });

                return result;
            }
            finally
            {
                await session.CloseAsync();
            }
        }
    }
}
```

Файл CommonRepository.cs

Реалізація кодової вимоги «Загальні методи роботи з графом»

```
using Diplom.Core.Models;
using Neo4j.Driver;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DataAccess.Repositories
{
    public class CommonRepository
    {
        private readonly IDriver _driver;
```

```
        public CommonRepository(IDriver driver)
        {
            _driver = driver;
        }

        public async Task DeleteAllData()
        {
            var session = _driver.AsyncSession();
            try
            {
                await session.ExecuteWriteAsync(async tx =>
                {
                    await tx.RunAsync("MATCH (n) DETACH DELETE n");
                });
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<List<Node>> GetAllNodesWithRelationships()
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node) OPTIONAL MATCH (n)-[r]->(m) RETURN n,
collect({ id: m.id, weight: r.weight, end: m.id }) as relationships");

                    var nodes = new List<Node>();

                    await reader.ForEachAsync(record =>
                    {
                        var                  nodeProperties                  =
record["n"].As<INode>().Properties;
                        var                   relationships                  =
record["relationships"].As<List<Dictionary<string, object>>>();

                        var nodeObject = new Node
                        {
                            Id                                                 =
Guid.Parse(nodeProperties["id"].As<string>()),
                            Name = nodeProperties["name"].As<string>(),
                            Position                                           =
int.Parse(nodeProperties["position"].As<string>()),
                            CreatedOn                                          =
DateTime.Parse(nodeProperties["createdOn"].As<string>()),
                            Color   =   nodeProperties.ContainsKey("color")   ?
nodeProperties["color"].As<string>() : string.Empty,
                            Edge = new List<Edge>()
                        };

                        if (relationships != null)
                        {
                            foreach (var relationshipData in relationships)
                            {
                                if   (relationshipData["id"]   !=   null   &&
relationshipData["weight"] != null && relationshipData["end"] != null)
                                {
                                    var relationship = new Edge
                                    {
                                        Id                                     =
Guid.Parse(relationshipData["id"].ToString()),
```

```
                                        Weight                                  =
Convert.ToInt32(relationshipData["weight"]),
                                        EndNode                                 =
Guid.Parse(relationshipData["end"].ToString())
                                    };
                                    nodeObject.Edge.Add(relationship);
                                }
                            }
                        }

                        nodes.Add(nodeObject);
                    });

                    return nodes;
                });

                return result;
            }
            finally
            {
                await session.CloseAsync();
            }
        }

        public async Task<List<Node>> GetNodesByPattern(string name)
        {
            var session = _driver.AsyncSession();
            try
            {
                var result = await session.ExecuteReadAsync(async tx =>
                {
                    var reader = await tx.RunAsync(
                        "MATCH (n:Node) WHERE n.name STARTS WITH $name " +
                        "OPTIONAL MATCH (n)-[r]->(m) " +
                        "RETURN  n,  collect({  id:  r.id,  weight:  r.weight,
endNode: m.id }) as edges",
                        new { name });

                    var nodes = new List<Node>();

                    await reader.ForEachAsync(record =>
                    {
                        var                 nodeProperties                  =
record["n"].As<INode>().Properties;
                        var                   edgesData                     =
record["edges"].As<List<Dictionary<string, object>>>();

                        var nodeObject = new Node
                        {
                            Id                                              =
Guid.Parse(nodeProperties["id"].As<string>()),
                            Name = nodeProperties["name"].As<string>(),
                            Position                                        =
int.Parse(nodeProperties["position"].As<string>()),
                            CreatedOn                                       =
DateTime.Parse(nodeProperties["createdOn"].As<string>()),
                            Color   =   nodeProperties.ContainsKey("color")   ?
nodeProperties["color"].As<string>() : string.Empty,
                            Edge = new List<Edge>()
                        };

                        foreach (var edgeData in edgesData)
                        {
                            if (edgeData["id"] != null && edgeData["weight"]
!= null && edgeData["endNode"] != null)
                            {
```

```
                                    var edge = new Edge
                                    {
                                        Id                                      =
Guid.Parse(edgeData["id"].ToString()),
                                        Weight                                  =
Convert.ToInt32(edgeData["weight"]),
                                        EndNode                                 =
Guid.Parse(edgeData["endNode"].ToString())
                                    };
                                    nodeObject.Edge.Add(edge);
                                }
                            }

                            nodes.Add(nodeObject);
                        });

                        return nodes;
                    });

                    return result;
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public     async     Task<Node>     GetNodeByPatternWithMinIndex(string
namePattern)
            {
                var session = _driver.AsyncSession();
                try
                {
                    var result = await session.ExecuteReadAsync(async tx =>
                    {
                        var reader = await tx.RunAsync(
                            "MATCH (n:Node) WHERE n.name STARTS WITH $namePattern
RETURN n",
                            new { namePattern });

                        var nodes = new List<Node>();

                        await reader.ForEachAsync(record =>
                        {
                            var                  nodeProperties                 =
record["n"].As<INode>().Properties;
                            var nodeObject = new Node
                            {
                                Id                                              =
Guid.Parse(nodeProperties["id"].As<string>()),
                                Name = nodeProperties["name"].As<string>(),
                                Position                                        =
int.Parse(nodeProperties["position"].As<string>()),
                                CreatedOn                                       =
DateTime.Parse(nodeProperties["createdOn"].As<string>()),
                                Color   =   nodeProperties.ContainsKey("color")   ?
nodeProperties["color"].As<string>() : string.Empty,
                                Edge = new List<Edge>()
                            };

                            nodes.Add(nodeObject);
                        });

                        return nodes;
                    });
```

```
                    if (result.Any())
                    {
                        var nodeWithMinIndex = result
                            .Where(n => int.TryParse(n.Name.Substring(namePattern.Length), out _))
                            .OrderBy(n => int.Parse(n.Name.Substring(namePattern.Length)))
                            .FirstOrDefault();

                        return nodeWithMinIndex;
                    }

                    return null;
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public async Task CreateRootedNodeCopies(Node baseNode, Node rootNode, int index)
            {
                var session = _driver.AsyncSession();
                try
                {
                    await session.ExecuteWriteAsync(async tx =>
                    {
                        var newRootNodeId = Guid.NewGuid().ToString();
                        await tx.RunAsync(
                            "CREATE (n:Node { id: $newRootNodeId, name: $name, position: $position, createdOn: $createdOn, color: $color })",
                            new { newRootNodeId, name = $"{rootNode.Name}v{index}", position = rootNode.Position, createdOn = rootNode.CreatedOn.ToString("o"), color = rootNode.Color }
                            );

                        foreach (var edge in rootNode.Edge)
                        {
                            var newEdgeId = Guid.NewGuid().ToString();
                            await tx.RunAsync(
                                "MATCH (source:Node { id: $newRootNodeId }), (target:Node { id: $endNode }) " +
                                "CREATE (source)-[:CONNECTION { id: $newEdgeId, weight: $weight }]->(target)",
                                new { newRootNodeId, endNode = edge.EndNode.ToString(), newEdgeId, weight = edge.Weight }
                                );
                        }

                        var connectionEdgeId = Guid.NewGuid().ToString();
                        await tx.RunAsync(
                            "MATCH (source:Node { id: $baseNodeId }), (target:Node { id: $newRootNodeId }) " +
                            "CREATE (source)-[:CONNECTION { id: $connectionEdgeId, weight: 1 }]->(target)",
                            new { baseNodeId = baseNode.Id.ToString(), newRootNodeId, connectionEdgeId }
                            );
                    });
                }
                finally
                {
                    await session.CloseAsync();
                }
            }
```

```
        }
    }
```

Файл **EdgeRepository.cs**

Реалізація кодової вимоги «Методи для роботи з ребрами»

```
    using Diplom.Core.Models;
    using Neo4j.Driver;
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;

    namespace DataAccess.Repositories
    {
        public class EdgeRepository
        {
            private readonly IDriver _driver;

            public EdgeRepository(IDriver driver)
            {
                _driver = driver;
            }

            public async Task CreateEdgeOneWay(Guid sourceNodeId, Guid
targetNodeId, int weight)
            {
                var session = _driver.AsyncSession();
                try
                {
                    await session.ExecuteWriteAsync(async tx =>
                    {
                        var edgeId = Guid.NewGuid();
                        var query = "MATCH (source:Node { id: $sourceNodeId }),
(target:Node { id: $targetNodeId }) " +
                                    "CREATE (source)-[rel:CONNECTION { id:
$edgeId, weight: $weight }]->(target)";
                        await tx.RunAsync(query, new { sourceNodeId =
sourceNodeId.ToString(), targetNodeId = targetNodeId.ToString(), edgeId =
edgeId.ToString(), weight });
                    });
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public async Task CreateEdgeOneToOne(Guid sourceNodeId, Guid
targetNodeId, int weight1, int weight2)
            {
                var session = _driver.AsyncSession();
                try
                {
                    await session.ExecuteWriteAsync(async tx =>
                    {
                        var edgeId1 = Guid.NewGuid();
                        var edgeId2 = Guid.NewGuid();

                        await tx.RunAsync(
                            "MATCH (source:Node { id: $sourceNodeId }) " +
                            "MATCH (target:Node { id: $targetNodeId }) " +
```

