

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

Випускна кваліфікаційна робота бакалавра
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Вебзастосунок для інтерактивної візуалізації
соціальних графів великої розмірності»**

Виконав студент групи ПЗ-22-3
КУПРІЯНОВ Данило Ігорович

Керівник роботи:
АНТОНЮК Дмитро Сергійович

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КУПРІЯНОВ Данило Ігорович**

Керівник роботи: **АНТОНЮК Дмитро Сергійович**

Тема роботи: **«Вебзастосунок для інтерактивної візуалізації соціальних графів великої розмірності»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **процес обробки та інтерактивної візуалізації соціальних графів великої розмірності у вебсередовищі з використанням методів, силових алгоритмів просторового компонування та технологій апаратного прискорення рендерингу.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Антонюк Д. С.	20.05.2026р	20.05.2026р
2	Антонюк Д. С.	22.05.2026р	22.05.2026р
3	Антонюк Д. С.	24.05.2026р	24.05.2026р

РЕФЕРАТ

У кваліфікаційній роботі розглянуто питання проєктування та реалізації вебзастосунку для інтерактивної візуалізації соціальних графів великої розмірності, призначеного для аналізу складних мережевих структур.

Окрему увагу приділено оптимізації обчислювальних процесів безпосередньо у браузері, механізмам розрахунку фізичних сил просторового компонування (Force-directed algorithms) за допомогою фонових потоків Web Workers, а також реалізації високопродуктивного графічного ядра на базі технології WebGL (для двовимірного та тривимірного просторів). У роботі також описано структуру інтерфейсу, процес парсингу локальних наборів даних (формати JSON, CSV) та результати навантажувального тестування основних сценаріїв використання при відмальовуванні великих обсягів даних.

У роботі проаналізовано предметну область, досліджено наявні аналоги аналітичних систем, обґрунтовано вибір архітектури Single Page Application (SPA) із застосуванням парадигми Local-First, що підвищує рівень конфіденційності інформації та визначено сучасні інструментальні засоби розробки (React, Sigma.js, Graphology).

Метою роботи є аналіз, проєктування та реалізація вебзастосунку для інтерактивної візуалізації соціальних графів, забезпечення зручної взаємодії аналітиків із великими даними та вирішення проблеми візуального перевантаження шляхом інтеграції апаратного прискорення.

Кваліфікаційна робота містить 66 сторінок, 36 ілюстрацій, 4 таблиці, 5 додатків та 34 використаних літературних джерела.

КЛЮЧОВІ СЛОВА: SINGLE PAGE APPLICATION, LOCAL-FIRST, СОЦІАЛЬНИЙ ГРАФ, ВІЗУАЛІЗАЦІЯ, WebGL, СИЛОВИЙ АЛГОРИТМ, ВЕБЗАСТОСУНОК.

					ІПЗ.КР.Б-121.26 - ПЗ								
Змн.	Арк.	№ докум.	Підпис	Дата									
Розроб.		Купріянов Д.І.			Вебзастосунок для інтерактивної візуалізації соціальних графів великої розмірності				Літ.	Арк.	Аркушів		
Керівник		Антонюк Д.С.									3	66	
									ФІКТ, ар. ІПЗ-22-3				
Рецензент		Локтікова Т.М.											
Зав. Каф.		Вакалюк. Т.А.											

ABSTRACT

The qualification work considers the issue of designing and implementing a web application for the interactive visualization of high-dimensional social graphs, designed for the analysis of complex network structures.

Special attention is paid to the optimization of in-browser computational processes, mechanisms for calculating the physical forces of spatial layout (Force-directed algorithms) using background Web Workers, and the implementation of a high-performance graphics core based on WebGL technology (for 2D and 3D spaces). The work also describes the interface structure, the process of parsing local datasets (JSON, CSV formats), and the results of load testing of the main usage scenarios when rendering large data arrays.

The work analyzes the subject area, investigates existing analogues of analytical systems, and justifies the choice of a Single Page Application (SPA) architecture using the Local-First paradigm to ensure complete data confidentiality and modern development tools (React, Sigma.js, Graphology).

The purpose of the work is to analyze, design, and implement a web application for the interactive visualization of social graphs, to ensure convenient interaction of analysts with big data, and to solve the problem of visual clutter by integrating hardware acceleration.

The qualification work contains 66 pages, 36 illustrations, 4 tables, 5 appendices and 34 used literary sources.

KEYWORDS: SINGLE PAGE APPLICATION, LOCAL-FIRST, SOCIAL GRAPH, VISUALIZATION, WEBGL, FORCE-DIRECTED ALGORITHM, WEB APPLICATION.

					ІПЗ.КР.Б-121.26 - ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для інтерактивної візуалізації соціальних графів великої розмірності			Лім.	Арк.	Аркуші
Розроб.		Купріянов Д.І.								
Керівник		Антонюк Д.С.							4	66
								ФІКТ, зр. ІПЗ-22-3		
Рецензент		Локтікова Т.М.								
Зав. Каф.		Вакалюк. Т.А.								

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧ	9
1.1. Постановка задачі та вибір архітектурного підходу	9
1.2. Аналіз аналогів програмного забезпечення	11
1.3. Обґрунтування вибору архітектурного підходу	14
Висновки до першого розділу:.....	17
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ІНТЕРАКТИВНОЇ ВІЗУАЛІЗАЦІЇ СОЦІАЛЬНИХ ГРАФІВ	18
2.1. Архітектура застосунку та підсистема обробки даних	18
2.2. Математичне моделювання та алгоритми компоновання графів	25
2.3. Проєктування користувацького інтерфейсу та управління станами ...	30
Висновки до другого розділу:	33
РОЗДІЛ 3. ПОРЯДОК РОБОТИ, РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ	34
3.1. Базові сценарії візуалізації та управління даними.....	34
3.2. Робота з аналітичними модулями та налаштування простору	38
3.3. Тестування стабільності системи та перевірка сценаріїв	48
Висновки до третього розділу	52
ВИСНОВОК	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ.....	55
ДОДАТКИ.....	59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

API – Application Programming Interface

CORS – Cross-Origin Resource Sharing

CPU – Central Processing Unit

CSV – Comma-Separated Values

DOM – Document Object Model

FPS – Frames Per Second

GPU – Graphics Processing Unit

HR – Human Resources

HTTP – HyperText Transfer Protocol

JS – JavaScript

JSON – JavaScript Object Notation

ORM – Object-Relational Mapping

RAM – Random Access Memory

REST – Representational State Transfer

SMM – Social Media Marketing

SPA – Single Page Application

UI/UX – User Interface / User Experience

WebGL – Web Graphics Library

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Аналіз масивних мереж взаємодій перестав бути виключно академічною задачею. Сучасний бізнес – від банківського сектору до маркетингових агентств – генерує терабайти транзакційних та комунікаційних логів. Звичайні табличні формати даних не здатні розкрити приховані топологічні зв'язки. Пошук тіньових фінансових схем, виявлення макро-інфлюенсерів або аналіз ізольованості корпоративних відділів потребує просторової візуалізації.

Однак рендеринг графів великої розмірності (понад 10 тисяч вершин) у вебсередовищі створює критичне навантаження на апаратну частину. Традиційні браузерні інструменти стикаються з проблемою падіння частоти кадрів та повним блокуванням інтерфейсу під час розрахунку фізичних сил компонування. Тому розробка оптимізованого клієнт-центричного вебзастосунку, який використовує апаратне прискорення WebGL та ізольовані математичні алгоритми для аналізу мереж без залучення серверних потужностей, є надзвичайно актуальним інженерним завданням [12].

Додатковим стимулом для створення такого рішення є питання безпеки комерційних даних. Корпоративний сектор неохоче передає сирі логі транзакцій чи списки клієнтів на сторонні сервери для аналітики. Класичні клієнт-серверні додатки мають постійні ризики витоку інформації. Тому використання парадигми Local-First, коли всі масиви даних завантажуються та обробляються виключно в оперативній пам'яті браузера користувача, вирішує проблему конфіденційності. Відмова від класичних маніпуляцій із DOM-деревом на користь WebGL та ізольованих фонових потоків Web Workers дозволяє обійти базові ліміти браузера і дає змогу працювати з масивними графами без втрати швидкодії інтерфейсу.

Метою кваліфікаційної роботи є проектування та розробка високонавантаженого вебзастосунку для інтерактивної візуалізації, топологічного аналізу та кластеризації соціальних графів великої розмірності у двовимірному та тривимірному просторах.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

Завданням дослідження є досягнення поставленої мети, шляхом виконання визначених етапів:

- аналіз предметної області, наявних програмних аналогів та проблематику рендерингу великих обсягів даних;
- дослідження наявних архітектурних підходів;
- аналіз вимог щодо захисту персональних даних;
- проєктування клієнт-центричної архітектури вебзастосунку для забезпечення інформаційної безпеки та швидкодії;
- дослідження та імплементування алгоритму просторового компонування графів із застосуванням паралельних обчислень;
- обґрунтування та вибір інструментальних засобів;
- розроблення інтерактивного користувацького інтерфейсу та інтегрування модулів галузевої бізнес-аналітики;
- програмна реалізація клієнтської частини вебзастосунку;
- проведення функціонального та навантажувального тестування розробленої системи на великих наборах даних;

Методи дослідження. Для розв'язання поставлених завдань у роботі використано комплекс методів теорії графів, методи математичного моделювання, а також методи об'єктно-орієнтованого та функціонального програмування.

Об'єкт дослідження є процесом візуального та математичного аналізу складних структур даних у вигляді масивних графів.

Предмет дослідження є алгоритм просторового компонування, методи топологічної кластеризації та програмні оптимізації рендерингу в браузерному середовищі. Для розв'язання поставлених завдань у роботі використано комплекс методів теорії графів для обчислення метрик центральності та виявлення прихованих спільнот, методів математичного моделювання для симуляції фізичних процесів просторового компонування та методи об'єктно-орієнтованого та функціонального програмування для проєктування модульної архітектури користувацького інтерфейсу на базі React.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧ

1.1. Постановка задачі та вибір архітектурного підходу

Наразі аналітичні центри, маркетологи та дослідники даних стикаються з необхідністю обробки значних обсягів мережевої інформації. У традиційних системах інформація про подібні структури зберігається у вигляді сирих масивів даних: списків суміжності або розріджених матриць зв'язків. Аналіз великої кількості таких зв'язків вручну або за допомогою звичайних таблиць стає вкрай неефективним, оскільки людський мозок найкраще сприймає патерни саме у візуальному просторовому поданні. Тому фахівці гостро потребують систем, що забезпечують швидку математичну обробку та оперативний візуальний доступ до даних, які відображають структури соціальних мереж.

У зв'язку з цим актуальною є задача розробки вебзастосунку для інтерактивної візуалізації соціальних графів великої розмірності. Цей застосунок має автоматизувати процес просторової побудови мережі (компонування) та забезпечити зручний доступ до аналітики з використанням стандартного веббраузера без необхідності завантаження додаткового десктопного ПЗ. Основною функціональністю такого застосунку є імпорт датасетів у форматі JSON та CSV, розрахунок координат, побудова вершин та ребер, візуальна фільтрація даних за певними атрибутами та інтерактивне управління параметрами фізичної симуляції графа в режимі реального часу.

Для розв'язання поставленої задачі доцільно використання сучасних вебтехнологій, що підтримують апаратне прискорення графіки [9]. Важливо зазначити, що застосування модульного (компонентного) підходу є одним із ключових методів вирішення задачі. Він передбачає логічний поділ клієнтської частини системи на зону управління користувацьким інтерфейсом (панелі налаштувань, сайдбар) та відокремлене математичне і графічне ядро (вікно рендерингу).

Для реалізації важких математичних обчислень доцільніше використання технології фонових потоків Web Workers, яка забезпечить високу продуктивність

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				9
Змн.	Арк.	№ докум.	Підпис	Дата		

та асинхронну модель розрахунку силових алгоритмів (Force-directed algorithms) [11]. Як графічний рушій доцільно використати низькорівневий API WebGL, що сприяє підвищенню продуктивності рендерингу за рахунок відеокарти [12]. Використання даного комплексного підходу до проєктування програмної системи забезпечить масштабованість, стабільність відображення десятків тисяч елементів та зручність кінцевого використання.

Враховуючи поставлену задачу та специфіку предметної області, виникає необхідність ретельного підбору архітектурної моделі. Традиційні рішення демонструють низьку ефективність при передачі масивних топологічних структур через мережу, що вимагає переходу до децентралізованих підходів обробки даних.

Для реалізації вебзастосунку було обрано архітектуру Single Page Application (SPA) із застосуванням парадигми Local-First [6,21]. Такий підхід принципово відрізняється від класичних клієнт-серверних рішень, оскільки передбачає повну відмову від серверної частини (бекенду) для зберігання та обробки даних. Уся логіка, збереження стану системи та математичні розрахунки виконуються виключно у браузері користувача.

Специфікою обраної архітектури є застосування концепції «Thick Client» [30]. Це означає, що абсолютне обчислювальне навантаження, пов'язане з виконанням математичних силових алгоритмів (перерахунок координат тисяч вузлів), повністю лягає на процесор та відеокарту самого користувача. Оскільки фізична симуляція соціального графа вимагає десятків оновлень стану щосекунди, використання серверних потужностей для розрахунку кожного кадру та передачі його по мережі є технічно неефективним через високі показники мережевої затримки [10]. Локальна обробка даних безпосередньо у браузері максимізує швидкодію системи та забезпечує миттєвий відгук інтерфейсу.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз аналогів програмного забезпечення

На сучасному ринку програмного забезпечення існує низка рішень для візуалізації та аналізу мережових структур, як десктопних, так і хмарних. Для прийняття правильних архітектурних рішень необхідно провести їх детальний аналіз.

Найвідомішим десктопним інструментом академічного рівня є програма Gephi, яка забезпечує обробку сотень тисяч вузлів [4]. Ця система представлена у вигляді десктопної програми, де користувач може завантажити свої масиви даних та застосувати до них різноманітні алгоритми кластеризації (рис. 1.1).

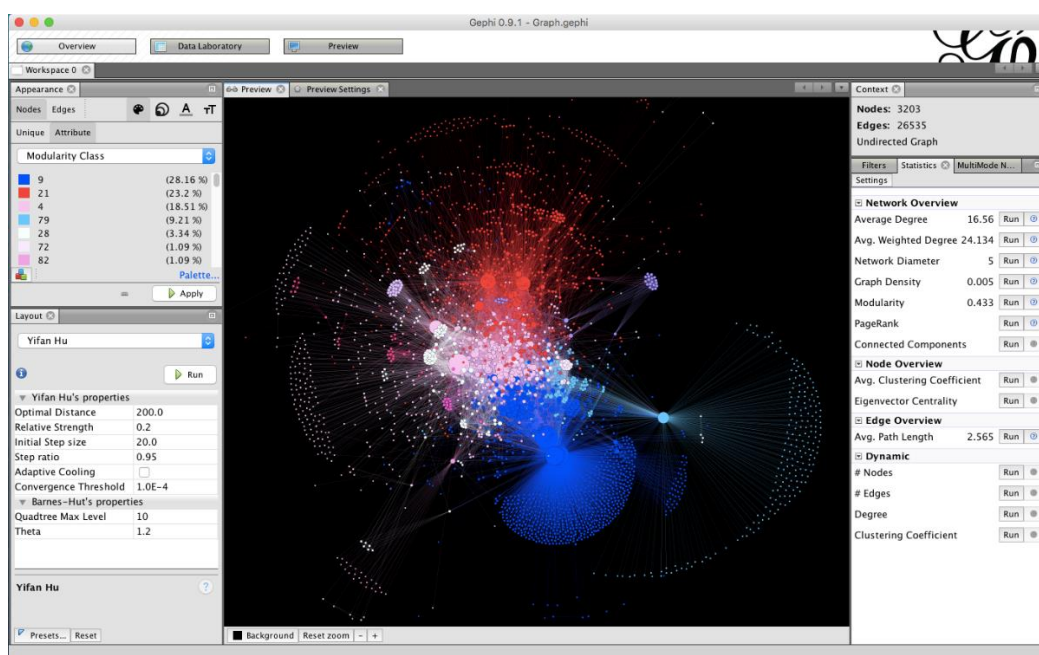


Рис. 1.1. Головна сторінка Gephi

Щодо функціональних можливостей, інструментарій Gephi підтримує високоефективні алгоритми кластеризації та просторового компонування (наприклад, ForceAtlas2) [9]. Програмний комплекс здатний обробляти масивні графи, надаючи аналітику розширений набір фільтрів. Такий математичний апарат зробив зазначену платформу визнаним стандартом для проведення глибоких досліджень у галузі топологічного аналізу мереж. Проте, як класичне десктопне програмне забезпечення, Gephi має вагомі недоліки: система вимагає встановлення Java-оточення, характеризується високим порогом входу для

нових користувачів, а результати її роботи складно оперативно інтегрувати в сучасні корпоративні вебпортали для спільного доступу. Крім того, дистрибуція інтерактивних результатів потребує залучення додаткових технічних ресурсів, оскільки базова архітектура орієнтована переважно на генерацію статичних візуалізацій.

Іншим популярним рішенням, спрямованим безпосередньо на веброзробку, є JavaScript-бібліотека Cytoscape.js (рис. 1.2) [31].