```
                            "CREATE (source)-[:CONNECTION { id: $edgeId1, weight:
$weight1 }]->(target), " +
                            "(target)-[:CONNECTION   {   id:   $edgeId2,   weight:
$weight2 }]->(source)",
                            new
                            {
                                sourceNodeId = sourceNodeId.ToString(),
                                targetNodeId = targetNodeId.ToString(),
                                edgeId1 = edgeId1.ToString(),
                                edgeId2 = edgeId2.ToString(),
                                weight1,
                                weight2
                            });
                    });
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public async Task UpdateNodeWithRelationships(Node node)
            {
                var session = _driver.AsyncSession();
                try
                {
                    await session.ExecuteWriteAsync(async tx =>
                    {
                        await tx.RunAsync(
                            "MATCH (n:Node { id: $id }) " +
                            "SET   n.name   =   $name,   n.position   =   $position,
n.createdOn = $createdOn, n.color = $color",
                            new  {  id  =  node.Id.ToString(),  name  =  node.Name,
position   =   node.Position,   createdOn   =   node.CreatedOn.ToString("o"),   color   =
node.Color });

                        await tx.RunAsync(
                            "MATCH (n:Node { id: $id })-[r]-() DELETE r",
                            new { id = node.Id.ToString() });

                        foreach (var relationship in node.Edge)
                        {
                            await tx.RunAsync(
                                "MATCH (start:Node { id: $startId }), (end:Node {
id: $endId }) " +
                                "CREATE   (start)-[:CONNECTION   {   id:   $edgeId,
weight: $weight }]->(end)",
                                new  {  startId  =  node.Id.ToString(),  endId  =
relationship.EndNode.ToString(),  edgeId  =  relationship.Id.ToString(),  weight  =
relationship.Weight });
                        }
                    });
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public async Task<List<Edge>> GetEdgesByNodeId(Guid nodeId)
            {
                var session = _driver.AsyncSession();
                var result = await session.RunAsync(
                    "MATCH  (n:Node  {  id:  $nodeId  })-[r:CONNECTION]->(m)  RETURN
r",
                    new { nodeId = nodeId.ToString() });
```

```
                var edges = new List<Edge>();

                await foreach (var record in result)
                {
                    var relationship = record["r"].As<IRelationship>();
                    edges.Add(new Edge
                    {
                        Id                                                  =
Guid.Parse(relationship.Properties["id"].As<string>()),
                        Weight = relationship.Properties["weight"].As<int>(),
                        EndNode = Guid.Parse(relationship.EndNodeId.ToString())
                    });
                }

                return edges;
            }

            public async Task<bool> UpdateEdgeWeight(Guid edgeId, int newWeight)
            {
                var session = _driver.AsyncSession();
                try
                {
                    var result = await session.ExecuteWriteAsync(async tx =>
                    {
                        var reader = await tx.RunAsync(
                            "MATCH ()-[r {id: $edgeId }]->() SET r.weight =
$newWeight RETURN COUNT(r) as updatedCount",
                            new { edgeId = edgeId.ToString(), newWeight });

                        var record = await reader.SingleAsync();
                        var updatedCount = record["updatedCount"].As<int>();

                        return updatedCount > 0;
                    });

                    return result;
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public async Task<bool> DeleteEdge(Guid edgeId)
            {
                var session = _driver.AsyncSession();
                try
                {
                    var result = await session.ExecuteWriteAsync(async tx =>
                    {
                        var reader = await tx.RunAsync(
                            "MATCH ()-[r {id: $edgeId}]->() DELETE r RETURN
COUNT(r) as deletedCount",
                            new { edgeId = edgeId.ToString() });

                        var record = await reader.SingleAsync();
                        var deletedCount = record["deletedCount"].As<int>();

                        return deletedCount > 0;
                    });

                    return result;
                }
                finally
                {
                    await session.CloseAsync();
```

```
                }
            }

            public async Task<int> CountEdges()
            {
                var session = _driver.AsyncSession();
                try
                {
                    var result = await session.ExecuteReadAsync(async tx =>
                    {
                        var reader = await tx.RunAsync("MATCH ()-[r:CONNECTION]-
>() RETURN COUNT(r) as edgeCount");
                        var record = await reader.SingleAsync();
                        return record["edgeCount"].As<int>();
                    });
                    return result;
                }
                finally
                {
                    await session.CloseAsync();
                }
            }

            public   async   Task<bool>   IsEdgeExists(Guid   startNodeId,   Guid
endNodeId)
            {
                var session = _driver.AsyncSession();
                try
                {
                    var result = await session.ExecuteReadAsync(async tx =>
                    {
                        var reader = await tx.RunAsync(
                            "MATCH   (start:Node   {   id:   $startNodeId   })-[r]-
>(end:Node { id: $endNodeId }) RETURN COUNT(r) > 0 AS exists",
                            new { startNodeId = startNodeId.ToString(), endNodeId
= endNodeId.ToString() });

                        var record = await reader.SingleAsync();
                        return record["exists"].As<bool>();
                    });

                    return result;
                }
                finally
                {
                    await session.CloseAsync();
                }
            }
        }
    }
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Платформа для аналізу та проєктування топологічних організацій розподілених систем

Програма та методика тестування

КПІ.ІТ-0213.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Артем ВОЛОКИТА

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Костянтин ЛАЗАНЧУК

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є вебзастосунок для проведення консультацій з психологом у браузерах Chrome, Edge, Mozilla.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox);

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування. Детальний опис засобів та процедури тестування знаходиться у розділі 3 пояснювальної записки. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування інтерфейсу користувача;

– тестування зручності використання;

– тестування у різних браузерів.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Платформа для аналізу та проєктування топологічних організацій розподілених систем

Керівництво користувача

КПІ.ІТ-0213.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Артем ВОЛОКИТА

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Костянтин ЛАЗАНЧУК

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

“DiplomApp” – це платформа для аналізу та проєктування топологічних організацій розподілених систем. Користувачі можуть створювати та редагувати топологічні організації, взаємодіяти з різними компонентами системи та переглядати аналітичні дані. Додаток працює у сучасних веб-браузерах, таких як Google Chrome, Mozilla Firefox, Opera та Safari. Для роботи програми вимагається постійне підключення до інтернету.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішного функціонування даної платформи для роботи з топологічними графами, потрібно урахувати необхідні апаратні вимоги, які забезпечать стабільну та ефективну роботу системи.

Рекомендовані апаратні вимоги:

– intel Core 2duo або AMD Athlon з мінімальною частотою від 1 ГГц;

– оперативна пам’ять: Мінімум 4 ГБ для стабільної роботи;

– місце на диску: Не менше ніж 1 ГБ вільного місця для тимчасових файлів та кешу;

– мережеве підключення: Інтернет з’єднання зі швидкістю не менше 5 Мбіт/с для стабільної комунікації з застосунком.

2.2 Завантаження застосунку

Для користування застосунком потрібно відкрити у браузері URL-адресу застосунку. Завантаження сторонніх файлів чи програм не потребується.

2.3 Перевірка коректної роботи

Після відкриття URL браузером має відобразитися базова сторінка.

3 ВИКОНАННЯ ПРОГРАМИ

Користувач знаходиться на базовій сторінці, яка відображає увесь функціонал продукту на рисунку 4.1.

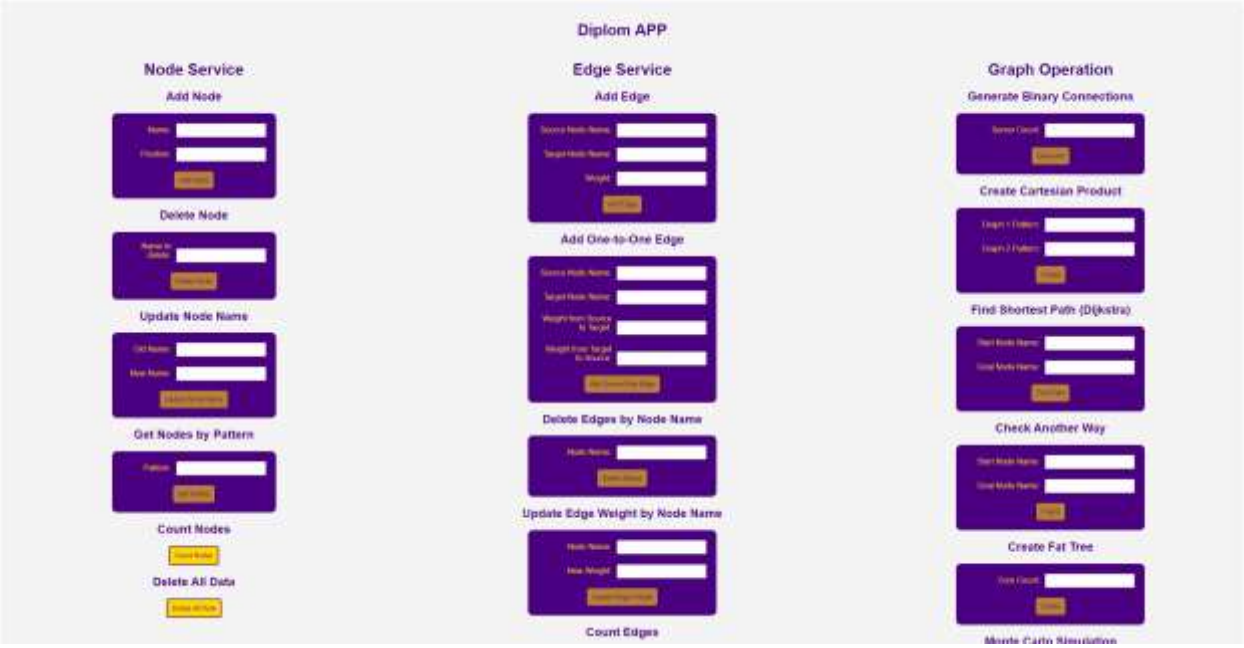


Рисунок 4.1 – Базова сторінка додатку

Користувач хоче створити вузол у систему, для цього потрібно заповнити панель «Add Node». Результат заповнення на рисунку 4.2