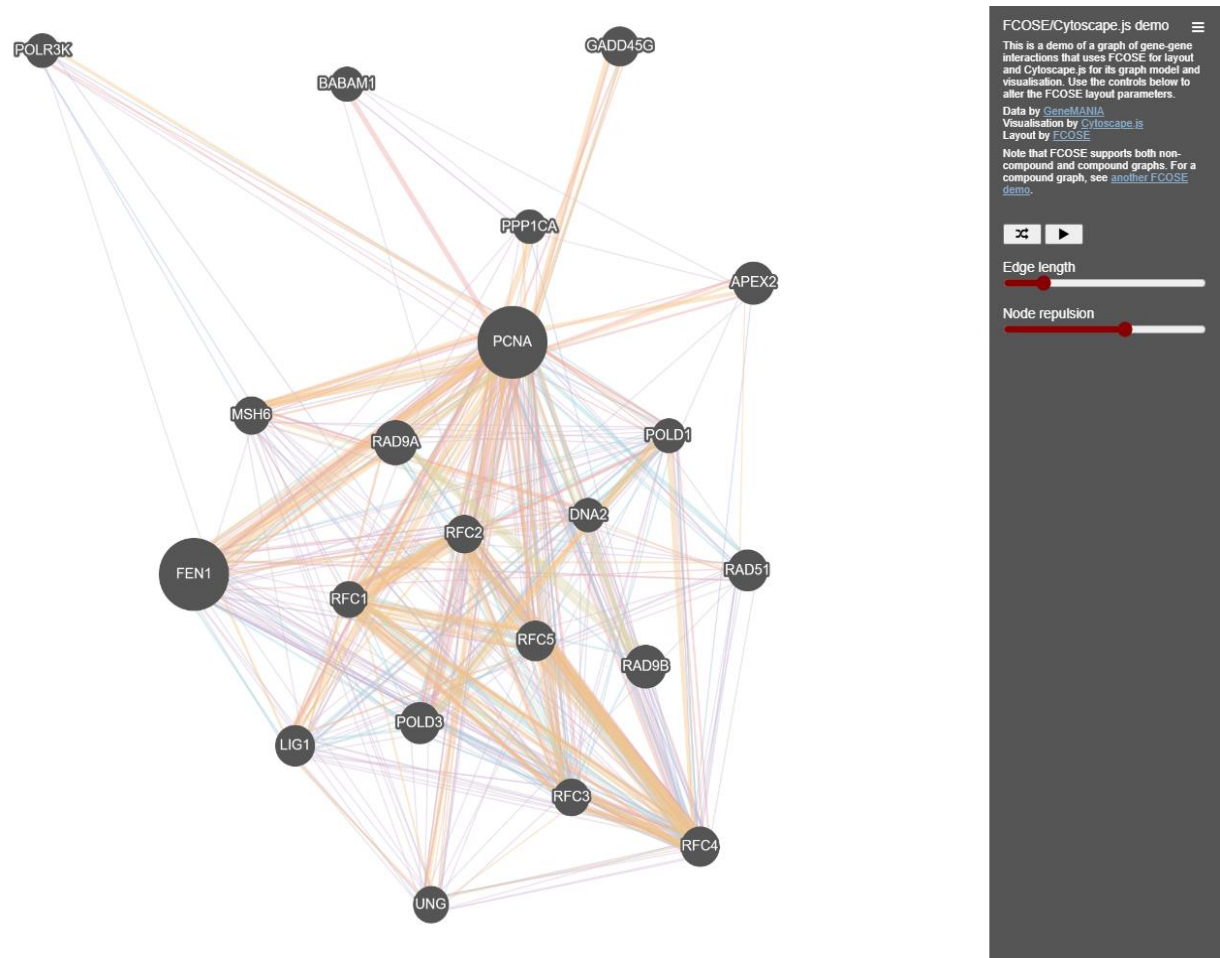


Рис. 1.2. Візуалізація графа за допомогою Cytoscape.js

Цей сервіс працює безпосередньо в браузері і його основним завданням є моделювання невеликих біологічних та соціальних мереж. Бібліотека пропонує чудову інтерактивність та підтримку мобільних пристроїв. Проте її архітектура базується на застарілих технологіях маніпуляції DOM-елементами або використанні HTML5 Canvas 2D [18, 32]. Унаслідок цього під час ініціалізації та

динамічної візуалізації наборів даних розмірністю понад 5000 елементів обчислювальне навантаження на центральний процесор (CPU) стрімко зростає, що призводить до блокування головного потоку виконання та повної втрати реактивності клієнтського інтерфейсу.

Третім інноваційним аналогом є корпоративна платформа Graphistry, яка орієнтована на корпоративний сегмент та кібербезпеку [33] (рис. 1.3).

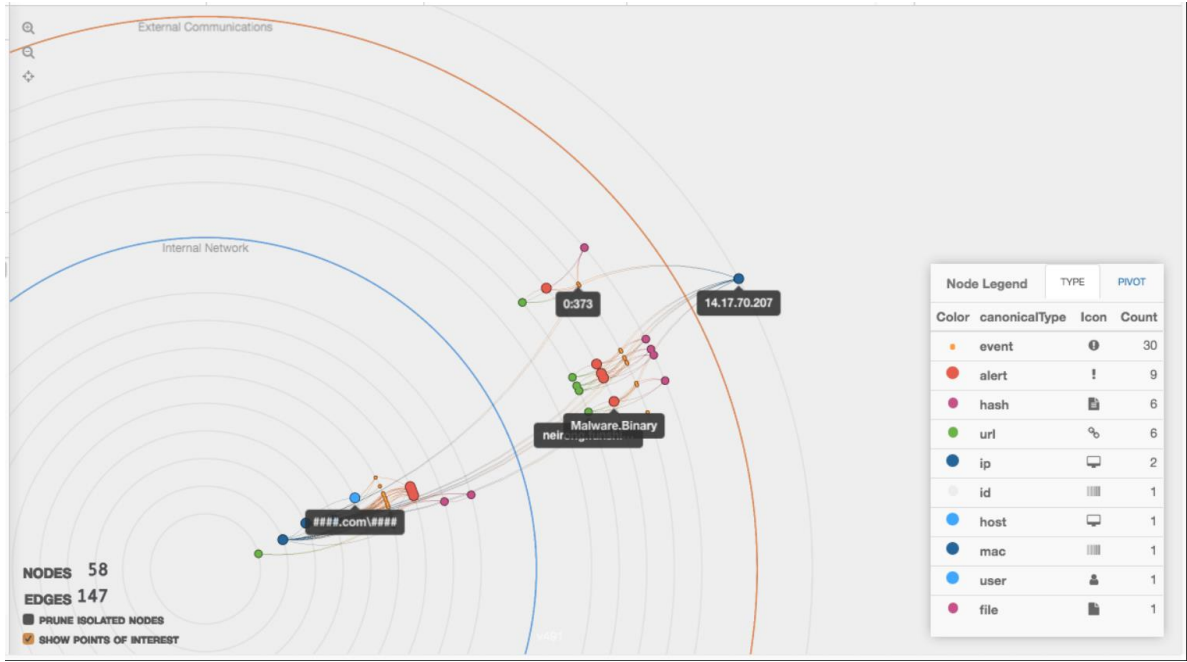


Рис. 1.3. Інтерфейс платформи Graphistry

Ця платформа демонструє найсучасніший підхід, використовуючи віддалені хмарні графічні процесори для рендерингу графів, що складаються з мільйонів вузлів, транслюючи результат на клієнтський комп'ютер. Незважаючи на високу продуктивність системи, її комерційний характер, висока вартість ліцензій та закритий вихідний код роблять її недоступною для локальних розробок або створення незалежних дослідницьких інструментів.

Отже, на ринку існують різноманітні інформаційні системи для роботи з графами. Проте більшість із них має суттєві обмеження: жорстку прив'язку до серверної інфраструктури, платне корпоративне ліцензування або складність локального розгортання. Це обґрунтовує актуальність розробки власного клієнт-

центричного вебзастосунку, який здатен забезпечити високу швидкодію візуалізації великих масивів даних абсолютно безкоштовно та локально у браузері кінцевого користувача.

За результатами аналізу складено порівняльну таблицю характеристик аналогів (таблиця 1.1).

Таблиця 1.1

Таблиця порівняльного аналізу системи

Критерії порівняння	Gephi	Cytoscape.js	Graphistry	Розроблюваний вебзастосунок
Робота безпосередньо у браузері	-	+	+	+
Висока продуктивність (>20 000 вузлів)	+	-	+	+
Безкоштовність та відкритий код	+	+	-	+
Апаратне прискорення графіки (WebGL)	-	-	+	+
Не потребує встановлення додаткового ПЗ	-	+	+	+

1.3. Обґрунтування вибору архітектурного підходу

Ефективність функціонування web-орієнтованої системи для інтерактивної візуалізації складних мережових структур безпосередньо залежить від коректності вибору технологічного стеку та інструментальних засобів розробки на кожному рівні архітектури. Основним критерієм вибору інструментів для даної кваліфікаційної роботи є забезпечення максимальної продуктивності графічної підсистеми клієнта при обробці великих масивів даних у поєднанні із гнучкістю та швидкістю розробки.

Як базову технологічну платформу та основну мову програмної реалізації було обрано JavaScript [2]. Даний вибір зумовлений її універсальністю,

кросплатформністю та нативною підтримкою з боку всіх сучасних веббраузерів без необхідності використання сторонніх плагінів чи модулів розширення.

Враховуючи вимоги до безпеки та продуктивності, було прийнято архітектурне рішення повністю відмовитися від зовнішніх серверів на користь браузерних обчислень. Замість передачі важких датасетів через мережу, система використовує нативний HTML5 File API (зокрема клас FileReader) та спеціалізовані інструменти потокового парсингу, такі як бібліотека PapaParse для локального зчитування структурованих файлів (JSON, CSV) безпосередньо в оперативну пам'ять [5]. Це гарантує високий рівень конфіденційності корпоративної аналітики та мінімізує час ініціалізації.

Для побудови користувацького інтерфейсу аналітичного дашборду було обрано JavaScript-бібліотеку React [2]. Вибір обґрунтований її компонентною архітектурою, яка дозволяє декомпонувати складний інтерфейс системи на незалежні, ізольовані модулі, що повторно використовуються (панель завантаження файлів, блок повзунків фізичної симуляції, інформаційні віджети метаданих).

Ключовою перевагою React для даного проєкту є технологія Virtual DOM та гнучкі механізми управління життєвим циклом компонентів. Оскільки симуляція графа передбачає безперервну зміну тисяч просторових координат, пропускання цих потоків даних через стандартні стани React призвело б до критичного перевантаження браузера [2,26]. Саме тому архітектура системи використовує Virtual DOM виключно для обслуговування панелей керування [26]. Це мінімізує кількість ресурсомістких маніпуляцій із реальним DOM-деревом під час взаємодії користувача з фільтрами чи повзунками [18]. Разом з тим важке математичне ядро та графічне полотно ізольовані від циклів перемальовування React, що забезпечує стабільну плавність навіть під екстремальними навантаженнями.

Математичне моделювання та представлення внутрішньої структури мережі у пам'яті застосунку реалізується за допомогою спеціалізованої JavaScript-бібліотеки Graphology [8]. Дана бібліотека надає строго типізовані, оптимізовані

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

структури даних для збереження орієнтованих, неорієнтованих та змішаних графів, підтримує швидку генерацію ітераторів по вершинах та суміжних ребрах, а також містить вбудовані методи розрахунку топологічних метрик (наприклад, ступенів вершин), що необхідно для подальшого аналізу [22].

Для подолання обмежень продуктивності, притаманних стандартним технологіям HTML5 Canvas 2D, безпосередній графічний вивід здійснюється за допомогою бібліотеки Sigma.js, яка функціонує на базі низькорівневого графічного API WebGL [12, 24, 32]. Завдяки WebGL графічне ядро вебзастосунку отримує прямий доступ до апаратного прискорення відеокарті клієнтського пристрою. Замість послідовного малювання елементів на CPU, Sigma.js трансліює масиви координат вершин та ребер у вершинні буфери і обробляє їх паралельно за допомогою шейдерів безпосередньо на GPU. Поряд із цим для реалізації повноцінної тривимірної візуалізації мережі було інтегровано спеціалізовану бібліотеку React Force Graph 3D, що дозволяє ефективно та плавно рендерити граф великої розмірності у 3D-просторі [27].

Це архітектурне рішення дозволяє підтримувати високу плавність відображення при інтерактивних маніпуляціях (масштабуванні та панорамуванні) з графами великої розмірності.

Для забезпечення повної реактивності інтерфейсу під час виконання ресурсомістких ітераційних розрахунків просторового рознесення вузлів (алгоритми типу Force-directed) для 2D-режиму та інтегрованого рушія d3-force для 3D-режиму) інтегровано технологію Web Workers [28, 29]. Дане рішення дозволяє створити ізольований від основного інтерфейсу фоновий потік обчислень на рівні операційної системи.

Обчислювання функціонує всередині Web Worker, де безперервно перераховує просторові вектори сил притягання та відштовхування вершин, асинхронно передаючи готові координати в головний потік для рендерингу WebGL, завдяки чому інтерфейс користувача зберігає повну реактивність на події введення (кліки та рухи миші).

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

Для забезпечення стабільної, безперебійної та продуктивної роботи розроблюваного вебзастосунку на основі проведеного аналізу архітектури визначено комплексні вимоги до програмно-апаратного оточення.

Мінімальні вимоги до клієнтських пристроїв:

Процесор із частотою не менше 3.0 ГГц

Оперативна пам'ять не менше 8 ГБ

Вільний дисковий простір – не менше 2 ГБ

Наявність дискретної або інтегрованої відеокарти

Наявність сучасного веббраузера

Доступ до мережі інтернет

Операційна система Linux (Ubuntu, Debian) або Windows 10/11

Висновки до першого розділу:

У першому підрозділі було детально проаналізовано та поставлено задачу, визначено проблематику візуального перевантаження та описано основні функції для реалізації системи інтерактивної візуалізації графів.

У другому підрозділі знайдено відомі аналоги програмного забезпечення (Gephi, Cytoscape.js, Graphistry), визначено їх переваги, недоліки та функціональність для врахування їх у створенні власної системи.

У третьому підрозділі було визначено та обґрунтовано вибір парадигми Local-First та архітектури Single Page Application (SPA). Доведено доцільність повної відмови від класичної клієнт-серверної моделі на користь перенесення обчислювального навантаження на пристрій користувача (Thick Client).

У четвертому підрозділі детально обґрунтовано вибір інструментальних засобів розробки та застосування бібліотеки React у зв'язці з технологією фонових потоків Web Workers для паралельних обчислень та низькорівневого графічного API WebGL для апаратного прискорення рендерингу. Окрім цього, визначено мінімальні вимоги до апаратного забезпечення клієнтських систем.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ІНТЕРАКТИВНОЇ ВІЗУАЛІЗАЦІЇ СОЦІАЛЬНИХ ГРАФІВ

2.1. Архітектура застосунку та підсистема обробки даних

Проектування архітектури вебзастосунку для інтерактивної візуалізації соціальних графів великої розмірності спирається на жорсткі вимоги до швидкодії. Головна концептуальна умова – це автономність обробки даних та чітке розмежування зон відповідальності.

З огляду на обрану парадигму локальної обробки, єдиним актором у системі виступає «Аналітик» (кінцевий користувач) [30]. Вебзастосунок повністю відкидає використання централізованих серверних баз даних для зберігання конфіденційної аналітики. Відповідно, класичні системи авторизації користувачів, що базуються на рішеннях Passport.js, виявляються надлишковими. Потреба в ролях «Адміністратора» або «Модератора» повністю зникає [15].

Це перетворює застосунок на незалежний, ізольований професійний інструмент, який можна використовувати навіть у закритих корпоративних мережах без доступу до Інтернету.

Функціональний простір Аналітика охоплює весь цикл роботи з графовою моделлю: від відкриття сирих JSON/CSV-файлів до візуальної навігації у тривимірному просторі та застосування галузевих математичних метрик. Опис базових прецедентів використання наведено у таблиці 2.1.

Таблиця 2.1

Опис прецедентів актора «Аналітик»

Назва варіанту використання	Розроблюваний вебзастосунок
Завантаження датасетів	Завантаження структурованих файлів (JSON, CSV) із локального носія безпосередньо в оперативну пам'ять браузера. Зворотне вивантаження обчислених матриць.

Просторова навігація (2D/3D)	Інтерактивна взаємодія зі сценою: масштабування, панорамування, автоматичне центрування та фокусування камери на конкретному вузлі за його ідентифікатором.
Управління фізикою компонування	Зміна параметрів гравітації, сили відштовхування вершин та масштабу відображення в режимі реального часу під час активної симуляції.
Запуск бізнес-модулів	Виклик алгоритмів для вирішення прикладних задач: пошук лідерів думок (Маркетинг), локалізація тіньових схем (Фрод-контроль) та картування відділів (HR).
Тестування та генерація (Пісочниця)	Запуск математичних моделей (наприклад, алгоритму Барабаші-Альберта) для створення синтетичних безмасштабних мереж та стрес-тестування системи візуалізації.

Для забезпечення системної цілісності розроблюваної системи необхідно деталізувати функціональні сценарії взаємодії актора «Аналітик» із внутрішніми модулями вебзастосунку. Кожен із наведених у таблиці 2.1 прецедентів ініціює складний ланцюжок асинхронних подій у клієнтському середовищі.

Сценарій «Завантаження датасетів». Даний прецедент є точкою входу в аналітичний цикл. Взаємодія розпочинається з активації інтерфейсного елемента вибору файлів, що тригерує нативний HTML5 File API. Програмна система не здійснює відправку сирого файлу на віддалений сервер, а натомість ініціалізує екземпляр класу FileReader. Залежно від виявленого розширення (.json або .csv), керування передається відповідному синтаксичному аналізатору. Для форматів CSV залучається інтегрований потоковий парсер, який асинхронно розбиває текстові рядки на об'єкти зв'язків (Source, Target, Weight). Після успішної десеріалізації об'єктів у пам'яті створюється граф-структура, а інтерфейс користувача переходить у стан відображення метаданих.