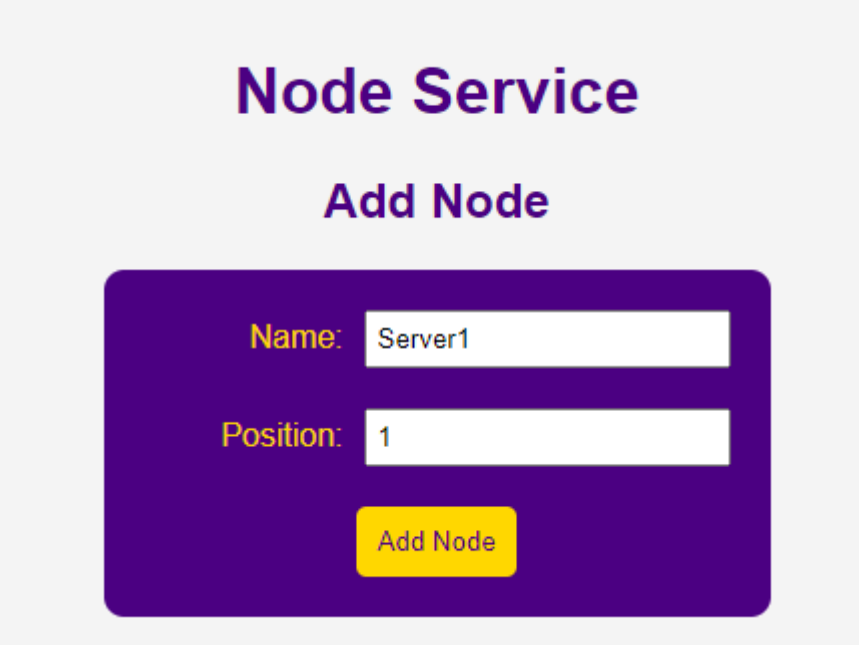


Рисунок 4.2 – Заповнення полів для створення вузла

Результат створеного вузла на рисунку 4.3

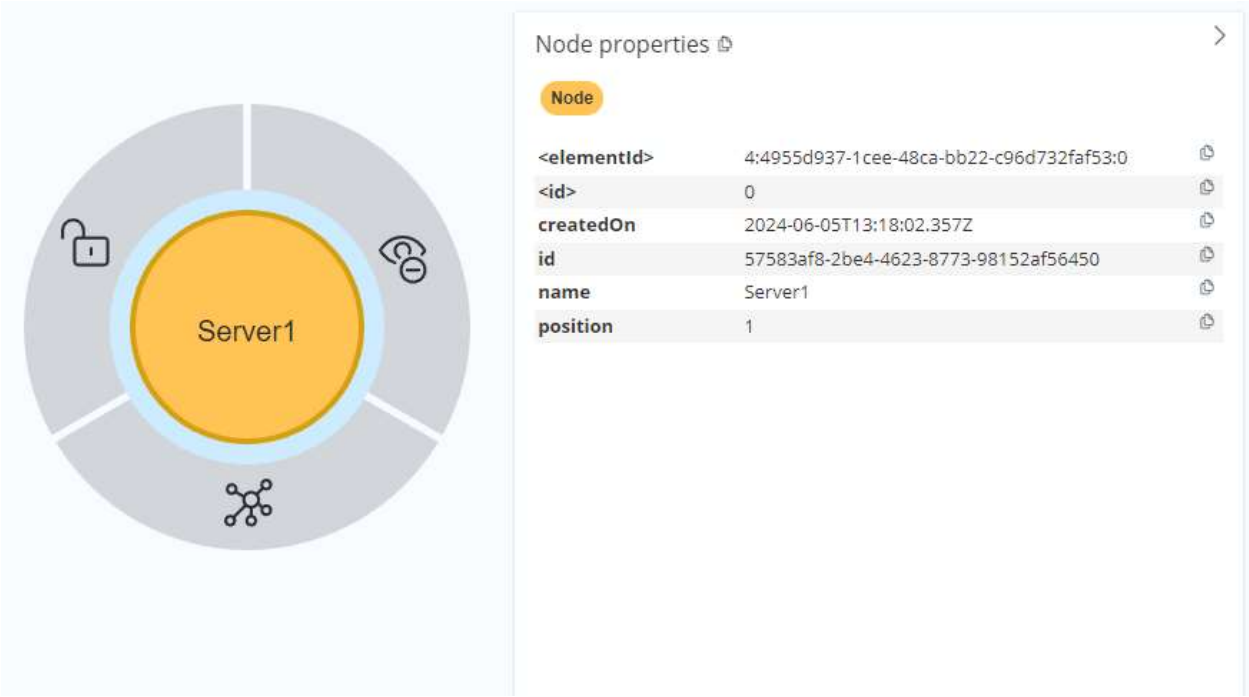


Рисунок 4.3 – Створений вузол у базі даних

Тепер ми можемо при помилці оновити даний вузол, а сами змінити йому і’мяю Для цього нам потрібно заповнити панель «Update Node Name». Результат на рисунку 4.4

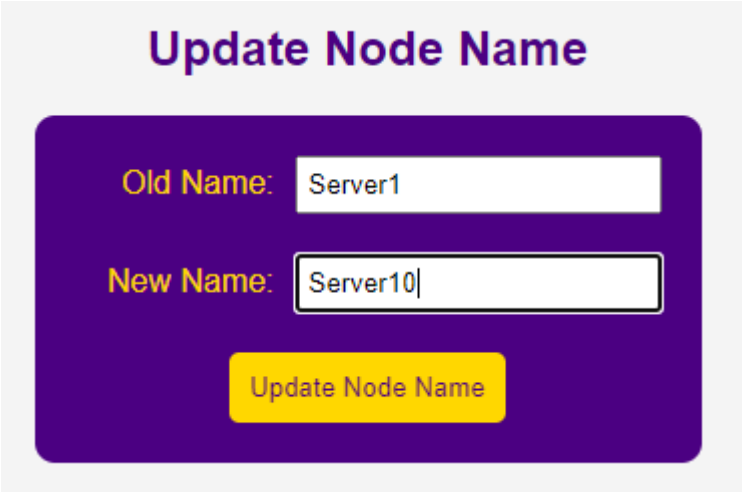


Рисунок 4.4 – Заповнення полів для оновлення назви вузла

Результат оновленого вузла на рисунку 4.5

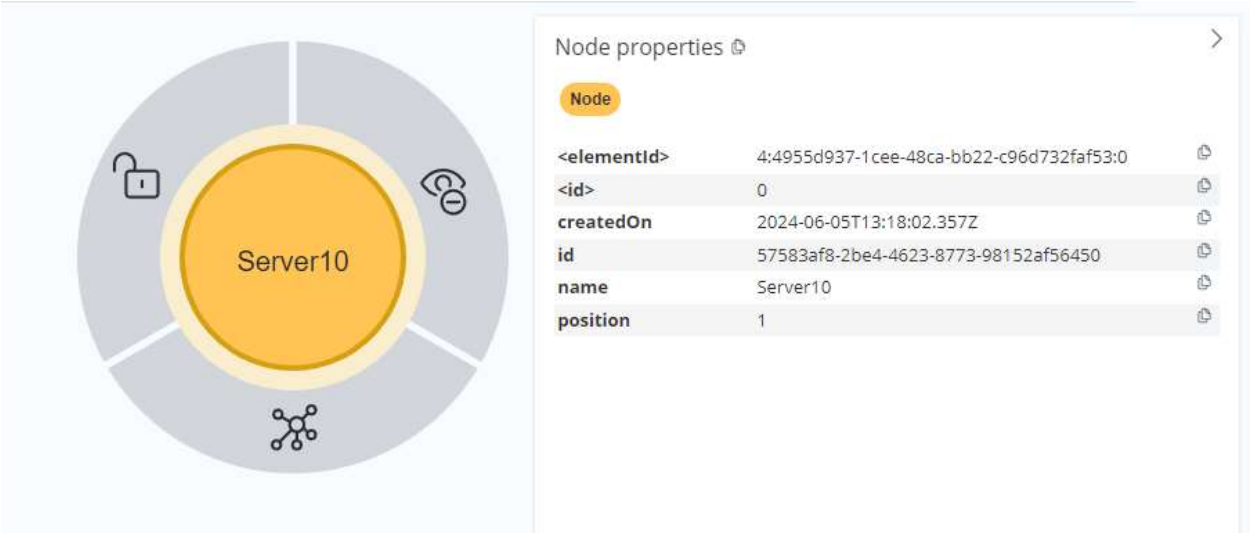


Рисунок 4.5 – Оновлений вузол у базі даних

Користувач хоче створити перше ребро, для цього йому потрібно створити ще один вузол. Умовно це буде з ім’ям Server2. Для створення ребра між вузлами, нам потрібно перейти до панельки «Add Edge». Користувач повинен заповнити усі поля. Результат на рисунку 4.6