Сценарій «Просторова навігація (2D/3D)». Цей функціональний блок відповідає за ергономіку дослідження топології. Користувач здійснює маніпуляції за допомогою периферійних пристроїв введення (миша, трекпад). У двовимірному просторі рушій Sigma.js перехоплює події панорамування та

масштабування, миттєво перераховуючи матрицю афінних перетворень на графічному процесорі (GPU). У тривимірному просторі Three.js керує віртуальною камерою у межах координат. Додатково функціонує алгоритм автоматичного центрування, який динамічно обчислює геометричний центр мас поточної мережі та плавно переміщує фокус камери, запобігаючи втраті об'єкта з поля зору аналітика.

Сценарій «Управління фізикою компонування». Взаємодія передбачає модифікацію поведінки силового аттрактора безпосередньо під час виконання ітерацій симуляції. Аналітик через інтерфейсні повзунки змінює коефіцієнти гравітації, сили кулонівського відштовхування та масштабного коефіцієнта. Програмний комплекс перехоплює ці зміни через реактивні стани та асинхронно транслює оновлені числові параметри у фоновий потік Web Worker. Фізичний рушій миттєво адаптує вектори сил, що дозволяє користувачеві візуально спостерігати процес стиснення або розширення граф-структури у режимі реального часу.

Сценарій «Запуск бізнес-модулів». Прецедент орієнтований на виконання прикладних задач аналізу. Користувач обирає цільову галузеву вкладку («Маркетинг», «Фінанси» або «HR відділ»). Запуск сканування активує специфічні математичні фільтри. Система ініціює розрахунок топологічних метрик (алгоритм Лувена або центральність ступеня), ізолюючи важкі ітераційні обходи матриці суміжності. Після завершення математичного аналізу результати повертаються у вигляді модифікованих атрибутів вершин, які візуально відображають приховані закономірності датасету.

Сценарій «Тестування та генерація (Пісочниця)». Даний прецедент дозволяє аналітику функціонувати в режимі автономного моделювання. Користувач самостійно задає межі розмірності (кількість вершин і ребер) та активує генератор топологій Барабаші-Альберта. Система програмно імітує еволюційне зростання мережі на основі принципу преференційного приєднання. Це дає змогу проводити стрес-тестування графічного конвеєра та вивчати поведінку мереж за відсутності реальних файлів даних.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Для формалізації функціональних вимог до системи та візуального представлення взаємодії користувача з програмним комплексом було розроблено діаграму прецедентів використання згідно з методологією UML, яку спроектовано за допомогою синтаксису PlantUML [1, 16] (рис. 2.1).

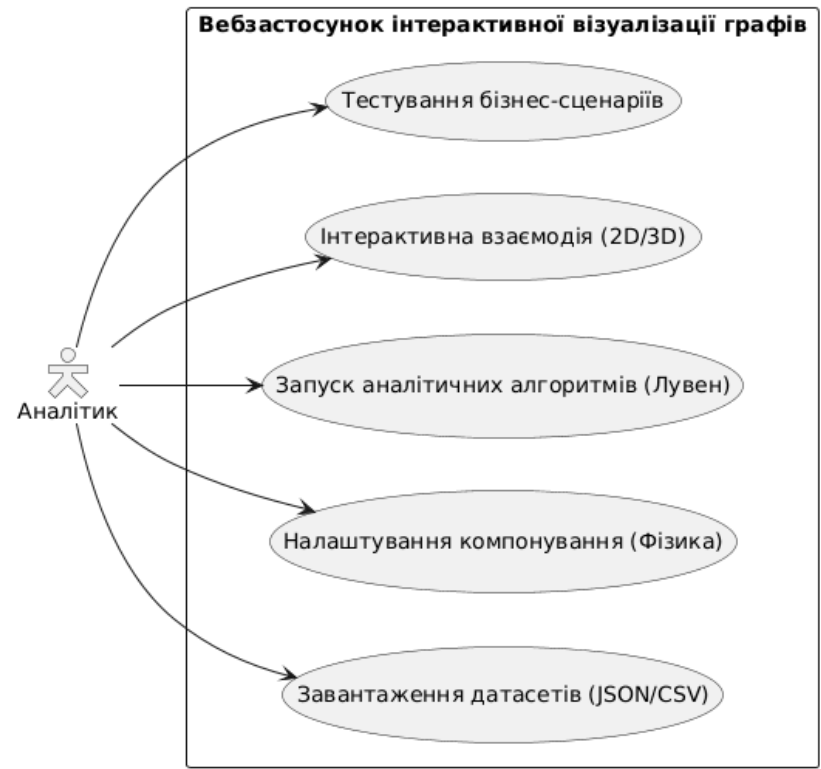


Рис. 2.1. Діаграма прецедентів (Use Case)

Як видно з рисунка 2.1, процес роботи Аналітика є лінійним та ітеративним. Усе починається з ініціалізації датасету, після чого користувач отримує доступ до паралельних гілок аналізу. Відсутність складних розгалужень ролей та модулів авторизації мінімізує когнітивне навантаження на спеціаліста.

Для реалізації заявленого функціоналу програмний комплекс спроектовано за допомогою модульної клієнтської архітектури. Відмова від класичної моделі «клієнт-сервер» гарантує максимальну швидкодію, оскільки усуває мережеві затримки при передачі сотень тисяч координат.

Структуру даного архітектурного підходу можна розділити на три логічні рівні, які функціонують виключно на стороні клієнта. Першим є рівень користувацького інтерфейсу, побудований на базі компонентного підходу бібліотеки React [2]. Він відповідає за відображення панелей управління

(сайдбару), обробку дій користувача та передачу команд до обчислювального ядра.

Другим виступає обчислювальне ядро та стан. Тут структура графа зберігається у спеціалізованих структурах даних Graphology, а для розрахунку фізичних сил використовується ізольований фоновий потік Web Worker, що дозволяє виконувати ітераційні обчислення без блокування браузера [8, 11].

Третім є рівень апаратного рендерингу, який отримує розраховані координати від фонового потоку і транлює їх у графічне представлення за допомогою технології WebGL та бібліотек Three.js, Sigma.js [12, 34, 24]. Завдання з відмальовування тисяч геометричних примітивів делегується безпосередньо на графічний процесор комп'ютера користувача, що забезпечує стабільну частоту кадрів.

Інформаційна взаємодія між цими рівнями відбувається асинхронно через механізми подій та передачу повідомлень, що робить систему стійкою до високих навантажень навіть при роботі з графами великої розмірності.

Для кращого розуміння інформаційних потоків та взаємодії між вищезазначеними рівнями було побудовано схему модульної клієнтської архітектури вебзастосунку (рис. 2.2).

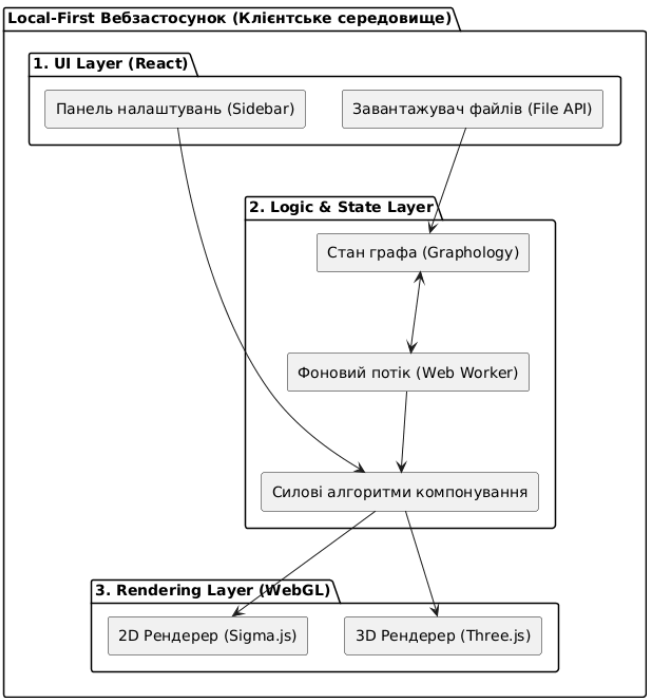


Рис. 2.2. Модульна клієнтська архітектура вебзастосунку

На рисунку 2.2 детально проілюстровано концепцію Data Flow всередині клієнтського середовища. Процес розпочинається на рівні UI, де за допомогою HTML5 File API відбувається читання та десеріалізація користувацького файлу. Отримані дані передаються до логічного ядра, яке ініціалізує математичну модель графа. Ключовою архітектурною особливістю, зображеною на схемі, є двостороння асинхронна взаємодія між станом графа та ізольованим фоновим потоком Web Worker.

Web Worker виконує ресурсомісткі ітераційні обчислення і передає готові просторові координати вершин безпосередньо до рівня рендерингу (WebGL), де бібліотеки Sigma.js (для 2D) та Three.js (для 3D) здійснюють фінальне відмальовування на графічному процесорі.

Враховуючи специфіку роботи аналітиків із чутливими корпоративними даними (наприклад, логами банківських транзакцій, списками клієнтів або внутрішніми комунікаціями компаній), у розроблюваному вебзастосунку було прийнято рішення відмовитися від класичних реляційних серверних баз даних, таких як PostgreSQL, та засобів ORM таких як Sequelize [17, 23]. Замість використання клієнтських бібліотек для кешування серверних станів, таких як TanStack Query, система маніпулює виключно локальними даними [25]. Відсутність мережевих запитів до бекенду через протокол HTTP/1.1 усуває критичні затримки [20].

Такий підхід вирішує критичні проблеми B2B-сегменту. По-перше, забезпечується високий рівень інформаційної безпеки, оскільки дані, що становлять комерційну таємницю, не передаються через мережу Інтернет на сторонні сервери. Це виключає ризик їх перехоплення або витоку через вразливості баз даних. По-друге, досягається максимальна продуктивність завдяки відсутності мережевих запитів до бекенду, що усуває затримки, пов'язані з пропускнуою здатністю. По-третє, гарантується автономність, завдяки чому аналітик має можливість працювати із системою в ізольованих корпоративних мережах.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Далі система перевіряє розширення файлу та ініціює відповідний алгоритм обробки. Під час обробки формату JSON використовується клас FileReader, який асинхронно зчитує вміст файлу [7]. Після зчитування система валідує наявність масивів вузлів та ребер, додаючи їх до екземпляра графа. У випадку обробки таблиць CSV інтегровано бібліотеку-парсер PapaParse [5]. Вона асинхронно аналізує рядки файлу, автоматично розпізнаючи ключові колонки джерела та цілі, динамічно генеруючи відсутні вузли та встановлюючи зв'язки.

Процес завантаження та обробки реалізовано за допомогою нативного HTML5 File API згідно з актуальною специфікацією HTML Standard [32]. Коли користувач обирає файл через інтерфейс застосунку, браузер створює об'єкт.

Після успішної десеріалізації та побудови графової структури в оперативній пам'яті, модуль парсингу ініціює подію оновлення стану застосунку.

Ця подія автоматично запускає перерахунок фізичних сил компонування ForceAtlas2 та оновлює графічний буфер WebGL для візуалізації завантаженої мережі [12, 29].

Для детальнішого розуміння часових інформаційних потоків та виклику методів під час виконання ресурсомістких алгоритмів було додатково спроектовано діаграму послідовностей (Sequence Diagram) (рис.2.3).

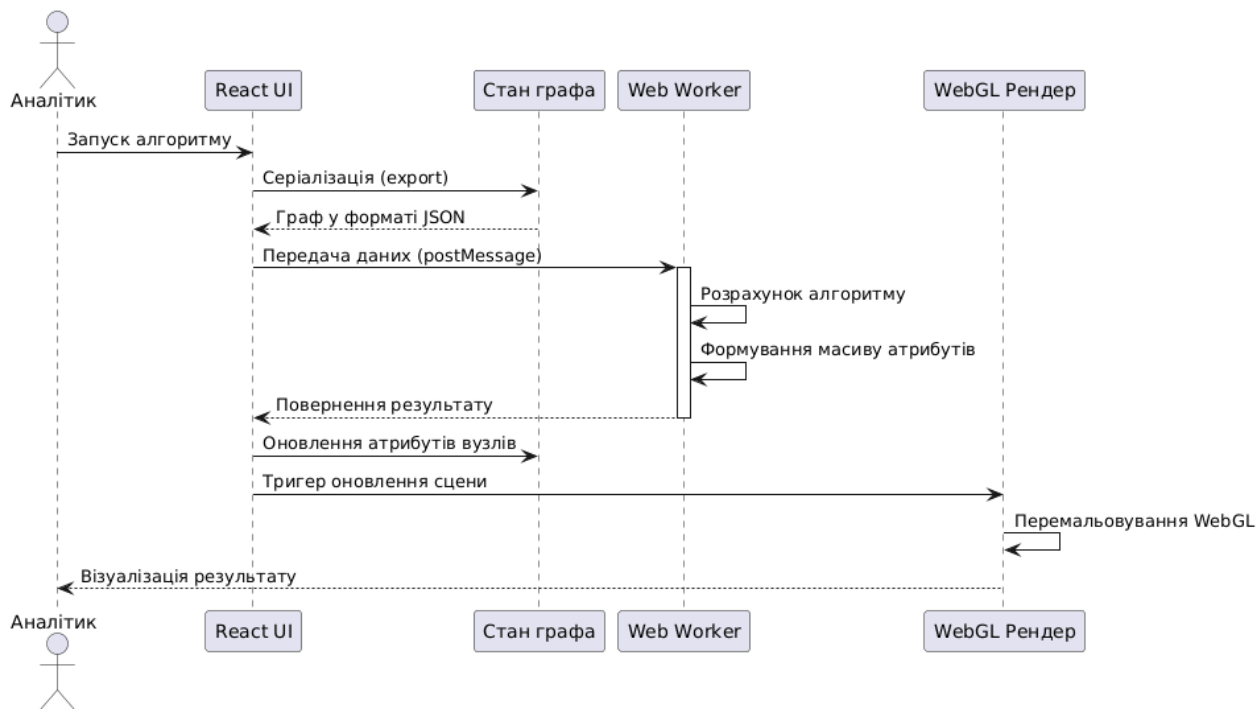


Рис. 2.3. Діаграма послідовностей взаємодії компонентів

Як видно з рисунка 2.3, у процесі асинхронної взаємодії беруть участь п'ять основних ліній життя, безпосередньо Аналітик (користувач), рівень інтерфейсу (React UI), стан графа (Graphology), ізольований фоновий потік (Web Worker) та графічний конвеєр (WebGL Рендер).

Процес ініціюється Аналітиком, який через графічний інтерфейс запускає обраний алгоритм (наприклад, оптимізацію модулярності або розрахунок центральності). Рівень React UI миттєво звертається до поточного стану графа і виконує експорт у формат JSON.

Після отримання даних, головний потік асинхронно передає їх до фонового середовища Web Worker за допомогою системного методу передачі повідомлень. Завдяки такому делегуванню головний потік залишається вільним, що запобігає перевантаженню браузера та гарантує Аналітику плавну взаємодію з інтерфейсом навіть під час розрахунків.

Далі Web Worker переходить в активний стан. У фоні розгортається математична модель, виконуються ітерації обчислень та формується кінцевий масив атрибутів. Оскільки складність таких розрахунків може сягати квадратичних значень, їх ізоляція від DOM-дерева є обов'язковою вимогою продуктивності. Після завершення обчислень Web Worker повертає масив результатів назад до головного потоку.

Отримавши опрацьовані дані, інтерфейс оновлює атрибути вузлів безпосередньо у стані графа та відправляє тригер до компонента WebGL Рендер. На фінальному етапі графічний конвеєр приймає нові масиви координат, оновлює вершинні буфери відеокарти та перемальовує сцену, після чого Аналітик бачить візуалізований результат роботи алгоритму.

2.2. Математичне моделювання та алгоритми компонування графів

Сирі дані не мають просторової форми. Щоб перетворити хаос зв'язків на читабельну структуру, у вебзастосунку реалізовано імплементацію силових алгоритмів. Математична модель симуляції базується на принципах класичної

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		25

фізики. З огляду на повну відмову від серверної інфраструктури, всі обчислювальні процеси декомпоновано та оптимізовано для виконання виключно на стороні клієнтського браузера.

Математичною основою просторового позиціонування вершин графа є імплементація силових алгоритмів, зокрема алгоритму ForceAtlas2 для двовимірного простору та адаптованого фізичного рушія d3-force для тривимірної візуалізації [28].

Математична модель симуляції базується на принципах класичної механіки [29]. Зокрема, використовується закон Кулона (відштовхування), згідно з яким всі вузли графа наділені однойменним віртуальним зарядом і відштовхуються один від одного з силою, обернено пропорційною відстані між ними. Це дозволяє уникнути візуального накладання вершин та рівномірно розподілити їх по площині або об'єму. Також застосовується закон Гука (притягання), де ребра графа виступають у ролі віртуальних пружин, які притягують пов'язані вершини одна до одної із силою, що пропорційна відстані між ними або вазі зв'язку.

Оптимізація обчислень за допомогою технології Web Workers реалізована через те, що розрахунок сили взаємодії за принципом «кожен з кожним» має квадратичну обчислювальну складність $O(N^2)$. Пряме виконання цих операцій призводить до миттєвого блокування основного потоку інтерфейсу користувача (UI thread). Для вирішення цієї проблеми логіку фізичної симуляції було повністю винесено в ізольований фоновий потік. Цей фоновий потік отримує сирі структури даних із логічного ядра Graphology [8]. Обчислення оптимізовано за допомогою квадродерев (для 2D) та октадерев (для 3D) на базі алгоритму Барнса-Хатта (Barnes-Hut), що знижує складність до $O(N \log N)$ [3].

Розраховані просторові координати вершин (X, Y, Z) транслюються назад у головний потік через високошвидкісні типізовані масиви Float32Array. Такий підхід гарантує, що інтерфейс користувача (кнопки, сайдбар, повзунки налаштувань) залишається повністю інтерактивним під час виконання важких математичних розрахунків.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Апаратне прискорення рендерингу інтегровано для забезпечення високої частоти кадрів (стабільних 60 FPS) під час маніпуляцій із графом великої розмірності, оскільки стандартні методи малювання (HTML5 Canvas 2D або SVG) були відкинуті як неефективні. Замість цього використано технологію WebGL реалізовану у двох режимах [12]. Режим 2D-візуалізації фокусується на максимальній продуктивності процесора та відеокarti при екстремальних навантаженнях. У ньому координати та атрибути кольору пакуються у буфери вершин і передаються в пам'ять GPU, а рендеринг контролюється низькорівневими вершинними та фрагментними шейдерами, що дозволяє миттєво обробляти панорамування та масштабування десятків тисяч елементів.

Своєю чергою, режим 3D-візуалізації переносить граф у тривимірний простір, що дозволяє вирішити проблему «волосяної грудки» (Hairball problem) за рахунок додавання третьої просторової осі (Z). У цьому вимірі вершини рендеряться як тривимірні геометричні примітиви (сфери або точки), а ребра — як просторові лінії з налаштовуваною товщиною та прозорістю, що забезпечує візуальне сприйняття топології мережі.

Для аналітичної обробки графа система не звертається до серверних скриптів. Вся логіка інкапсульована у клієнтському ядрі, а обробку та координацію даних покладено на інтегрований аналітичний модуль React-застосунку. Він поєднує в собі модуль топологічних метрик, модуль кластеризації та контролер прикладних бізнес-сценаріїв.

Модуль топологічних метрик обчислює показники центральності вершин за ступенем, що дозволяє динамічно змінювати радіус вузлів у пам'яті застосунку залежно від їхньої важливості в мережі [22]. Математично центральність за ступенем для вершини v у неорієнтованому графі обчислюється як $C_D(v) = \text{deg}(v)$, де $\text{deg}(v)$ - це кількість інцидентних ребер. Для коректного порівняння вузлів у графах різної розмірності цей показник нормалізується відносно загальної розмірності мережі за формулою $C'_D(v) = \frac{\text{deg}(v)}{N-1}$, де N - загальна кількість вершин у мережі.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

Модуль кластеризації реалізує евристичний алгоритм Лувена для виявлення прихованих спільнот на основі оптимізації модулярності графа, з наступним розфарбовуванням вузлів у відповідні кольори [22].

Математично модулярність Q мережі визначається як різниця між фактичною кількістю ребер усередині кластера та очікуваною кількістю таких ребер у випадковому графі з аналогічним розподілом ступенів за формулою

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

де A_{ij} - елемент матриці суміжності, k_i та k_j - ступені відповідних вершин, m - загальна кількість ребер, функція Кронекера $\delta(c_i, c_j)$ дорівнює 1, якщо вершини належать до одного кластера, і 0 інакше. Розрахунок відбувається ітеративно у фоновому потоці, де спочатку проводиться локальна оптимізація (переміщення вузлів до сусідніх спільнот для досягнення максимального приросту ΔQ), а потім агрегація вузлів у макро-вершини.

Повний цикл асинхронної взаємодії компонентів під час виконання важких розрахунків найкраще простежується на прикладі запуску алгоритму Лувена. Коли користувач ініціює аналіз через користувацький інтерфейс, компонент панелі управління викликає функцію «runLouvain». Ця подія змінює стан завантаження на активний, блокуючи повторні виклики. Далі екземпляр графа серіалізується у формат JSON за допомогою методу graph.export і передається до об'єкта Web Worker через метод postMessage. У цей момент виконання переходить у фоновий процес, який ініціалізує математичну модель та виконує ітерації розрахунку модулярності.

Після завершення обчислень у фоні, Worker формує кінцевий об'єкт, де кожному ідентифікатору вузла зіставляється номер кластера та відповідний

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

HEX-код кольору з попередньо заданої палітри. Цей об'єкт повертається в головний потік через подію «message».

Обробник події в головному потоці (Main Thread) ітерує вершини локального графа, застосовуючи нові кольорові атрибути, після чого викликається метод «sigmaRef.current.refresh» для примусового оновлення буферів WebGL та перемальовування Canvas без руйнування самого React-компонента. У разі використання тривимірного режиму, для синхронізації просторових координат використовується функція «syncGraphData3D», загорнута у хук «useCallback». Вона конвертує двовимірні координати у тривимірний простір шляхом масштабування, що забезпечує збереження цілісності топології під час перемикання між 2D та 3D рушіями.

Для дослідницьких цілей інтегровано модель Барабаші-Альберта, яка виступає генератором топологій, заснованим на принципі преференційного приєднання. Вона застосовується у Пісочниці для штучного створення мереж, імітуючи реальне зростання соціального середовища. Ймовірність p_i приєднання нового вузла до існуючого вузла i є пропорційною його поточному ступеню k_i за формулою

$$p_i = \frac{k_i}{\sum_j k_j}$$

Цей математичний метод програмно імітує утворення впливових «хабів» при реальному зростанні соціальних мереж.

Усіма цими інструментами керує контролер прикладних бізнес-сценаріїв. Він координує генерацію та миттєве розгортання спеціалізованих графів (маркетингових моделей, логів фінансових транзакцій для фрод-моніторингу, HR-мереж комунікації). Це дозволяє кінцевому споживачу перемикатися між абстрактним дослідженням та вирішенням конкретних прикладних задач без залучення зовнішніх серверних запитів.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3. Проектування користувацького інтерфейсу та управління станами

Ефективність роботи з великими даними залежить не лише від математичного ядра, а й від ергономіки. При проектуванні візуальної частини застосовано принцип мінімізації когнітивного навантаження: користувач повинен аналізувати граф, а не витрачати час на пошук інструментів. Інтерфейс виконано в темних тонах. Це не просто естетичний вибір, а фізіологічна необхідність: на темному тлі яскраво забарвлені кластери і тонкі напівпрозорі зв'язки сприймаються оком значно чіткіше і з меншим напруженням.

Візуально застосунок розділений на дві зони, масивний контейнер WebGL-рендерингу ліворуч та вузьку панель керування праворуч. Управління інформаційними потоками між цими зонами розроблено на базі деревоподібної ієрархії компонентів (рис. 2.4).

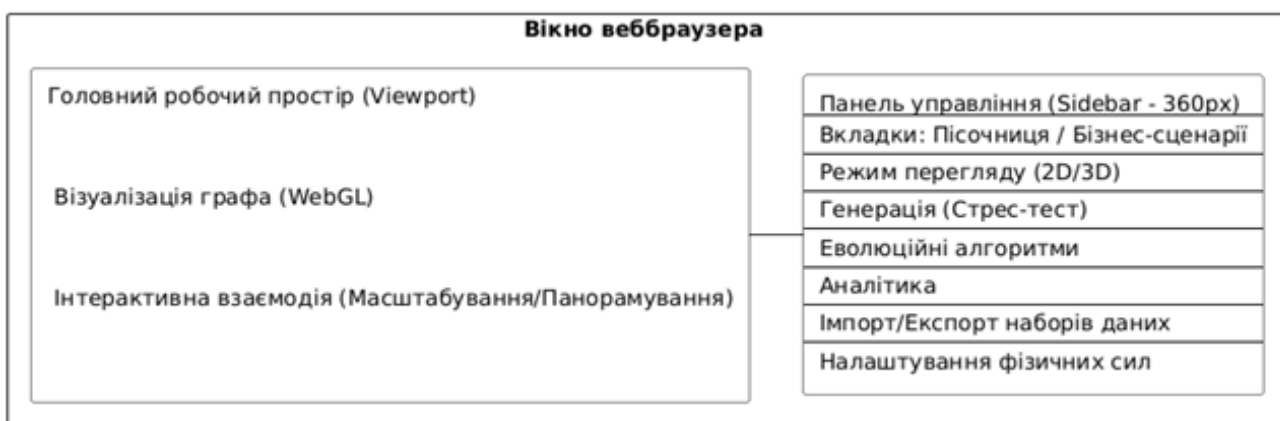


Рис. 2.4. Схематичний макет користувацького інтерфейсу

Як видно з макета на рисунку 2.4, візуальний простір екрана логічно розділено на дві основні зони. Головний робочий простір займає переважну частину екрана та слугує контейнером для WebGL-рендерингу [12]. Для забезпечення максимального контрасту між фоном та елементами графа (особливо при виділенні кластерів яскравими кольорами) було обрано темну кольорову палітру. Це критично знижує навантаження на зір користувача при тривалій аналітичній роботі.

Панель управління закріплена у правій частині екрана. Для побудови її окремих візуальних елементів застосовувалися концепції компонентного дизайну Material UI [13].

Вона забезпечує доступ до всіх інструментів керування застосунком, які згруповані за логічними категоріями у відповідних вкладках. З архітектурної точки зору розробка велася з використанням функціональних React-компонентів. Концептуальну ієрархію функціональних модулів та потоки передачі даних всередині головного компонента зображено на рисунку 2.5.



Рис. 2.5. Структурна схема ієрархії компонентів вебзастосунку

Дана ієрархія ілюструє концепцію єдиного джерела істини (Single Source of Truth). Головний компонент зберігає екземпляр графа та керує загальним станом системи. Зміна налаштувань автоматично передає необхідні дані та обробники подій на дочірні рівні у зону рендерингу та до контролерів сайдбару.

Зона управління розділена на логічні блоки, серед яких модуль парсингу файлів, контролери аналітики та перемикач вкладок. Останній надає доступ до математичних алгоритмів графа, інструментів ручного конструювання мережі, а також параметрів косметики та фізики симуляції.

Основним інструментарієм для побудови клієнтського середовища виступила бібліотека React.js [2]. Вибір продиктований не просто популярністю фреймворку, а гострою необхідністю контролювати стан тисяч вузлів без перезавантаження сторінки.

Візуалізація великих обсягів даних (понад 1000 вузлів та 1500 зв'язків) неминуче створює колосальне навантаження на процесор. Саме тому довелося

прийняти нестандартне архітектурне рішення і повністю ізолювати графічний інтерфейс від математичного ядра.

Структура обробляється окремою бібліотекою `graphology`, а сам інстанс графа закріплено за допомогою хука `useRef`. Завдяки цьому складні ітеративні алгоритми (наприклад, симуляція гравітації) працюють, не провокуючи безперервне перемальовування всього дерева компонентів.

Окремим аспектом управління станами є синхронізація просторових координат при перемиканні між 2D та 3D режимами візуалізації. Цей процес реалізовано через механізм умовного рендерингу. Коли користувач активує 3D-режим, поточний екземпляр рушія `Sigma.js` демонтується з DOM-дерева, а на його місце ініціалізується компонент `ForceGraph3D`.

Для запобігання втраті архітектури графа під час цього переходу розроблено алгоритм синхронізації просторових даних. Він ітерує всі вузли мережі, конвертуючи їхні двовимірні координати у тривимірний простір.

Для осі *Z* застосовується алгоритм випадкового розподілу в заданому діапазоні, що дозволяє миттєво надати мережі об'єму та уникнути накладання вершин. Весь процес синхронізації загорнуто в функцію (хук `useCallback`), що гарантує безшовний перехід між рушіями без необхідності повторного завантаження файлів.

Застосунок архітектурно є `Single Page Application (SPA)`, де навігація обробляється локально за допомогою внутрішнього стану компонентів (`React State`), що робить його розгортання абсолютно незалежним від складних бекенд-інфраструктур [19, 21].

Процес переведення продукту в робоче середовище складається з кількох кроків. Спочатку через термінал ініціюється збирання проєкту командою `npm run build`. На цьому етапі середовище `Node.js` виступає виключно як інструмент локальної збірки: воно компілює та оптимізує всі `React`-компоненти, мініфікує стилі та формує директорію зі статичними файлами [14].

Висновки до другого розділу:

У другому розділі було фундаментально обґрунтовано вибір клієнт-центричної архітектури Local-First для побудови вебзастосунку візуалізації соціальних графів. Повна відмова від серверних баз даних на користь браузерних обчислень усунула мережеві затримки та забезпечила значний рівень безпеки при роботі з комерційними датасетами. Задokumentовано процес локального парсингу форматів JSON та CSV.

Детально розкрито математичний базис системи. Для вирішення проблеми візуального перекриття десятків тисяч елементів застосовано силові алгоритми укладки, які імітують фізичні закони Гука та Кулона. Доведено критичну необхідність оптимізації розрахунків за допомогою структур Барнса-Хатта для збереження плавності роботи інтерфейсу. Проаналізовано топологічний інструментарій системи: застосування метрик центральності для виявлення впливових вузлів, впровадження алгоритму Лувена для кластеризації прихованих спільнот, а також використання моделі Барабаші-Альберта для генерації синтетичних безмасштабних мереж.

У фінальній частині розділу описано принципи ергономічного проектування користувацького інтерфейсу. Застосування ієрархічної структури компонентів та механізмів управління станом дозволило створити реактивний, безшовний Dashboard. Оптимізація пам'яті через ізоляцію життєвих циклів математичного ядра та WebGL-конвеєра гарантувала стабільну роботу застосунку під екстремальними навантаженнями, перетворивши звичайний веббраузер на потужне аналітичне середовище.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

РОЗДІЛ 3. ПОРЯДОК РОБОТИ, РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1. Базові сценарії візуалізації та управління даними

На етапі первинного завантаження програмного комплексу відбувається автоматичне генерування базової топологічної моделі у вигляді випадкового графа. Відповідно до початкових конфігураційних параметрів, графічне ядро системи за замовчуванням ініціалізує тривимірний (3D) простір візуалізації. Застосування такого алгоритму дозволяє здійснювати попередню оцінку продуктивності системи та тестування інструментів навігації без необхідності обов'язкового імпортування зовнішніх наборів даних (рис. 3.1).

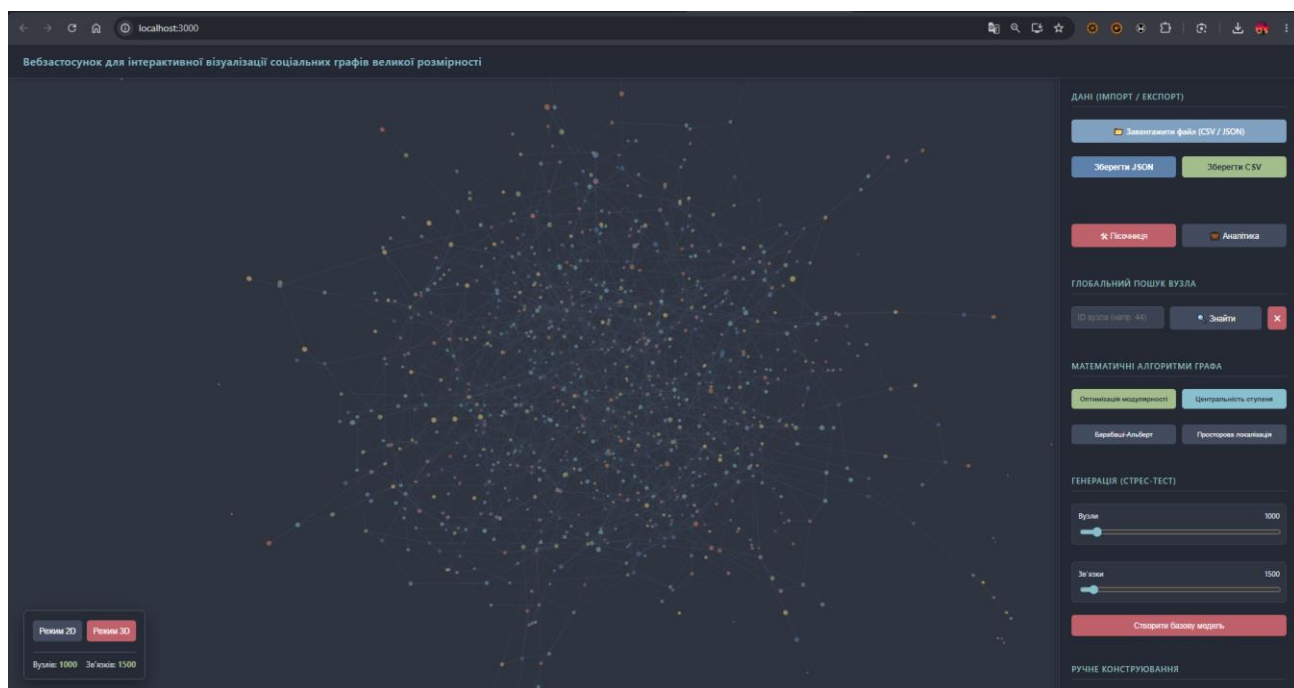


Рис. 3.1. Головна сторінка веб застосунку, відображення графа у 3D-режимі

У лівому нижньому куті інтерфейсу користувача розташовано інформаційну панель, основним завданням якої є безперервний моніторинг розмірності завантаженого набору даних. Цей елемент управління дозволяє фіксувати поточну кількість вузлів та зв'язків під час виконання операцій фільтрації або

кластеризації. Дані на панелі оновлюються синхронно зі змінами у графічному ядрі застосунку (рис. 3.2).