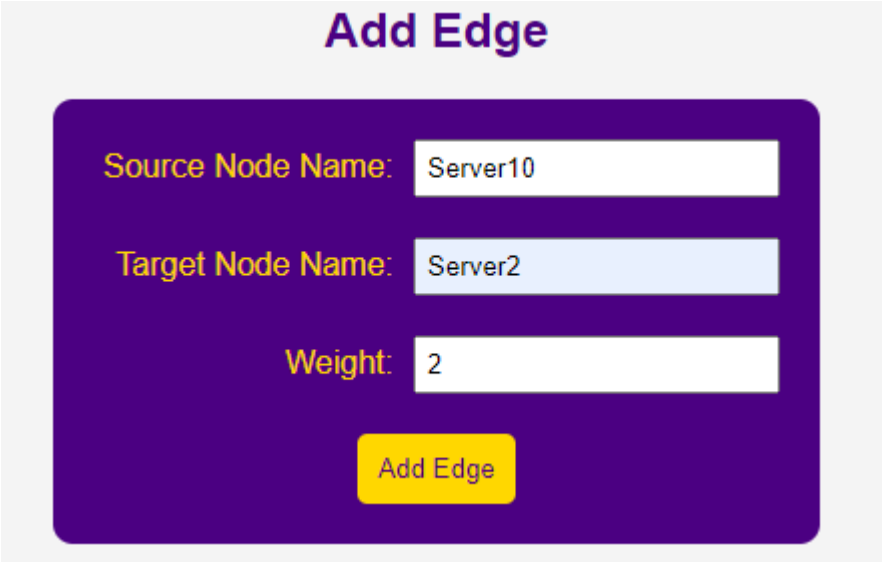


Рисунок 4.6 – Заповнення полів для створення ребра

Результат роботи на рисунку 4.7

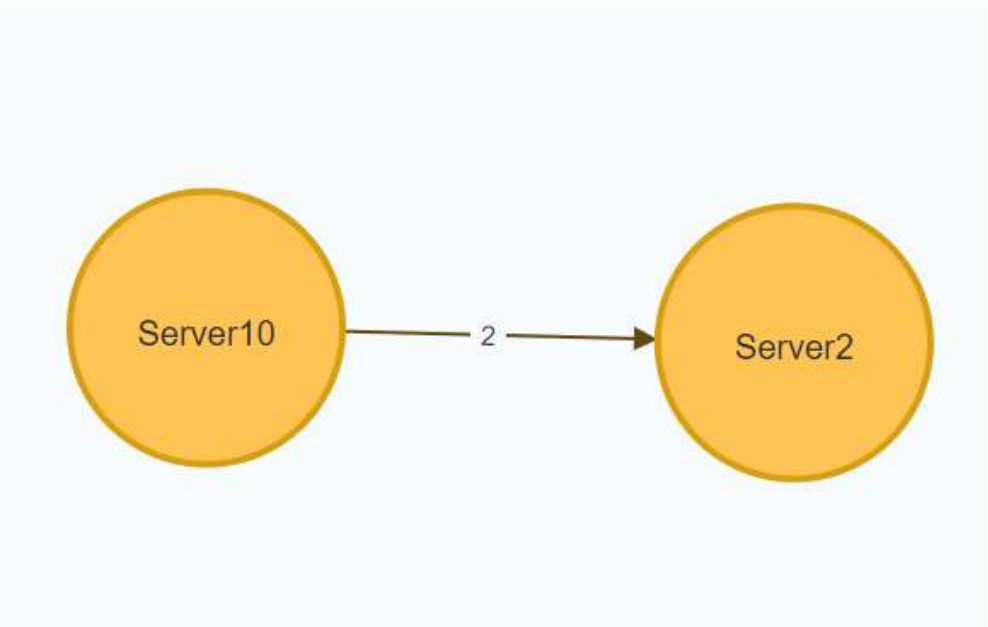


Рисунок 4.7 – Створення ребра між вузлами

Даний вузол дає можливість передавати данні з Server10 на Server2, але не у іншу сторону. Для цього нам потрібно реалізувати інший варіант об'єднання. Для цього створимо вузол з назвою Server3 та перейдемо до панелі «Add One-to-One Edge». Результат на рисунку 4.8

Рисунок 4.8 – Заповнення полів для створення ребра

Результат створеного ребра між Server10 та Server3 на рисунку 4.9

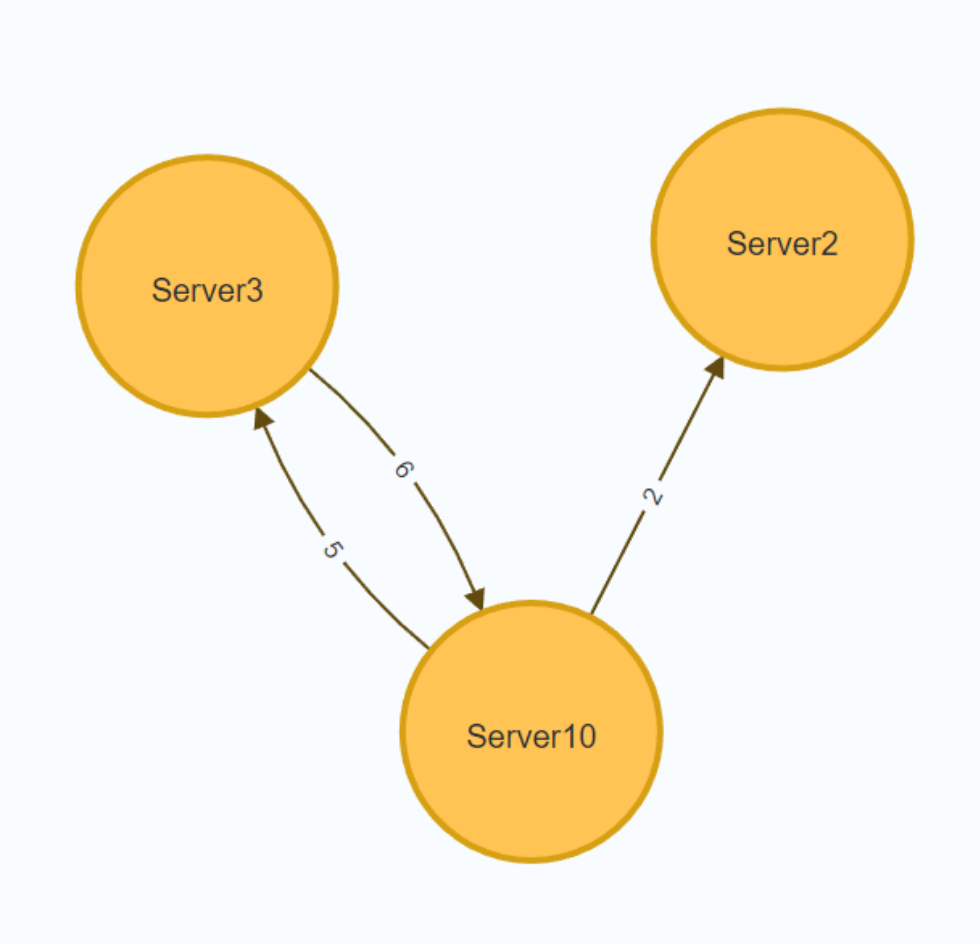


Рисунок 4.9 – Створення ребра між вузлами

Давайте оновимо ребра які ідуть із Server10 їм вагу на 10. Для цього нам потрібно перейти до панелі «Update Edge Weight by Node Name». Результат заповнення панелі на рисунку 4.10

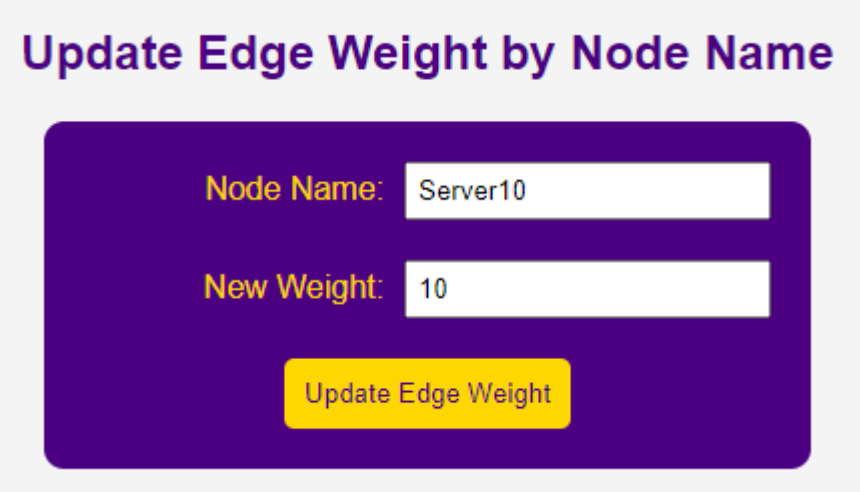


Рисунок 4.10 – Заповнення панелі «Update Edge Weight by Node Name»

Результат оновлених ребр на рисунку 4.11

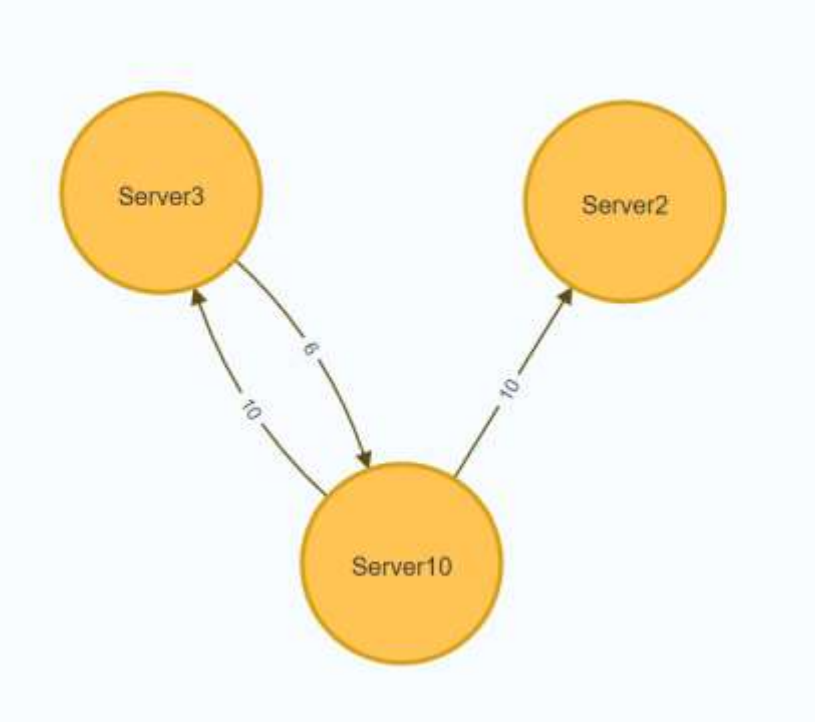


Рисунок 4.11 – Оновлені ребра

Давайте видалимо ребро, яке іде з Server10 до Server3 для цього нам потрібно перейти на панель «Delete Edged by Node Name». Результат на рисунку 4.12