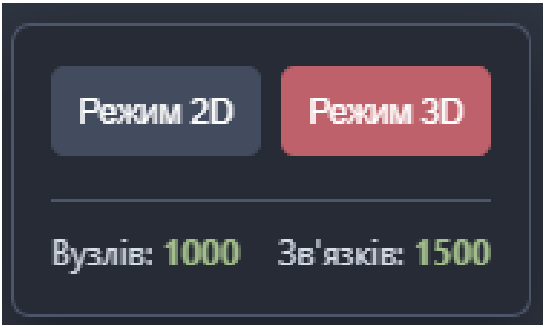


Рис. 3.2. Плаваюча панель навігації та моніторингу

Аналітик може у будь-який момент перемкнути вимірність сцени. Натискання на відповідні кнопки інтерфейсу («Режим 2D» або «Режим 3D») змінює внутрішній стан компонента, що за допомогою хуків життєвого циклу автоматично ініціює виклик функції `triggerRenderUpdate`. Система виконує двовимірну проєкцію просторових координат та передає рендеринг бібліотеці `sigma.js` [24] (рис. 3.3).



Рис. 3.3. Відображення соціального графа у двовимірному 2D-режимі

Проведення аналізу вимагає можливості інтеграції зовнішніх наборів даних. З цією метою верхній блок панелі керування виділено під операції введення та виведення інформації. Цей модуль слугує основною точкою входу, дозволяючи

завантажувати дослідницькі масиви даних у систему для подальшого опрацювання (рис. 3.4).

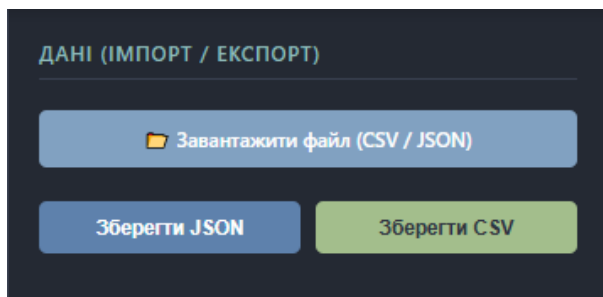


Рис. 3.4. Інтерфейсний блок управління даними

Елемент імпорту ініціює виклик стандартного вікна операційної системи для вибору локальних файлів. Розроблений вебзастосунок підтримує роботу з форматами збереження мережевих структур .json та .csv. Після вибору відповідного файлу автоматично розпочинається процес локальної десеріалізації та перевірки вхідних даних (рис. 3.5).

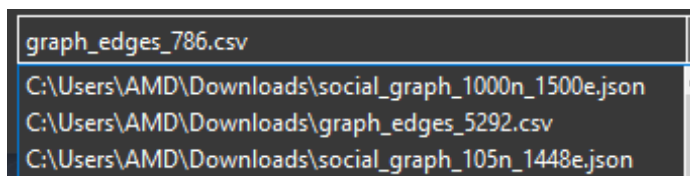


Рис. 3.5. Діалогове вікно вибору зовнішнього датасету

Для збереження результатів обробки в системі реалізовано модуль експорту даних. Він формує необхідну структуру безпосередньо на стороні клієнта і генерує кінцевий файл у форматі JSON або CSV. Такий підхід забезпечує збереження обчислених топологічних метрик та просторових координат вершин (рис. 3.6).

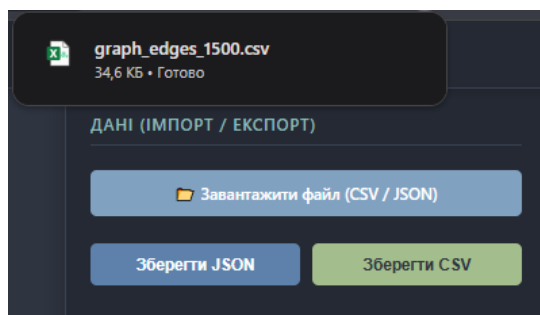


Рис. 3.6. Ініціалізація вивантаження результатів у форматі CSV

Ручний візуальний пошук конкретного об'єкта серед значного масиву елементів є нераціональним. З метою оптимізації процесу навігації було впроваджено алгоритм глобального пошуку по графу. Користувач має змогу ввести ідентифікатор цільової вершини у текстове поле, після чого система здійснює обхід структури даних (рис. 3.7).

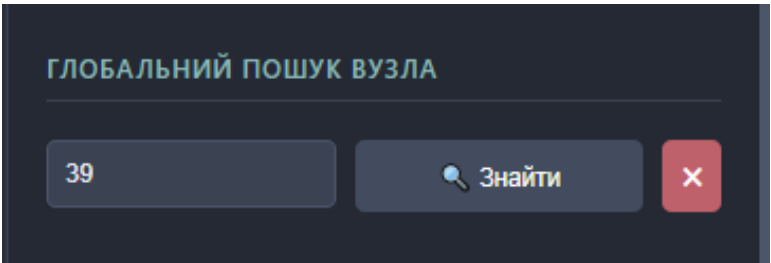


Рис. 3.7 Поле глобального пошуку вузла

При успішному пошуку система автоматично фокусується на цільовому вузлі. Для нього та всіх його інцидентних ребер застосовується контрастне виділення. Одночасно з цим для решти елементів графа встановлюється високий рівень прозорості, що дозволяє візуально відокремити досліджуваний об'єкт та суміжні вершини від загальної мережі (рис. 3.8).

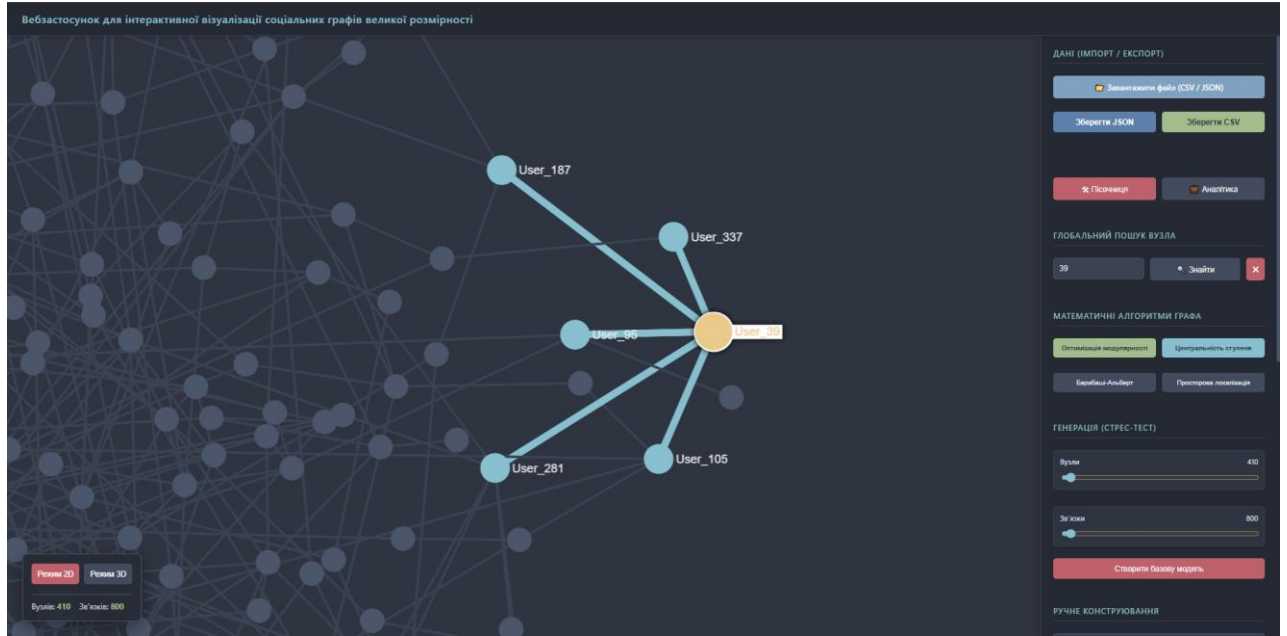


Рис. 3.8. Фокусування на об'єкті та затінення непов'язаних елементів

3.2. Робота з аналітичними модулями та налаштування простору

Для вирішення спеціалізованих прикладних задач у межах розробленої системи передбачено окремий аналітичний інструментарій. Відповідна програмна логіка згрупована у спеціальному розділі користувацького інтерфейсу, що забезпечує зручний доступ до функцій топологічного аналізу. Архітектурно цей функціонал поділено на три незалежні підсистеми (модулі), кожна з яких орієнтована на виконання конкретного класу алгоритмічних розрахунків відповідно до галузевої специфіки (рис. 3.9).

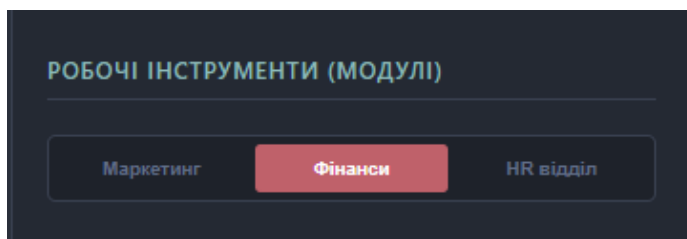


Рис. 3.9. Навігаційна панель аналітичних модулів

Аналітичний модуль «Маркетинг» орієнтований на виявлення найвпливовіших вузлів усередині досліджуваної мережі. Для початку аналізу встановлюється поріг мінімальної кількості зв'язків. Цей параметр слугує критерієм відокремлення цільових вершин від решти елементів графа (рис. 3.10).

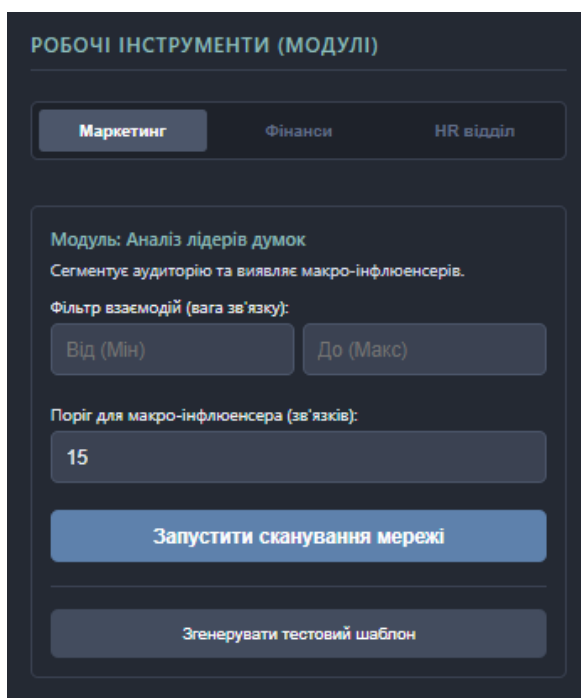


Рис. 3.10. Налаштування параметрів маркетингового сканування

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Програмний алгоритм обчислює ступінь центральності для кожної вершини в завантаженій мережі [22]. Якщо показник вузла перевищує встановлений поріг, система автоматично збільшує його масштабний коефіцієнт відображення. Додаткове контрастне виділення пришвидшує сегментацію об'єктів з найбільшою кількістю інцидентних ребер (рис. 3.11).

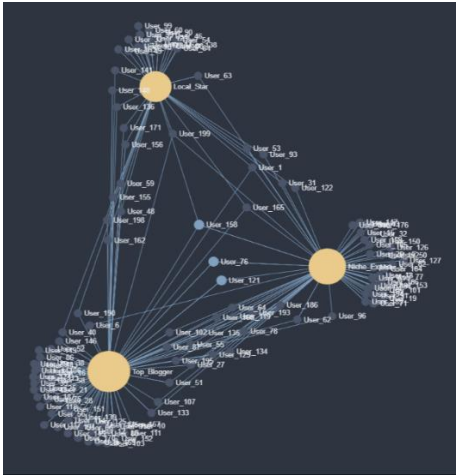


Рис. 3.11. Візуалізація макро-інфлюенсерів у соціальній мережі 2D

Модуль «Фрод-контроль та транзакції» розроблено для виявлення нетипової фінансової активності та потенційно аномальних транзакцій. З цією метою впроваджено функціонал для виокремлення щільних груп взаємодій між об'єктами. Для його реалізації застосовується математичний метод кластеризації Лувена, що групує вершини на основі щільності їхніх взаємодій [22] (рис. 3.12).

РОБОЧІ ІНСТРУМЕНТИ (МОДУЛІ)

Маркетинг

Фінанси

HR відділ

Модуль: Фрод-контроль та транзакції

Фільтр за сумою транзакції (€):

Від (Мін)

До (Макс)

Розмір підозрілої групи (кількість рахунків):

3

50

Виявити тіньові схеми

Згенерувати тестовий шаблон

Рис. 3.12. Управління параметрами фінансового моніторингу

Якщо в процесі обчислень виявляється замкнений кластер, що відповідає заданим параметрам розміру (наприклад, від 3 до 50 вузлів), система реагує відповідним чином. Усім елементам, які належать до цієї групи, автоматично призначається сигнальний маркер. Це дозволяє звернути увагу на виявлену сукупність об'єктів (рис. 3.13).

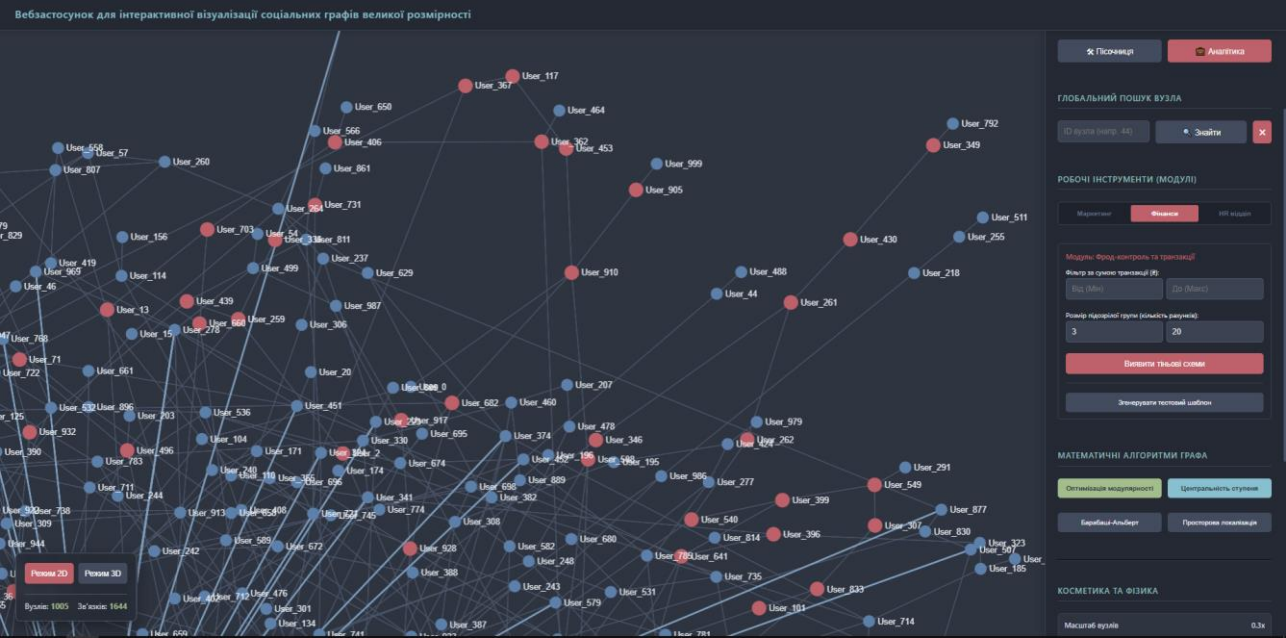


Рис. 3.13. Локалізація тіньових фінансових схем у мережі

У даному аналітичному модулі також реалізовано динамічний фільтр вагових коефіцієнтів ребер. Усі зв'язки, значення яких виходять за межі встановленого ліміту, отримують показник прозорості, що дорівнює нулю, і приховуються з робочої області. Це дозволяє оптимізувати відображення топології графа (рис. 3.14).

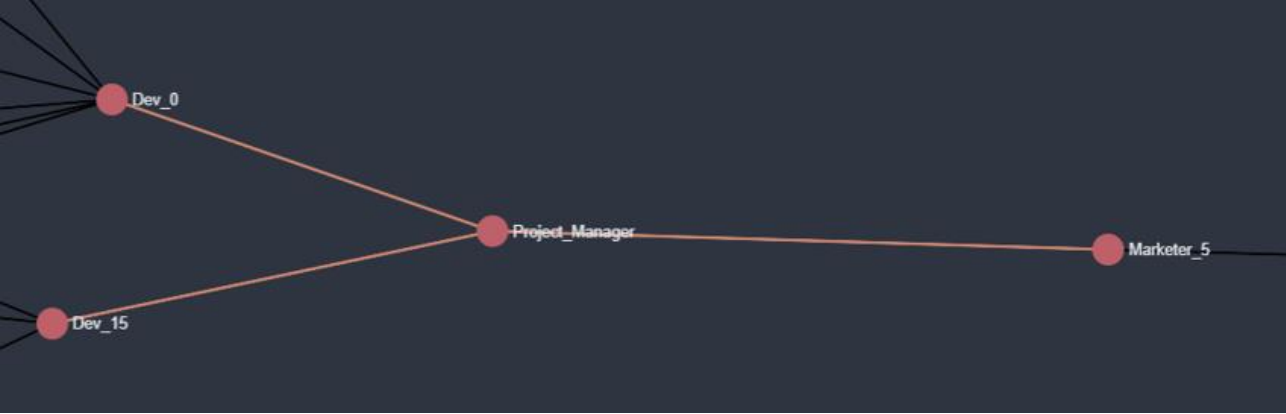


Рис. 3.14. Відокремлення ліміту транзакції

Для детальнішого дослідження активності конкретного вузла передбачено можливість взаємодії з ним безпосередньо у робочому просторі. У відповідь застосунок генерує інформаційну панель, що агрегує історію зв'язків обраного елемента. З метою покращення сприйняття вхідні та вихідні ребра позначаються різними кольоровими ідентифікаторами (рис. 3.15).