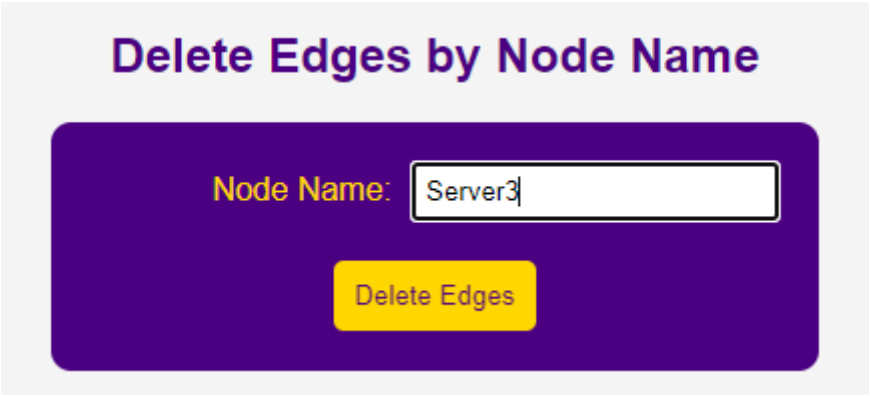


Рисунок 4.12 – Заповнення панелі «Delete Edged by Node Name»

Результат роботи на рисунку 4.13

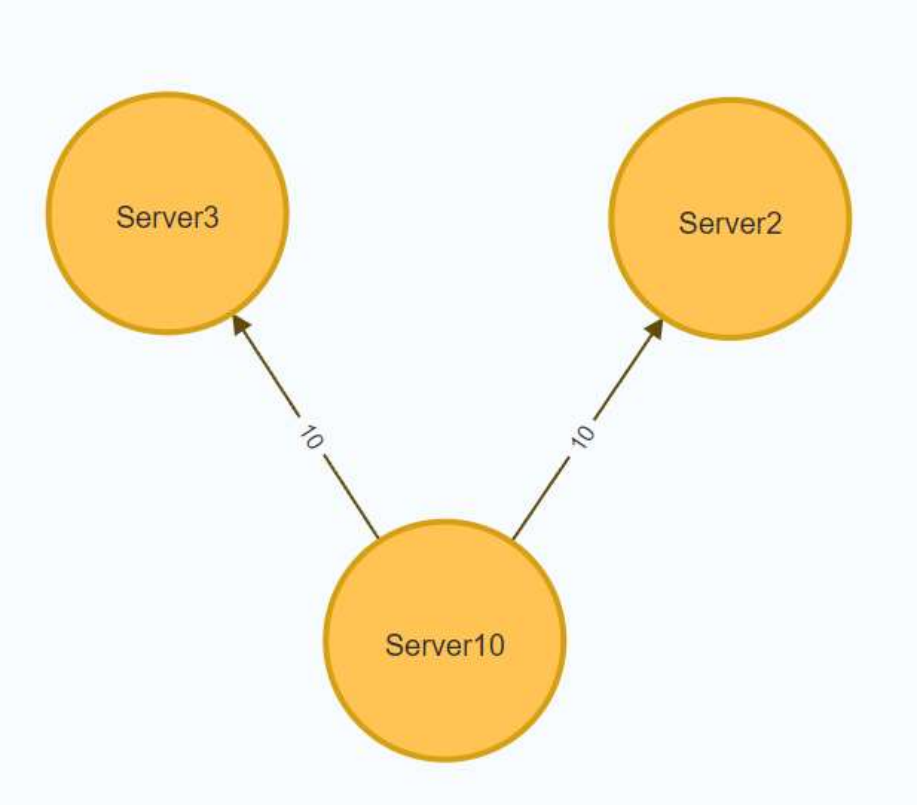


Рисунок 4.13 – Оновлений граф

Наступним кроком давайте згенеруємо випадковий граф на 100 вузлів. Для цього переходимо до панельки «Create Random Nodes and Edges» та заповнюємо поля. Результат на рисунку 4.14

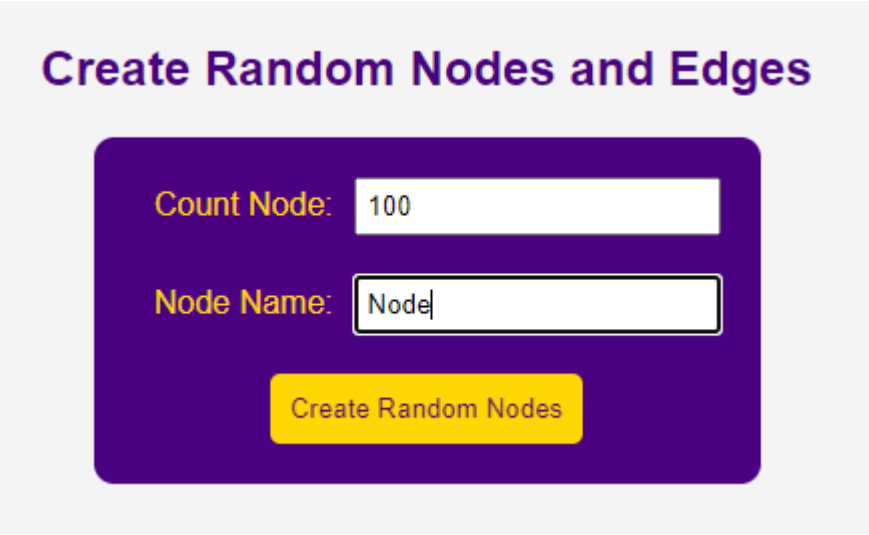


Рисунок 4.14 – Заповнення полів у панелі «Create Random Nodes and Edges»

Результат роботи на рисунку 4.15

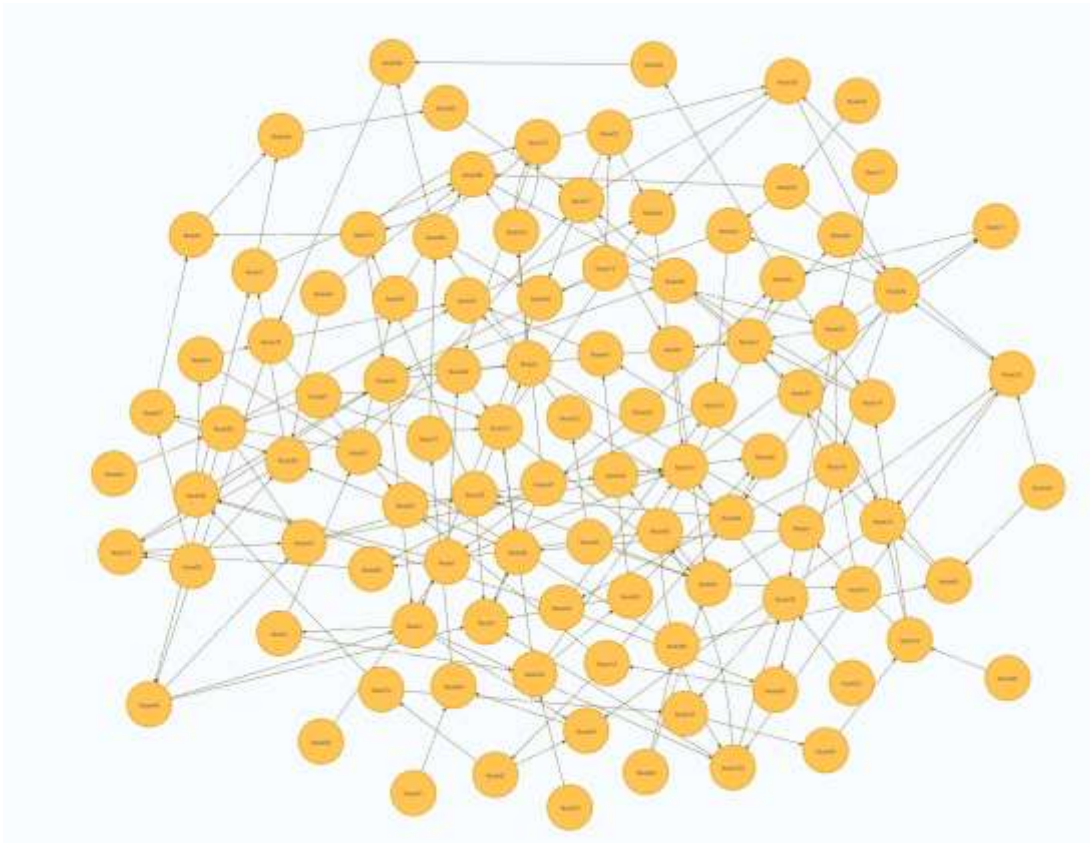


Рисунок 4.15 – Випадково графова система

Давайте створимо граф на основі кодових перетворень. Для цього нам потрібно перейти до панельки «Generate Binary Connection» Результат на рисунку 4.16