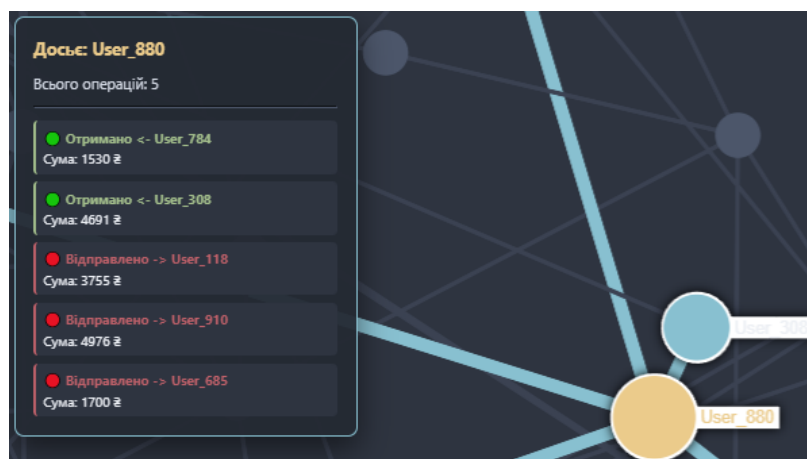


Рис. 3.15. Агреговане інформаційне досьє цільового рахунку

Аналітичний модуль для дослідження корпоративної структури призначений для аналізу внутрішніх комунікацій підприємства. На початку роботи задається бажаний розмір підмножин та встановлюється поріг зв'язків, який система використовує для ідентифікації ключових вершин. Програмний комплекс аналізує ці дані, формуючи ієрархічну модель (рис. 3.16).

РОБОЧІ ІНСТРУМЕНТИ (МОДУЛІ)

Маркетинг Фінанси **HR відділ**

Модуль: Корпоративна структура
Виявляє ізольованих співробітників та ключових менеджерів.

Розмір відділу (кількість осіб):

Поріг для статусу Менеджера (зв'язків):

Картувати відділи

Згенерувати тестовий шаблон

Рис. 3.16. Налаштування параметрів картування компанії

На основі заданих критеріїв алгоритм поділяє загальну мережу на окремі кластери. Ізольовані вузли з мінімальною кількістю комунікацій відображаються нейтрально. Разом з тим вузли з найвищим ступенем зв'язності масштабуються в межах своєї підмножини, що дає змогу оцінити управлінську структуру (рис. 3.17).

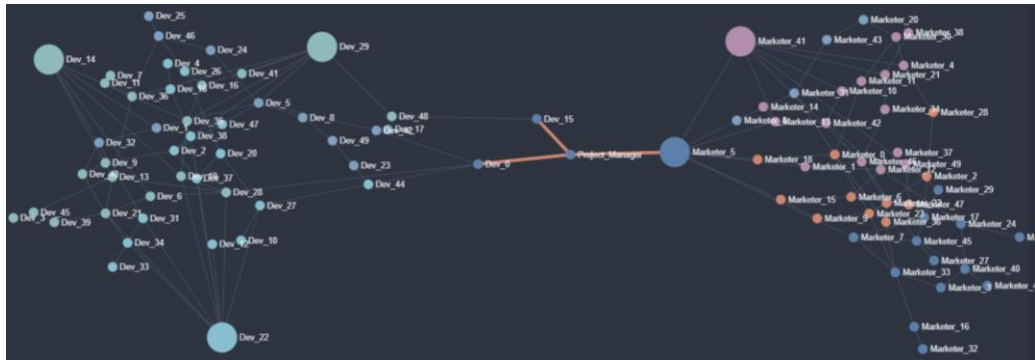


Рис. 3.17. Кластеризація корпоративних відділів підприємства

Функція оптимізації модулярності ініціює виконання математичного методу Лувена у фоновому потоці обчислень. Цей алгоритм ітеративно визначає оптимальне розбиття загальної мережі на спільноти і автоматично призначає їм ідентифікаційні кольори. Це сприяє виявленню підграфів із найвищим рівнем взаємодії (рис. 3.18).



Рис. 3.18. Результат оптимізації модулярності мережі

Функція розрахунку центральності за ступенем дозволяє перерозподіляти розміри вузлів відповідно до кількості їхніх інцидентних ребер. Даний математичний підхід забезпечує збільшення площі відображення для тих елементів мережі, які мають найвищий ступінь. Це дозволяє оцінити топологічну ієрархію системи (рис. 3.19).

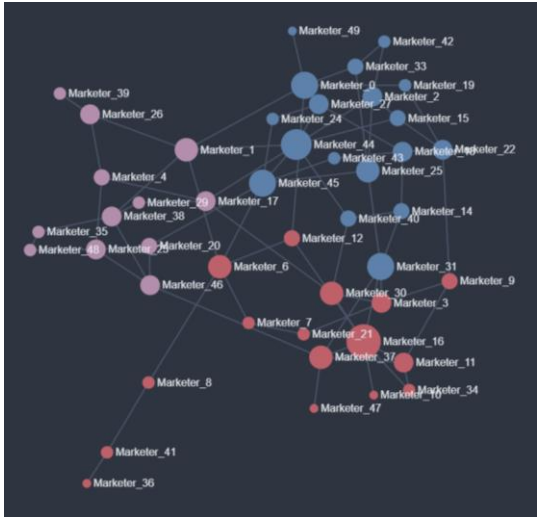


Рис. 3.19. Масштабування об'єктів за математичною центральністю

Інтегрований алгоритм Барабаші-Альберта застосовується для генерації синтетичних безмасштабних мереж. Він програмно створює нові вершини високого ступеня і пов'язує їх з існуючою топологією за математичним правилом преференційного приєднання. Це надає можливість моделювати еволюційне зростання мереж та тестувати стабільність застосунку (рис. 3.20).

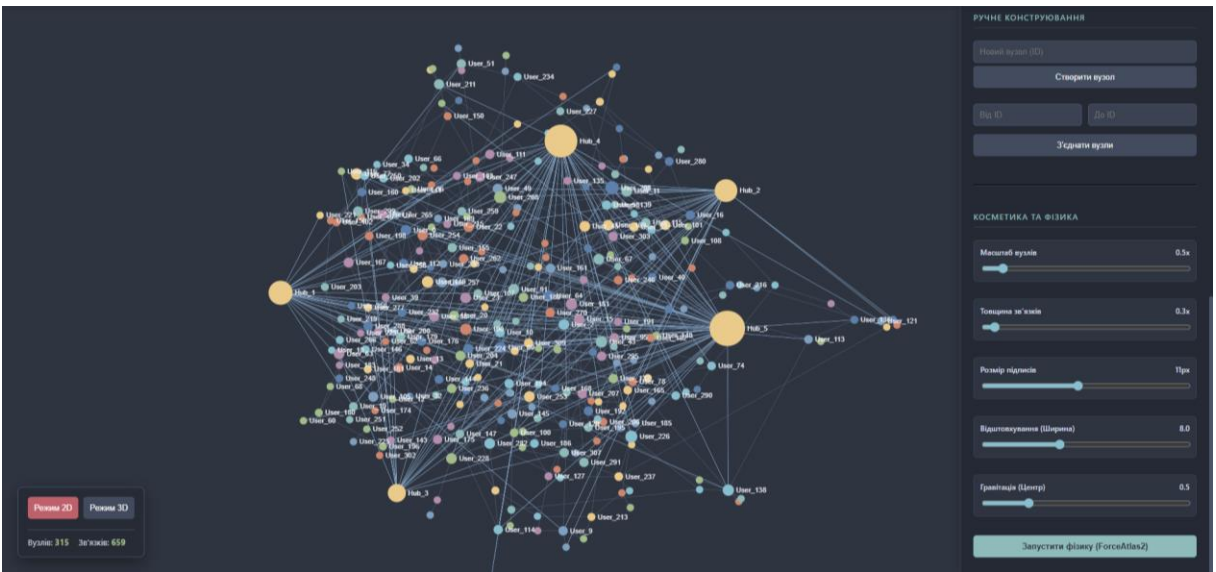


Рис. 3.20. Трансформація графа за моделлю Барабаші-Альберта

Алгоритм просторової локалізації перевіряє тривимірні координати об'єктів під час виконання симуляції. Якщо два непов'язані вузли перебувають на мінімальній відстані один від одного у віртуальному просторі, система реагує на цю подію. В результаті між ними автоматично формується нове структурне ребро (рис. 3.21).

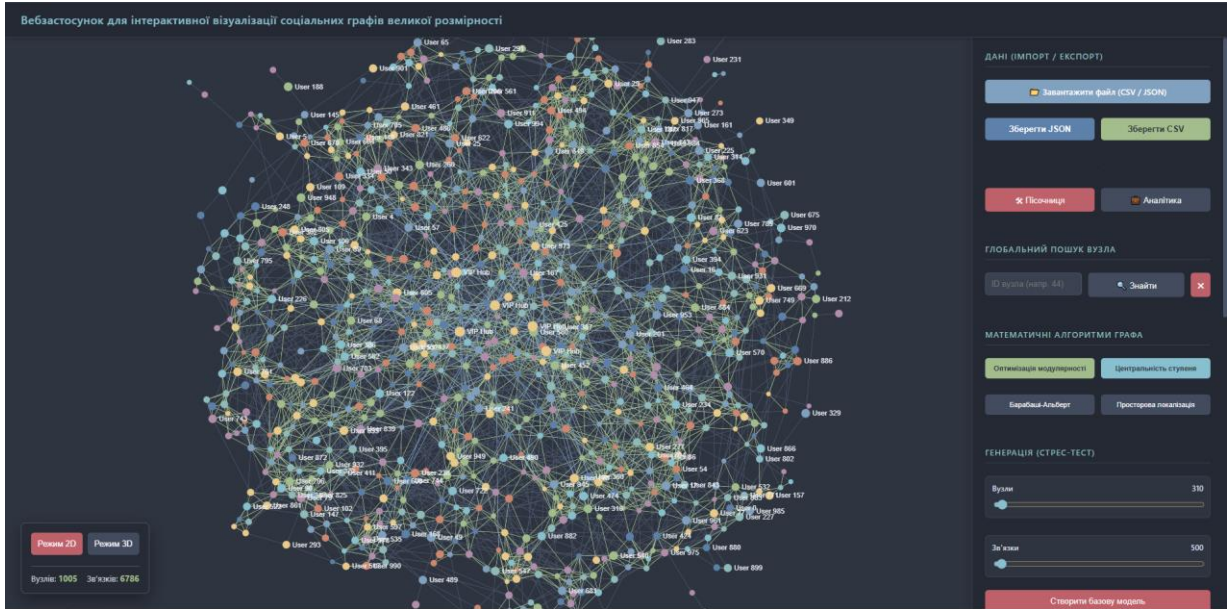


Рис. 3.21. Утворення зв'язків на основі просторової близькості

З метою перевірки обчислювальних ресурсів обладнання та стабільності графічного ядра розроблено модуль стрес-тестування. Він дозволяє навантажити систему згенерованими даними великої розмірності. Цей підхід виключає необхідність імпорту зовнішніх наборів даних (рис. 3.22).

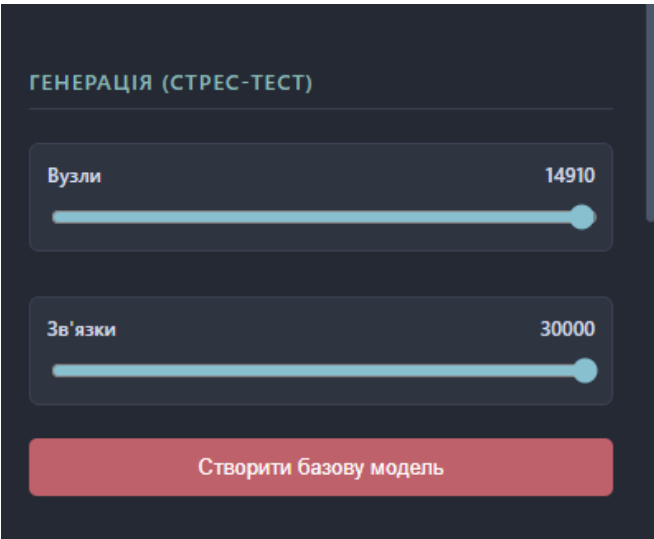


Рис. 3.22. Налаштування генератора стрес-тестування

За допомогою елементів керування користувач може встановити кількість згенерованих вузлів та зв'язків. Блок ручного конструювання призначений для точкового редагування топології мережі. Цей інструмент дозволяє створити нову вершину із заданим ідентифікатором для подальшого аналізу (рис. 3.23).

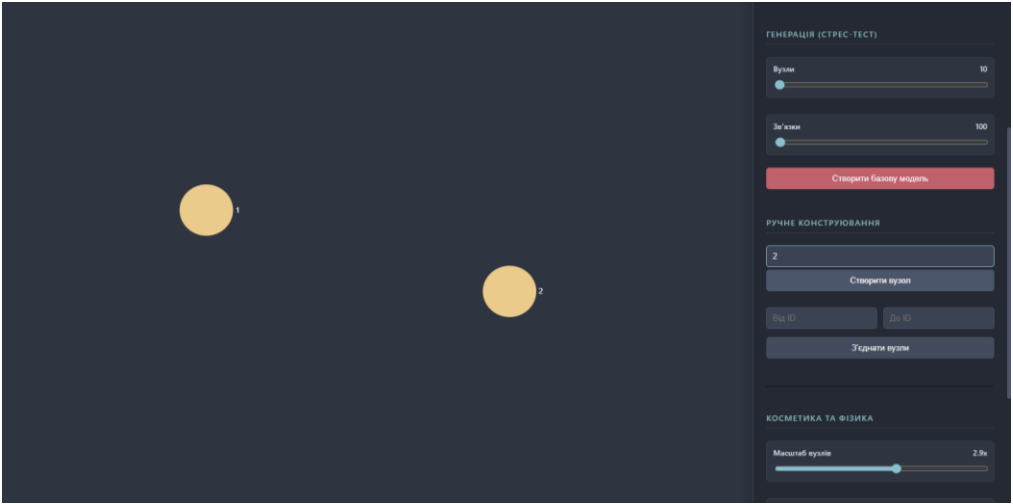


Рис. 3.23. Ініціалізація нового вузла в системі

Після ініціалізації нових об'єктів у системі створюється напрямлене ребро шляхом введення ідентифікаторів джерела та цілі. Ця функція забезпечує розширення існуючої топології без редагування вихідних файлів. Встановлення нових зв'язків між існуючими вершинами відбувається безпосередньо через інтерфейс користувача (рис. 3.24).

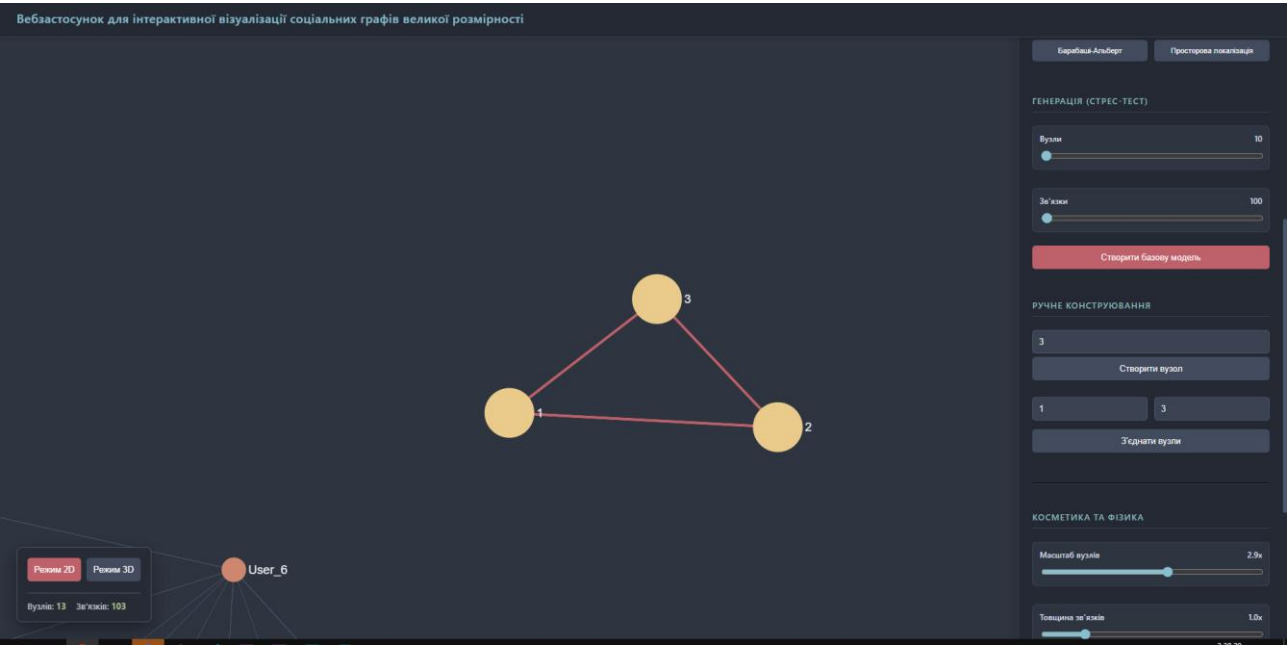


Рис. 3.24. Встановлення нового зв'язку між об'єктами

Для налаштування просторового розподілу об'єктів розроблено спеціалізований блок керування фізичними параметрами компонування. Цей модуль забезпечує пряму взаємодію з динамічним станом симуляції у режимі реального часу. Такий підхід дозволяє коригувати сили гравітації та відштовхування безпосередньо під час виконання алгоритму, виключаючи необхідність повного перезавантаження робочого середовища (рис. 3.25).

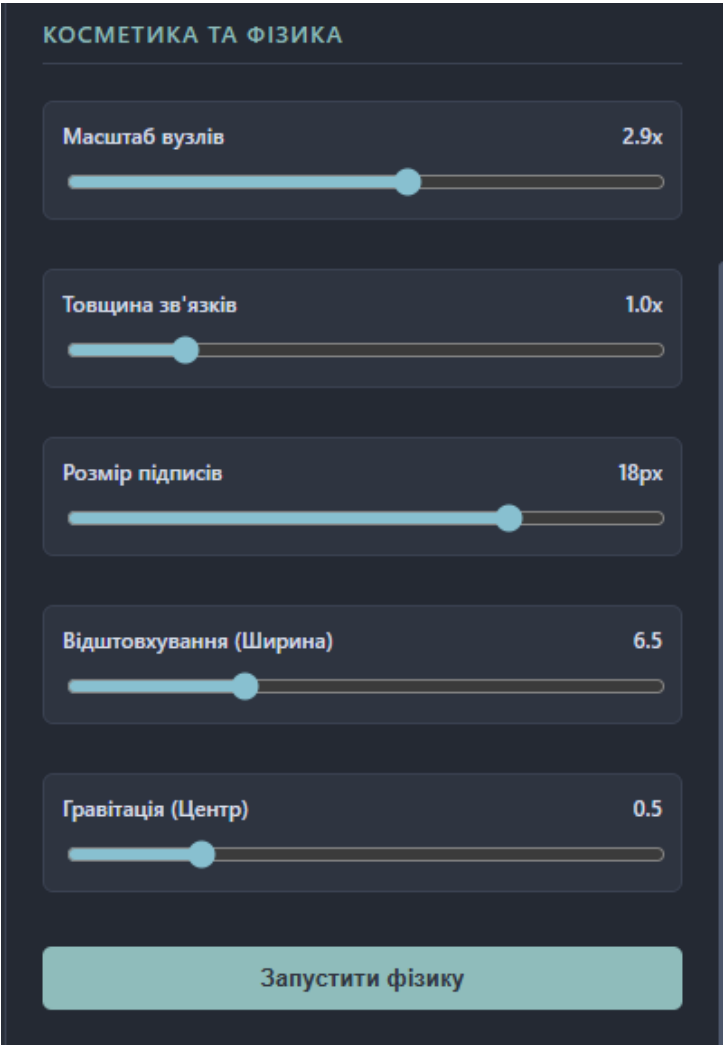


Рис. 3.25. Блок управління візуальними властивостями та фізичними силами

Механізм глобального масштабування забезпечує пропорційну трансформацію метричних характеристик графічних примітивів, зокрема радіусів вершин та товщини ребер. Використання даного інструменту мінімізує ймовірність візуального перекриття об'єктів під час дослідження графів великої розмірності із високим ступенем щільності зв'язків. Відповідне просторове

коригування є об'єктивною необхідністю для збереження цілісності сприйняття топологічної моделі на пристроях виведення з різною роздільною здатністю (рис. 3.26).

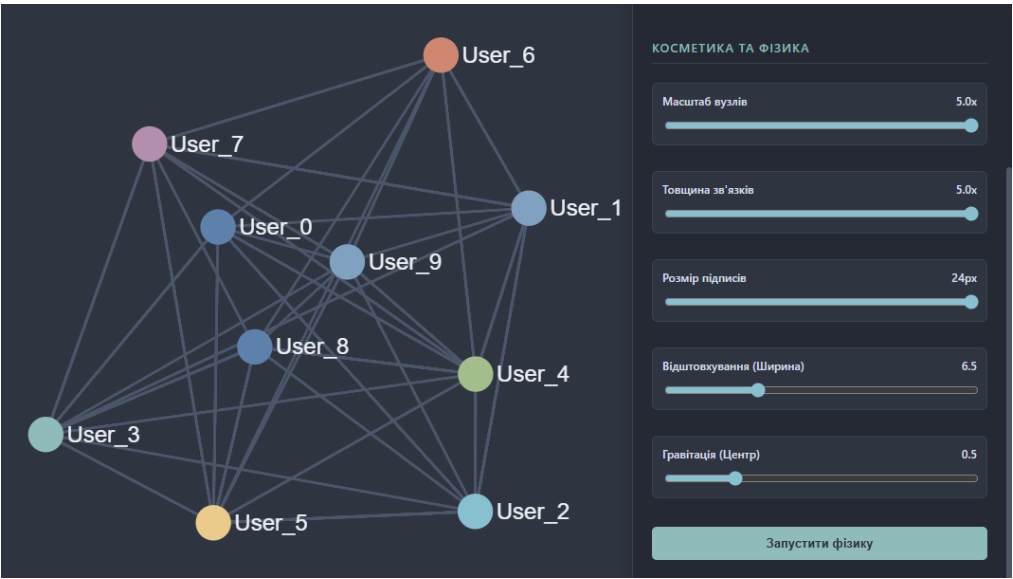


Рис. 3.26. Результат збільшення масштабу

Запуск силового алгоритму компоновання в двовимірному просторі активує процес оптимізації розташування об'єктів. Під час симуляції обчислювальне ядро циклічно перераховує вектори сил взаємодії. Внаслідок цього мережа набуває впорядкованої структури, а пов'язані групи вузлів формують чітко відокремлені кластери (рис. 3.27).

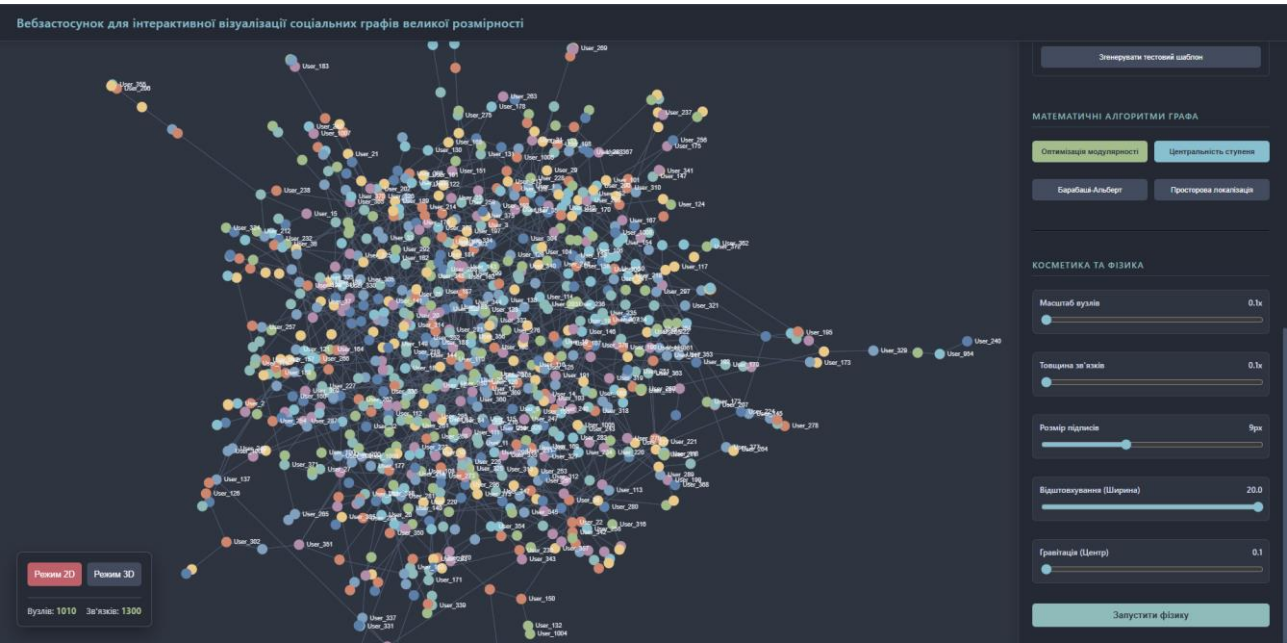


Рис. 3.27. Процес об'ємної оптимізації простору під час симуляції фізики

3.3. Тестування стабільності системи та перевірка сценаріїв

Для всебічної перевірки розробленого програмного рішення було проведено два етапи тестування: навантажувальне (для оцінки продуктивності графічного ядра) та функціональне (для перевірки логіки роботи та стійкості до некоректних вхідних даних). Метою навантажувального тестування був пошук межі апаратних можливостей клієнтської машини при рендерингу великих масивів даних без залучення серверних потужностей. Тестування виконувалося на персональному комп'ютері з такими характеристиками:

Процесор: AMD Ryzen 7 5700x (4.9 ГГц)

Оперативна пам'ять : 32 ГБ

Відеокарта (GPU): AMD Radeon RX 580

Середовище: Google Chrome (версія 149.0.7827.156 26)

Тестова методика передбачала генерацію масивного синтетичного графа, що складався з 15 000 вузлів та 30 000 ребер. Генерація здійснювалася за допомогою вбудованого інструменту «Пісочниця». Для забезпечення об'єктивності вимірювань кожен тестовий сценарій повторювався 5 разів, після чого фіксувалося середнє значення. Щоб уникнути викривлення результатів через фонові процеси операційної системи, використовувалося ізольоване середовище, браузер працював у режимі «Інкогніто», усі сторонні розширення були вимкнені, а єдиною відкритою вкладкою була сторінка локального вебзастосунку.

Вимірювання кадрової частоти та споживання оперативної пам'яті здійснювалося за допомогою вбудованого інструментарію профілювання Chrome DevTools. Симуляція фізики просторового компонування (ForceAtlas2) запускалася з такими базовими параметрами: гравітація - 0.5, відштовхування - 8.0, масштаб - 1.0x. Результати замірів зафіксовано у таблиці 3.1.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		48

Результати навантажувального тестування

Стан системи	Споживання RAM	Кадрова частота	Характер навантаження
Первинна генерація графа	122 МБ	55-74 FPS	Короткочасне навантаження на CPU під час створення об'єктів.
Статичний рендеринг (2D)	102 МБ	68 FPS	Мінімальне навантаження. GPU у стані спокою
Статичний рендеринг (3D)	255 МБ	40-60 FPS	Базове навантаження WebGL-контексту Three.js
Маніпуляція камерою (2D)	145 МБ	55-67 FPS	Легке навантаження на GPU
Маніпуляція камерою (Обертання 3D)	461 МБ	35-45 FPS	Середнє навантаження на GPU для відмальовування 3D-сфер та ліній
Симуляція фізики ForceAtlas2 (2D)	602 МБ	15-20 FPS	Критичне вузьке місце обчислень на CPU
Симуляція просторової гравітації (3D)	240 МБ	16-22 FPS	Максимальне навантаження на CPU та GPU

Для оцінки продуктивності графічного ядра здійснюється безперервний моніторинг використання апаратних ресурсів. Відповідний програмний інструментарій дозволяє відслідковувати поточні показники кадрової частоти та

обсяг споживання оперативної пам'яті системою. Результати вимірювання цих характеристик під час виконання алгоритмів наведено на рисунку 3.28.

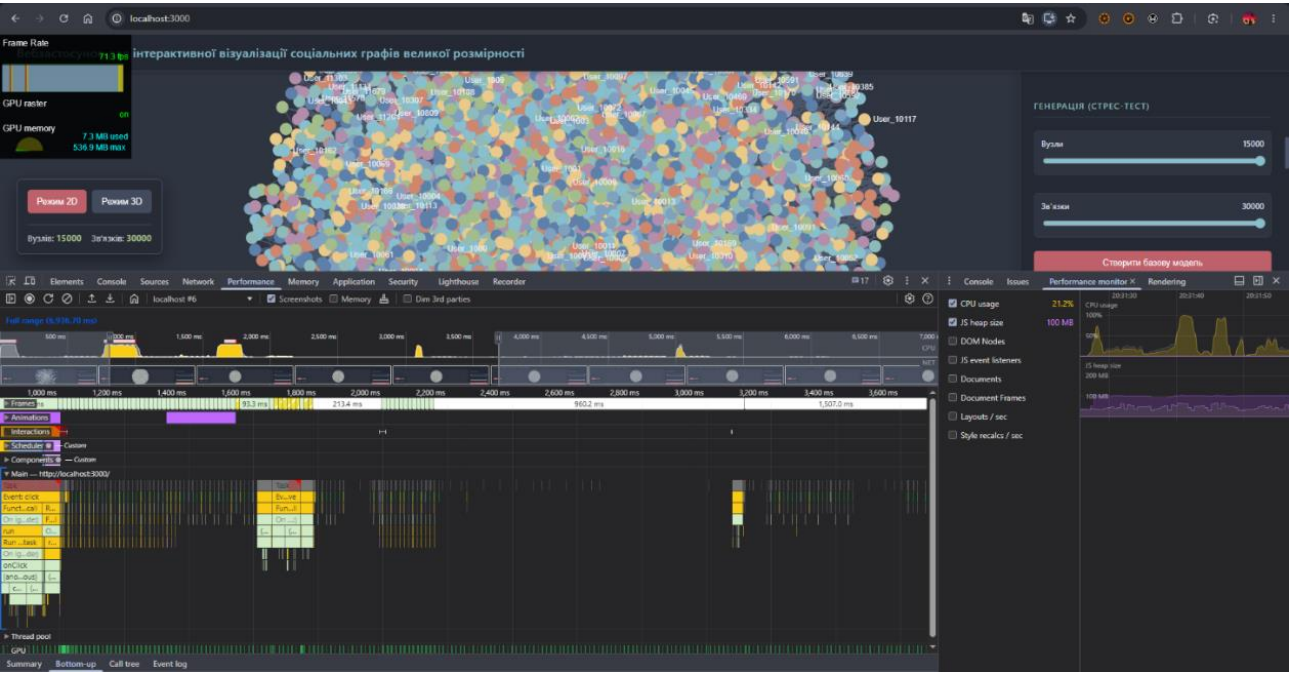


Рис. 3.28. Моніторинг оперативної пам'яті та кадрової частоти

Емпіричні дані підтвердили, що критичним вузьким місцем архітектури є обчислювальна складність ітеративних розрахунків алгоритму ForceAtlas2. Загалом система показала працездатність на тестових наборах до 15 000 вузлів і 30 000 ребер за вказаних умов. Наступним етапом стало функціональне тестування, спрямоване на перевірку коректності роботи заявлених модулів та стійкості до негативних сценаріїв (таблиця 3.2).

Таблиця 3.2

Результати функціонального та негативного тестування сценаріїв

Сценарій тестування	Вхідні дані / Дія користувача	Очікуваний результат	Фактичний результат	Статус
Імпорт валідного CSV (Функціональний)	Таблиця на 5000 рядків із наявними колонками Source та Target.	Побудова графа, генерація координат	Граф успішно відрендерено, зв'язки встановлено коректно.	Успішно

Продовження таблиці 3.2

Запуск алгоритму Лувена (Функціональний)	Граф із 1000 вузлів. Натискання кнопки оптимізації модулярності.	Розбиття мережі на спільноти, збереження відклику UI.	Вузли згруповано за кольорами, інтерфейс не заблоковано.	Успішно
Експорт результату (Функціональний)	Натискання "Зберегти JSON" після розфарбовування вузлів.	Завантаження файлу з метаданими (колір, розмір).	Файл збережено на диск із повним збереженням стану графа	Успішно
Некоректний JSON (Негативний)	Файл .json із порушеним синтаксисом.	Перехоплення помилки парсером, блокування імпорту.	Виведено сповіщення: «Помилка читання JSON файлу!».	Успішно
CSV без Source/Target (Негативний)	Завантаження таблиці, де відсутні ключові стовпці напрямку зв'язку.	Ігнорування помилкових рядків, запобігання краху WebGL.	Рядки проігноровано, помилок рендерингу не зафіксовано.	Успішно
Порожній файл (Негативний)	Завантаження файлу розміром 0 байт.	Відхилення операції обробки.	Процес імпорту перервано, система очікує валідний файл.	Успішно
Створення дубліката (Негативний)	Спроба через інтерфейс створити вузол з ID, який вже існує.	Захист цілісності матриці, скасування створення.	Сповіщення: «Некоректний ID або такий вузол вже існує».	Успішно
Перемикання 2D/3D під час фізики (Негативний)	Клік на «Режим 3D» під час активного перерахунку сил ForceAtlas2.	Зупинка ітерацій 2D, безпечна конвертація координат у 3D.	Алгоритм зупинено, граф відрендерено у тривимірному просторі.	Успішно

Повторна локалізація (Негативний)	Натискання «Просторова локалізація» на вже ідеально згрупованому графі.	Блокування дублювання ребер між сусідніми вузлами.	Сповіщення: «Немає близьких вузлів для утворення кластерів».	Успішно
Пошук неіснуючого вузла (Негативний)	Введення ідентифікатора, якого немає в датасеті, у глобальний пошук.	Запобігання помилкам фокусування камери.	Сповіщення: «Вузли не знайдені».	Успішно

За результатами комплексного тестування доведено стійкість розробленої системи як під час виконання регламентованих аналітичних операцій, так і при опрацюванні некоректних вхідних структур даних. Програмний комплекс містить механізми обробки виняткових ситуацій безпосередньо на клієнтському рівні. Це забезпечує функціонування застосунку та запобігає втраті цілісності просторових координат.