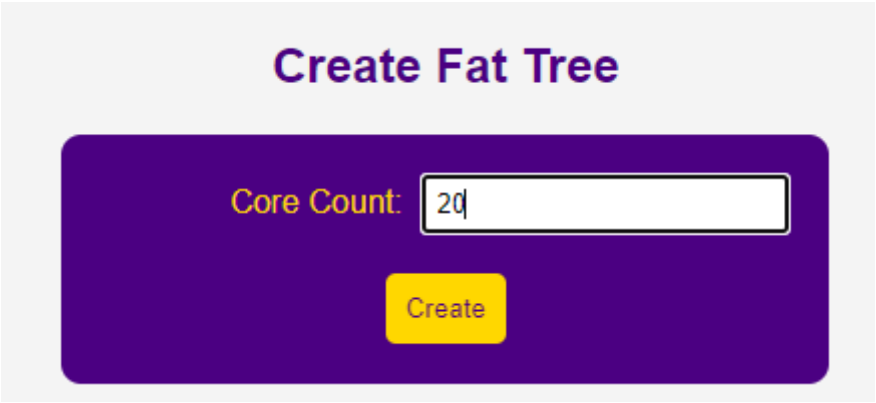


Рисунок 4.16 – Створення топологічного графа на основі кодових перетворень

Результат роботи на рисунку 4.17

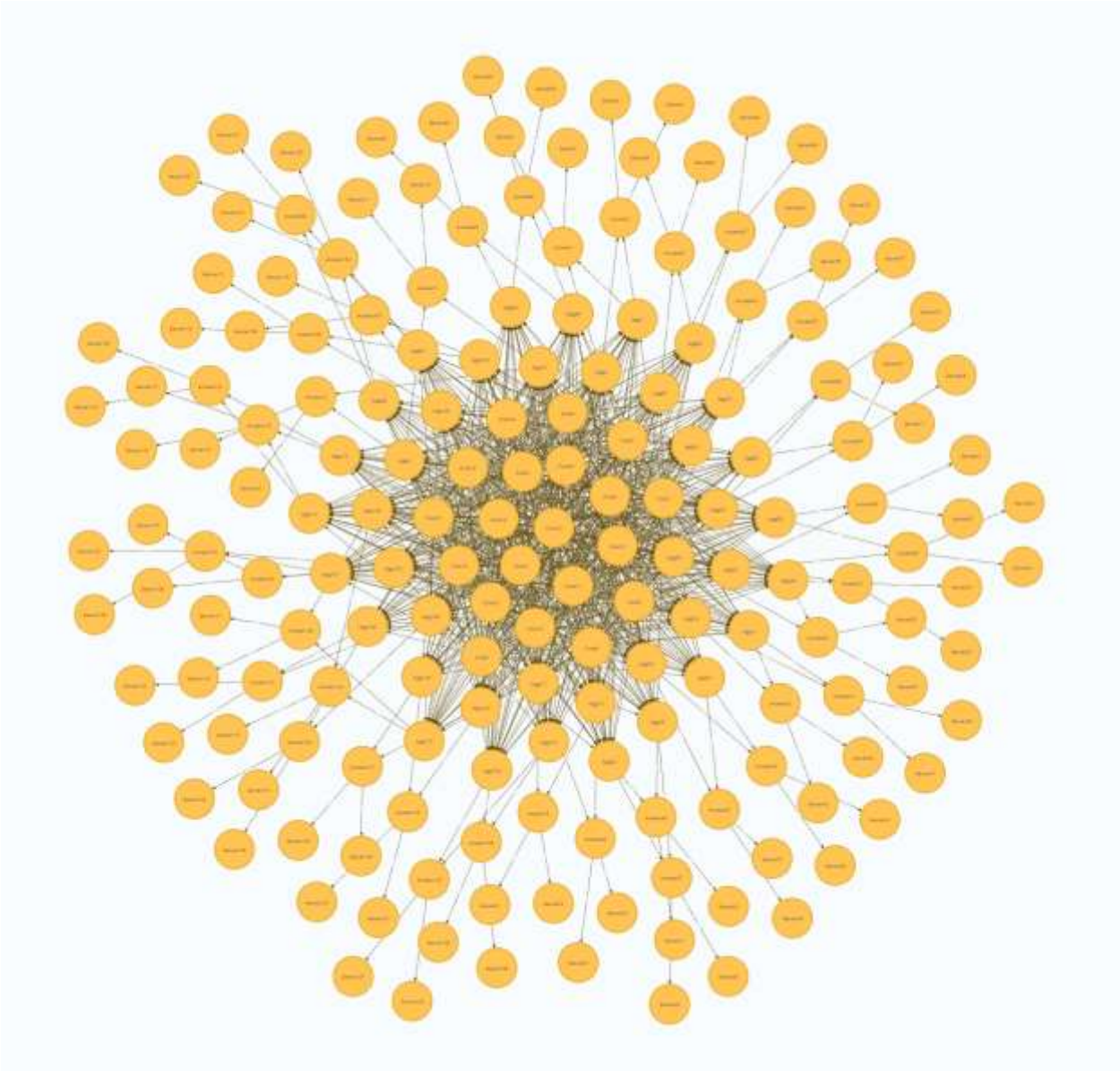


Рисунок 4.17 – Створена мережа «Жирне дерево»

Давайте об’єднаємо два графа, для цього нам потрібно очистити дану мережу та створити дві графові системи. Потім перейдемо до панелі «Create Cartesian Product» та заповнюємо поля. Результат на рисунку 4.18

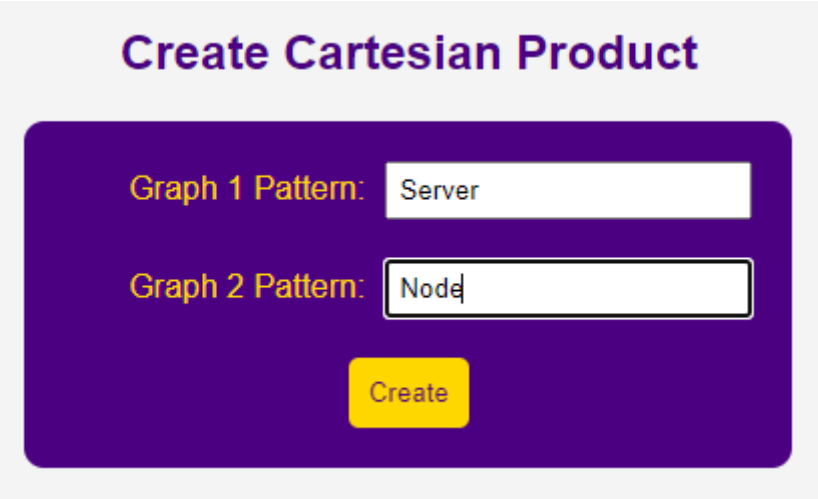


Рисунок 4.18 – Заповнення панелі «Create Cartesian Product»

Результат на рисунку 4.19

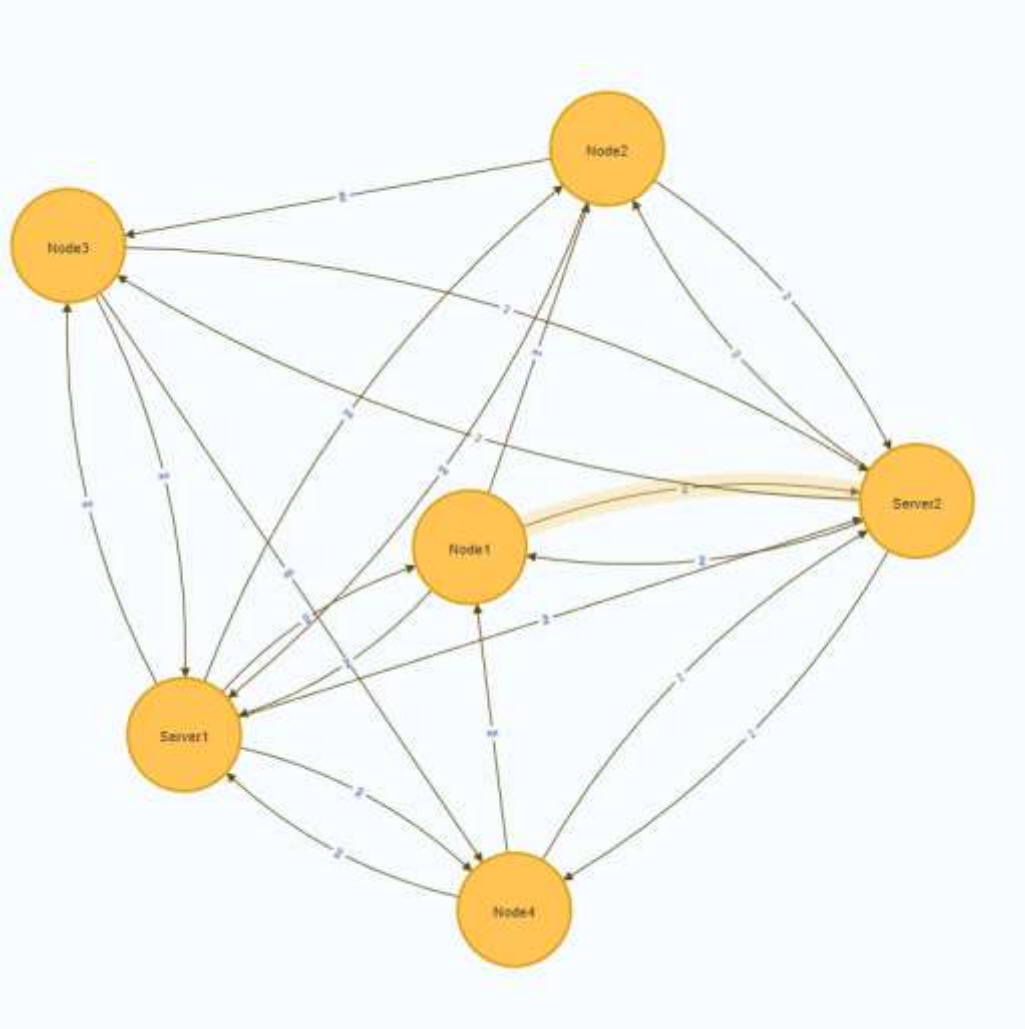


Рисунок 4.19 – Об’єднання двух систем

Давайте створимо нову мережу на 100 вузлів та спробуємо знайти шлях найкоротший шлях від одного вузла до іншого. Для цього нам потрібно перейти до панелі «Find Shortest Path (Dijkstra)». Результат на рисунку 4.20

Рисунок 4.20 – Заповнення панелі «Find Shortest Path (Dijkstra)»

Результат на рисунку 4.21

Path Server10 Server21 Server40 Server18 Server50 Server29 Server26

Рисунок 4.21 Результат роботи пошуку шляху Декйстри

Наступним кроком давайте перевіримо, який буде шлях якщо найкоротший був заблоковано. Для цього переходимо до панелі «Check Another Way» та заповнюємо поля. Результат на рисунку 4.22

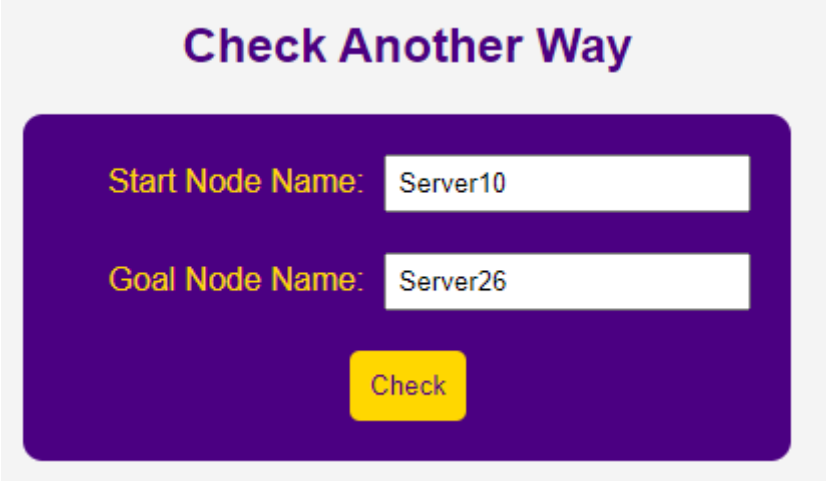


Рисунок 4.22 – Заповнення панелі «Check Another Way»

Результат роботи на рисунку 4.23

Path Server26

Рисунок 4.23 Результат пошуку альтернативного шляху

Далі давайте перевіримо на скільки дана мережа може коректо працювати, якщо деякі вузли не будуть працювати. Для цього нам потрібно перейти до панелі «Monte Carlo Simulation» та заповнити поля. Результат на рисунку 4.24

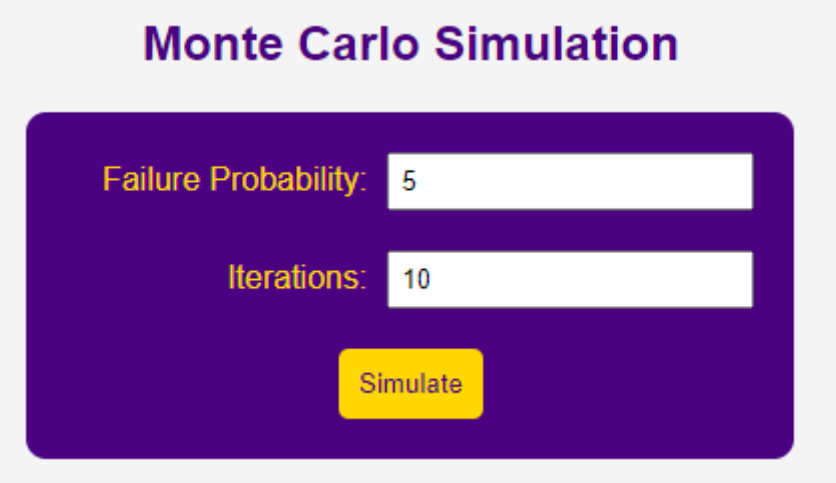


Рисунок 4.24 – Заповнення панелі «Monte Carlo Simulation»

Результат роботи на рисунку 4.25

Network reliability: 85.00%

Рисунок 4.25 результат перевірки мережи на робото здатність

Наступним кроком давайте перевіримо шлях від одного вузла до іншого використовуючи інший алгоритм. Для цього перейдемо до панелі «Find Shortest Path (A*)» та заповнимо у ньому поля. Результат на рисунку 4.25

Рисунок 4.25 – Заповнення панелі «Find Shortest Path (A*)»

Результат роботи на рисунку 4.26

Path Server10 Server21 Server40 Server18 Server50 Server29 Server26

Рисунок 4.26 Результат пошуку від одного вузла до іншого

Наступним, давайте зробимо декартовий добуток топологій, для цього візьмемо та створимо дві мережі та перейдемо до панелі «Execute Rooted Product». Результат створених двох мереж на рисунку та заповнення панелі «Execute Rooted Product» на рисунку

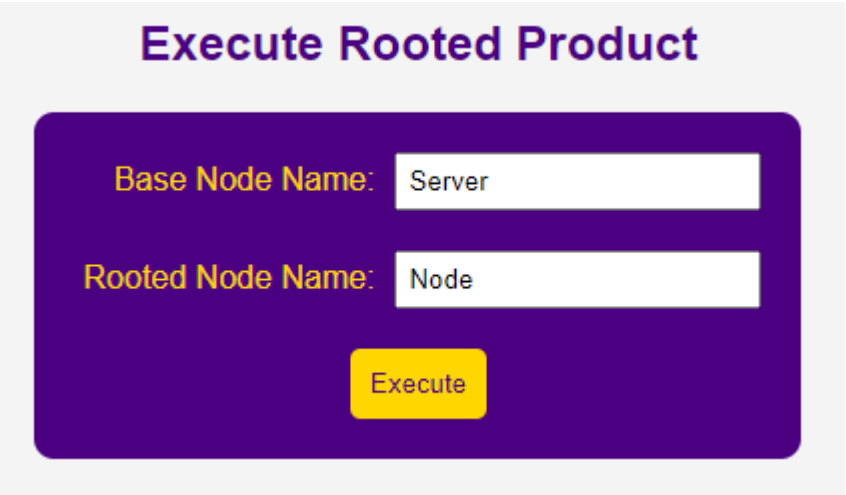


Рисунок 4.27 – Заповнення полів у панелі «Execute Rooted Product»

Результат роботи на рисунку 4.28

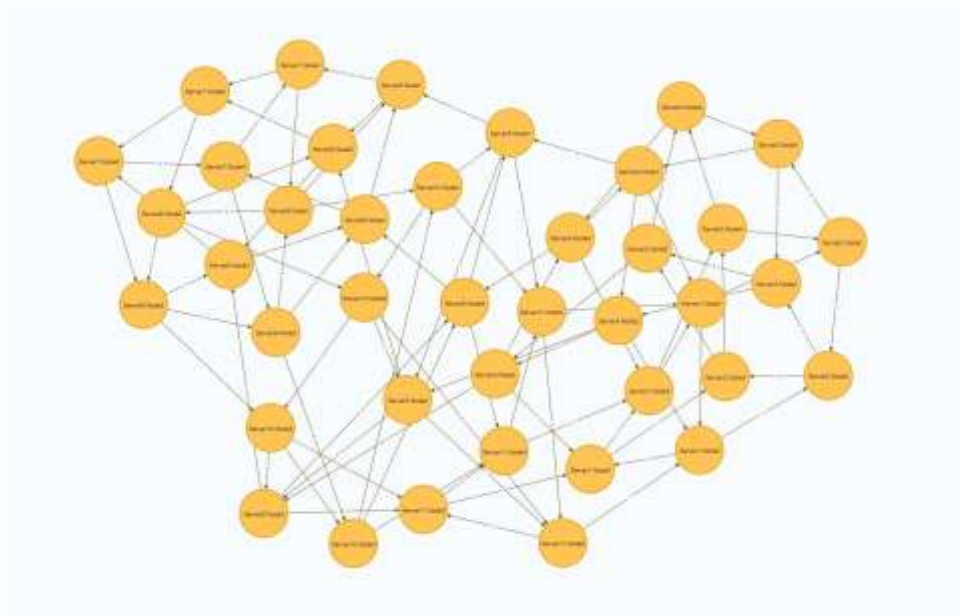


Рисунок 4.28 – Результат декартового добутку топологій

Останнім, буде реалізація графа через кодове перетворення. Для цього нам потрібно перейти до панелі та заповнимо поля. Результат на рисунку 4.29

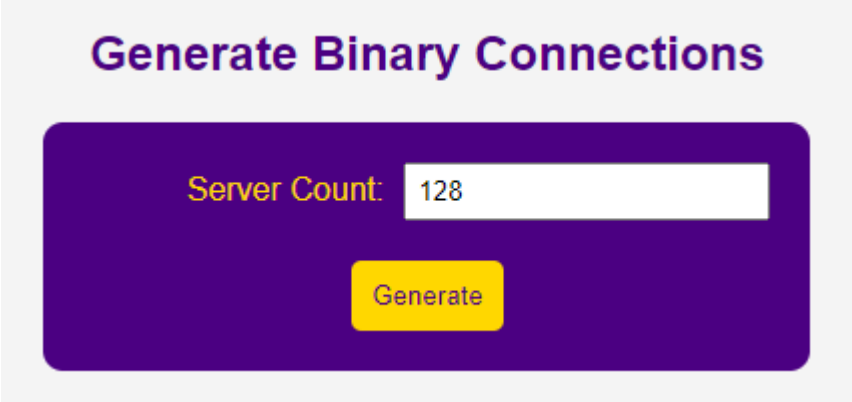


Рисунок 4.29 – Генерація графа через кодове перетворення

Результат роботи на рисунку 4.30

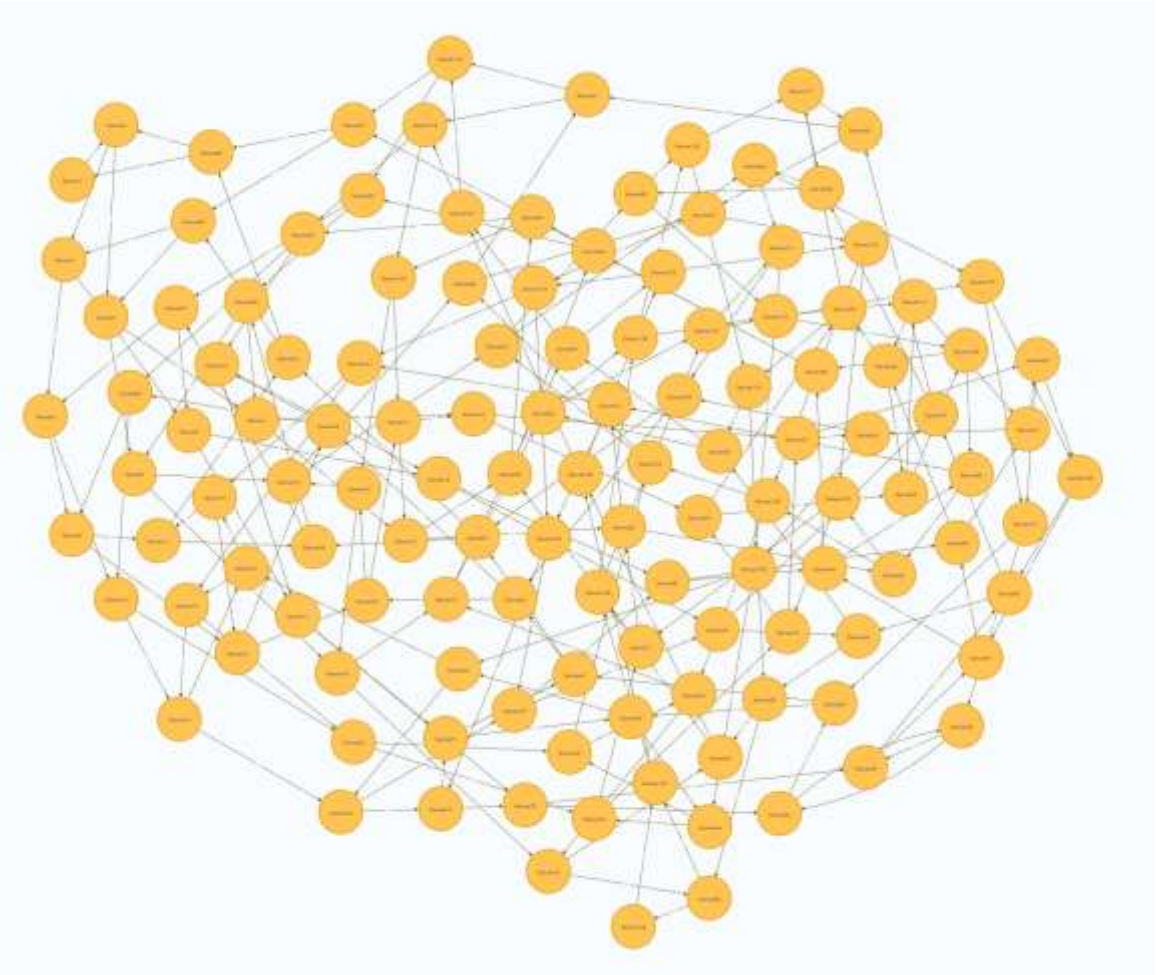


Рисунок 4.30 – Створення графу через кодове перетворення

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Платформа для аналізу та проєктування топологічних організацій розподілених систем

Графічний матеріал

КПІ.ІТ-0213.045440.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Артем ВОЛОКИТА

Нормоконтроль:

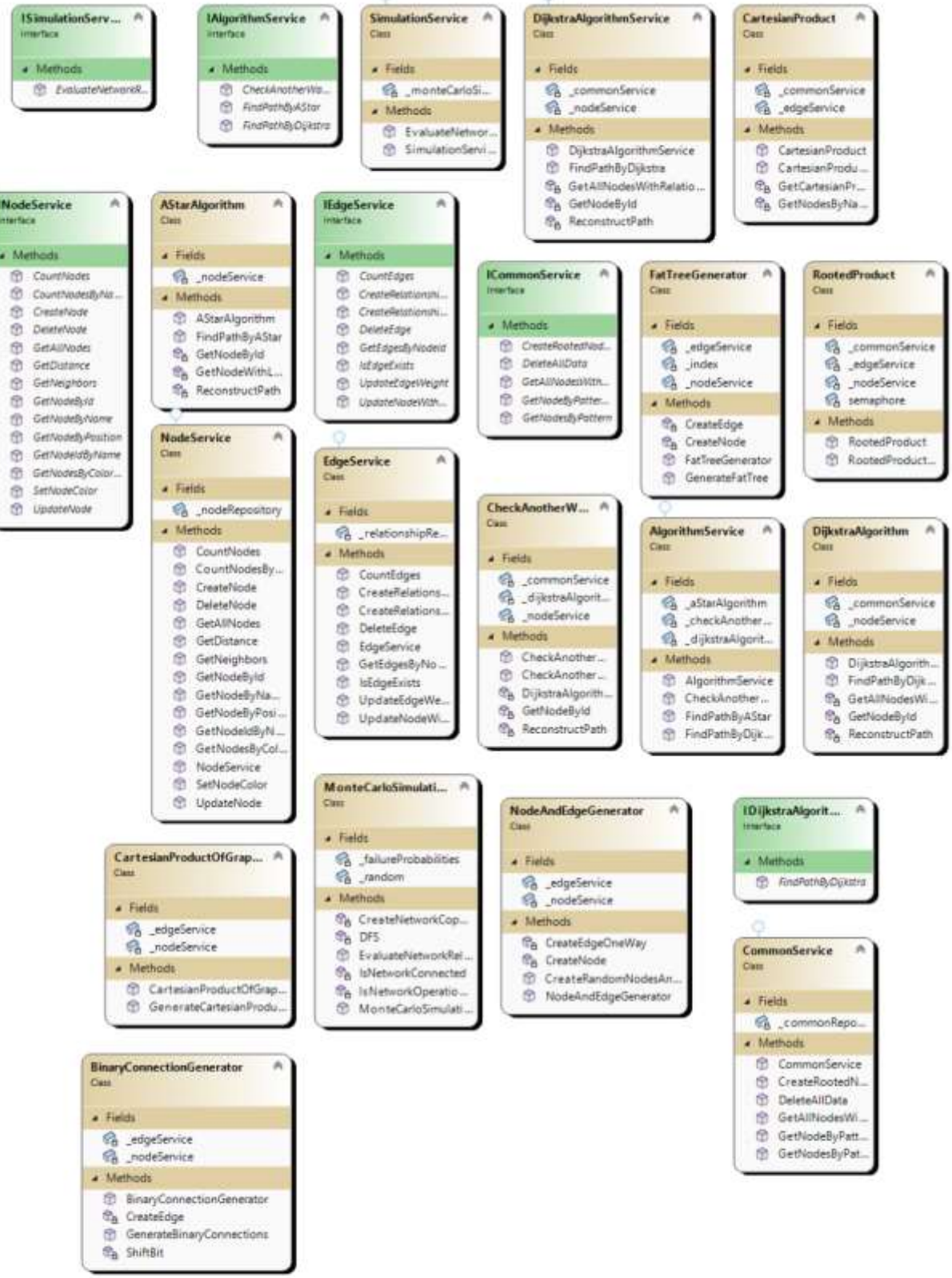
_____ Катерина ЛІЩУК

Виконавець:

_____ Костянтин ЛАЗАНЧУК

Київ – 2024

КПІ ІТ-0213 045440 06 99 ССК



					КПІ.ІТ-0213.045440.06.99.ССК
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Пазанчук К.В.			Схема структурна класів програмного забезпечення
Перевірив		Волокита А.М.			
Т. контр.					
Н. контр.		Пащук К.І.			Платформа для аналізу та проєктування топологічних організацій розподілених систем
Затвердив		Харіков Е.В.			

Лит.	Маса	Масштаб
Аркуш	Аркушів	
КПІ ім. Ігоря Сікорського Кафедра ІПІ гр. ІТ-02		

КПІ.ІП-0213.045440.06.99.ССМ

					КПІ.ІП-0213.045440.06.99.ССМ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Лазанчук К.В.			Схема структурна компонентів програмного забезпечення
Перевірив		Волокіта А.М.			
Т. контр.					
Н. контр.		Ліщук К.І.			Платформа для аналізу та проєктування топологічних організацій розподілених систем
Затвердив		Жаріков Е.В.			

Лит.	Маса	Масштаб
Аркуш	Аркушів	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-02

КПІ.ІП-0213.045440.06.99.ССВ

					КПІ.ІП-0213.045440.06.99.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантва використання	Літера / Маса / Масштаб
Розробив		Лазанчук К.В.				
Перевірив		Волокита А.М.				Аркуш / Аркушів
Т. контр.						
Н. контр.		Ліщук К.І.			Платформа для аналізу та проєктування топологічних організацій розподілених систем	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-02
Затвердив		Жаріков Е.В.				