Висновок до третього розділу

Збирання проєкту та розгортання статичного пакета на хмарний хостинг повністю закрили питання безперебійного доступу користувачів. Використання ізоляції математичного ядра від реактивного стану інтерфейсу виявилось архітектурною необхідністю, оскільки інакше DOM-дерево зазнавало б критичних перевантажень під час ітеративного перемальовування тисяч вершин.

Інтерфейс стабільно функціонує у двох просторах візуалізації WebGL для плоских 2D-мереж та Three.js для об'ємних 3D-моделей. Процес синтаксичного аналізу та структурування вхідних наборів даних у форматах CSV та JSON оптимізовано з метою мінімізації обчислювальних затримок. Інтегрований

модуль пошуку дозволяє автоматично позиціонувати камеру на цільовій вершині з одночасним застосуванням ефекту затемнення неактивних елементів.

Структура програмного комплексу включає три спеціалізовані модулі для вирішення прикладних аналітичних задач. Підсистема маркетингового аналізу забезпечує виявлення ключових вузлів впливу на основі розрахунку метрики центральності за ступенем. Модуль виявлення аномальних транзакцій використовує математичний алгоритм Лувена для локалізації щільних груп, тоді як підсистема аналізу корпоративних комунікацій здійснює структурну кластеризацію мережі.

Навантажувальне тестування на масивах із 15 000 об'єктів дозволило виявити та усунути вузькі місця. Перевантаження головного потоку було вирішено шляхом лімітування фонових ітерацій фізики та перенесення ресурсомістких обчислень до Web Workers. У результаті система показала стабільну працездатність на тестових наборах великої розмірності.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		53

ВИСНОВОК

Підсумком виконання кваліфікаційної роботи стала розробка вебзастосунку для інтерактивного аналізу багатовимірних соціальних графів. Було доведено ефективність перенесення обчислювальної бази на клієнтську сторону через архітектуру Single Page Application. Для запобігання надлишковому маніпулюванню DOM-деревом під час симуляції фізики граф ізольовано в математичне ядро бібліотеки Graphology. Застосунок поєднує два рушії відображення: для плоских топологічних метрик, інтегровано WebGL-орієнтовану бібліотеку Sigma.js. Якщо ж потрібен імерсивний огляд щільних кластерів, система перемикається на Three.js (react-force-graph) і розгортає об'ємну 3D-модель.

Процес синтаксичного аналізу та структурування вхідних наборів даних у форматах CSV та JSON було оптимізовано для мінімізації обчислювальних затримок. Інтегрований модуль пошуку дозволяє автоматично позиціонувати камеру на цільовій вершині з одночасним застосуванням ефекту затемнення неактивних елементів.

Практичне значення роботи полягає у створенні трьох аналітичних модулів. Маркетинговий інструмент дозволяє виявляти лідерів думок за допомогою метрики центральності. Модуль фрод-контролю знаходить ізольовані групи транзакцій алгоритмом кластеризації Лувена. HR-модуль візуалізує корпоративну структуру відділів. Додатково інтегровано математичний генератор на базі моделі Барабаші-Альберта та алгоритм імітації просторової локалізації об'єктів.

Завдяки інженерній оптимізації вдалося усунути перевантаження браузера шляхом обмеження ітерацій алгоритму ForceAtlas2. Проблема втрати просторових координат під час перемикавання вимірів вирішена синхронізацією станів. Як результат, розроблений інструмент стабільно витримує навантаження та демонструє високу працездатність на великих тестових наборах даних.

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		54

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Dobing B., Parsons J. How UML is used. Communications of the ACM. 2006. Vol. 49, No 5. P. 109–113. DOI: 10.1145/1125944.1125949
2. Gias A. U., Taher M. A. Performance evaluation of web-based single page applications. Proceedings of the 17th International Conference on Computer and Information Technology (ICCIT). 2014. P. 344–349. DOI: 10.1109/ICCITech.2014.7073104
3. Barnes J., Hut P. A hierarchical $O(N \log N)$ force-calculation algorithm. Nature. 1986. Vol. 324, No 6096. P. 446–449. DOI: 10.1038/324446a0
4. Bastian M., Heymann S., Jacomy M. Gephi: an open source software for exploring and manipulating networks. Proceedings of the Third International AAAI Conference on Weblogs and Social Media. 2009. Vol. 3, No 1. P. 361–362. DOI: 10.1609/icwsm.v3i1.13937
5. Santos A., Silva B. Performance evaluation of data serialization formats in web applications. IEEE Access. 2015. Vol. 3, No 1. P. 1024–1038. DOI: 10.1109/ACCESS.2015.2443122
6. Kleppmann M., Wiggins A., van Hardenberg P., McGilvray K. Local-first software: you own your data, in spite of the cloud. Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!). 2019. P. 154–178. DOI: 10.1145/3359591.3359737
7. Wang X., Zhao Y. Design and implementation of an efficient web-based large file processing system. Journal of Web Engineering. 2018. Vol. 17, No 6. P. 450–465. DOI: 10.13052/jwe1540-9589.1764
8. Rodriguez M. A., Neubauer P. The graph traversal pattern. Graph Data Management: Techniques and Applications. 2011. Vol. 1, No 1. P. 29–46. DOI: 10.4018/978-1-61350-053-8.ch002
9. Jacomy M., Venturini T., Heymann S., Bastian M. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the

- Gephi software. PLoS ONE. 2014. Vol. 9, No 6. e98679. DOI: 10.1371/journal.pone.0098679
10. Al-Fares M., Loukissas A., Vahdat A. A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review. 2008. Vol. 38, No 4. P. 63–74. DOI: 10.1145/1402946.1402967
 11. Jovanovic M., Vrdoljak M. Improving web application performance using Web Workers. Proceedings of the 35th International Convention MIPRO. 2012. P. 1042–1046. DOI: 10.1109/MIPRO.2012.6240788
 12. Marrin T. WebGL: A New Dimension for Web Graphics. IEEE MultiMedia. 2011. Vol. 18, No 3. P. 82–84. DOI: 10.1109/MMUL.2011.54
 13. Heer J., Agrawala M. Software Design Patterns for Information Visualization. IEEE Transactions on Visualization and Computer Graphics. 2006. Vol. 12, No 5. P. 853–867. DOI: 10.1109/TVCG.2006.178
 14. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing. 2010. Vol. 14, No 6. P. 80–83. DOI: 10.1109/MIC.2010.145
 15. Fett M., Kubler D., Smith A. A comprehensive formal security analysis of OAuth 2.0. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016. P. 1204–1215. DOI: 10.1145/2976749.2978385
 16. Petre M. UML in practice. Proceedings of the 35th International Conference on Software Engineering (ICSE). 2013. P. 722–731. DOI: 10.1109/ICSE.2013.6606618
 17. Stonebraker M., Rowe L. A. The design of POSTGRES. ACM SIGMOD Record. 1986. Vol. 15, No 2. P. 340–355. DOI: 10.1145/16856.16888.
 18. Charland A., Leroux B. Mobile application development: web vs. native. Communications of the ACM. 2011. Vol. 54, No 5. P. 49–53. DOI: 10.1145/1941487.1941504

19. Mikkonen T., Taivalaari A. Web applications: The fourth era. IEEE Software. 2018. Vol. 35, No 1. P. 120–125. DOI: 10.1109/MS.2018.111094362
20. Pautasso C., Zimmermann O., Leymann F. Restful web services vs. big web services: making the right architectural decision. Proceedings of the 17th International Conference on World Wide Web. 2008. P. 805–814. DOI: 10.1145/1367497.1367606
21. Garlan D. Software architecture: a travelogue. Proceedings of the Future of Software Engineering. 2014. P. 29–39. DOI: 10.1145/2593882.2593886
22. Otte E., Rousseau R. Social network analysis: a powerful strategy, also for the information sciences. Journal of Information Science. 2002. Vol. 28, No 6. P. 441–453. DOI: 10.1177/016555150202800601
23. Harrison N. B., Avgeriou P., Zdun U. Using patterns to capture architectural decisions. IEEE Software. 2007. Vol. 24, No 4. P. 38–45. DOI: 10.1109/MS.2007.124
24. Herman I., Melancon G., Marshall M. S. Graph visualization and navigation in information visualization: A survey. IEEE Transactions on Visualization and Computer Graphics. 2000. Vol. 6, No 1. P. 24–43. DOI: 10.1109/2945.841119
25. Fielding R. T., Taylor R. N. Principled design of the modern Web architecture. ACM Transactions on Internet Technology. 2002. Vol. 2, No 2. P. 115–150. DOI: 10.1145/514183.514185
26. Biorn-Hansen A., Majchrzak T. A., Gronli T. M. Progressive web apps: The possible web-native unifier for mobile development. Proceedings of the International Conference on Web Information Systems and Technologies. 2017. P. 344–351. DOI: 10.5220/0006353703440351
27. Gibson D., Faith J., Vickers P. A survey of two-dimensional graph layout techniques for information visualisation. Information Visualization. 2013. Vol. 12, No 3. P. 324–357. DOI: 10.1177/1473871612455749

28. Bostock M., Ogievetsky V., Heer J. D3 Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics. 2011. Vol. 17, No 12. P. 2301–2309. DOI: 10.1109/TVCG.2011.185
29. Fruchterman T. M. J., Reingold E. M. Graph drawing by force-directed placement. Software: Practice and Experience. 1991. Vol. 21, No 11. P. 1129–1164. DOI: 10.1002/spe.4380211102
30. Kruchten P. B. The 4+1 view model of architecture. IEEE Software. 1995. Vol. 12, No 6. P. 42–50. DOI: 10.1109/52.469759
31. Franz M., Lopes C. T., Huck G., Dong Y., Sumer O., Bader G. D. Cytoscape.js: a graph theory library for visualisation and analysis. Bioinformatics. 2016. Vol. 32, No 2. P. 309–311. DOI: 10.1093/bioinformatics/btv557
32. Anthes G. HTML5 leads a web revolution. Communications of the ACM. 2012. Vol. 55, No 7. P. 16–17. DOI: 10.1145/2209249.2209256
33. Pienta R., Abello J., Chau D. H., Fisher T. Vigor: Interactive visual exploration of massive graph edges. IEEE Transactions on Visualization and Computer Graphics. 2017. Vol. 23, No 1. P. 401–410. DOI: 10.1109/TVCG.2016.2598444
34. Mwalongo F., Krone M., Karch G., Becher M., Reina G., Ertl T. State-of-the-art report in web-based data visual analytics. Computer Graphics Forum. 2016. Vol. 35, No 3. P. 581–610. DOI: 10.1111/cgf.12929

ДОДАТКИ

		Купріянов Д.І.			ІПЗ.КР.Б-121.26 - ПЗ	Арк.
		Антонюк Д.С				
Змн.	Арк.	№ докум.	Підпис	Дата		59

Реалізація просторової синхронізації та управління режимами візуалізації

```
const syncGraphData3D = useCallback(() => {
  const graph = graphRef.current;
  const newNodes = graph.nodes().map(id => ({id,
    label: graph.getNodeAttribute(id, 'label') || id,
    color: graph.getNodeAttribute(id, 'color'),
    size: graph.getNodeAttribute(id, 'size') || 4,
    cluster: graph.getNodeAttribute(id, 'cluster') || 0,
    x: (graph.getNodeAttribute(id, 'x') || 0)*0.05,
    y: (graph.getNodeAttribute(id, 'y') || 0)*0.05,
    z: (graph.getNodeAttribute(id, 'z') || (Math.random()*1000-500))*0.05
  }));
  const newLinks = graph.edges().map(edge => ({
    source: graph.source(edge),
    target: graph.target(edge),
    id: edge,
    weight: graph.getEdgeAttribute(edge, 'weight') || 0
  }));
  setGraphData3D({ nodes: newNodes, links: newLinks });
  updateStats();
}, []);
const triggerRenderUpdate = useCallback(() => {
  updateStats();
  if (viewMode === "2D") {
    if (sigmaRef.current) sigmaRef.current.refresh();
  } else {
    syncGraphData3D();
  }
}, [viewMode, syncGraphData3D]);
```

Реалізація алгоритмів генерації

Алгоритм Барабаші-Альберт

```
const applyBarabasiAlbert = () => {
  const graph = graphRef.current;
  const currentNodes = graph.nodes();
  const totalEdges = graph.size * 2 || 1;
  for (let i = 0; i < 5; i++) {
    const newId = `Hub_${Date.now()}_${i}`;
    graph.addNode(newId, {
      x: Math.random() * 500 - 250, y: Math.random() * 500 - 250, z: Math.random() * 500
      - 250,
      size: 12, color: "#EBCB8B", label: `Hub_${i+1}`, labelColor: "#ECEFF4"
    });
    currentNodes.forEach(targetNode => {
      const probab = (graph.degree(targetNode) / totalEdges) * 30;
      if (Math.random() < probab) {
        const amount = Math.floor(Math.random() * 5000);
        graph.addEdge(newId, targetNode, { color: "#81A1C1", size: 1.5, weight: amount
        });
      }
    });
  }
  triggerRenderUpdate();
};
```

Метод просторової локалізації

```
// App.js
const applySpatialLocalization = () => { const graph = graphRef.current;
  if (graph.order === 0) return alert("Граф порожній!");
  setIsLoading(true);
  if (workerRef.current) {
    workerRef.current.postMessage({
      type: 'Localization',
      serializedGraph: graph.export(),
      viewMode: viewMode
    });
  }
};
```

```

// worker.js
else if (type === 'Localization') {
    const graph = new Graph({ multi: true });
    graph.import(serializedGraph);
    const nodes = graph.nodes();
    let newEdges = [];
    const MAX_EDGES_PER_NODE = 10;
    const currentEdgesCount = new Map();
    nodes.forEach(n => currentEdgesCount.set(n, 0));
    for (let i = 0; i < nodes.length; i++) {
        for (let j = i + 1; j < nodes.length; j++) {
            const n1 = nodes[i];
            const n2 = nodes[j];
            if (currentEdgesCount.get(n1) >= MAX_EDGES_PER_NODE || currentEdgesCount.get(n2) >=
MAX_EDGES_PER_NODE) continue;
            let dist = 1000;
            const attr1 = graph.getNodeAttributes(n1);
            const attr2 = graph.getNodeAttributes(n2);
            if (viewMode === "2D") {
                const dx = attr2.x - attr1.x;
                const dy = attr2.y - attr1.y;
                dist = Math.sqrt(dx * dx + dy * dy);
            } else {
                const dx = (attr2.x || 0) - (attr1.x || 0);
                const dy = (attr2.y || 0) - (attr1.y || 0);
                const dz = (attr2.z || 0) - (attr1.z || 0);
                dist = Math.sqrt(dx*dx + dy*dy + dz*dz);
            }
            if (dist < 200 && !graph.hasEdge(n1, n2)) {
                newEdges.push({ source: n1, target: n2, weight: Math.floor(Math.random() * 1000)
});
                currentEdgesCount.set(n1, currentEdgesCount.get(n1) + 1);
                currentEdgesCount.set(n2, currentEdgesCount.get(n2) + 1);
            }
        }
    }
    postMessage({ type: 'Localization_Done', newEdges });
}

```

Реалізація підсистеми асинхронного парсингу та імпорту файлів

```

const handleFileUpload = (event) => {
const file = event.target.files[0];
if (!file) return;
if (file.name.endsWith('.json')) {
const reader = new FileReader();
reader.onload = (e) => {
try {
const data = JSON.parse(e.target.result);
setJsonInput(JSON.stringify(data, null, 2));
handleLoadJson(data);
} catch (error) { alert("Помилка читання JSON файлу!"); }
};
reader.readAsText(file);
} else if (file.name.endsWith('.csv')) {
Papa.parse(file, {
header: true, skipEmptyLines: true,
complete: (results) => {
const graph = graphRef.current;
graph.clear();
results.data.forEach(row => {
const source = row.Source || row.source || row.From || row.from;
const target = row.Target || row.target || row.To || row.to;
const amount = row.Amount || row.amount || row.Weight || row.weight ||
Math.floor(Math.random() * 5000);
if (source && target) {
if (!graph.hasNode(source)) graph.addNode(source, { x: Math.random()*500-
250, y: Math.random()*500-250, z: Math.random()*500-250, size: 5, color: "#88C0D0",
label: source });
if (!graph.hasNode(target)) graph.addNode(target, { x: Math.random()*500-
250, y: Math.random()*500-250, z: Math.random()*500-250, size: 5, color: "#d0bf88ff",
label: target });
graph.addEdge(source, target, { color: "#4C566A", size: 1, weight:
Number(amount) });
}
});
clearSearch();
if (viewMode === "2D") {
workerRef.current.postMessage({
type: 'RunPhysics',
serializedGraph: graph.export(),
iterations: 25,
settings: { gravity: gravity || 0.5, scalingRatio: scalingRatio || 1 }
});
} else {
triggerRenderUpdate();
}
}
});
} else { alert("Підтримуються лише файли .json та .csv"); }
}

```

Реалізація модуля глобального пошуку та фокусування камери

```
const focusOnNode = (targetId) => {
const graph = graphRef.current;
if (!graph.hasNode(targetId)) return;
  setSearchQuery(targetId);
  setSearchFocusNode(targetId);
  const neighbors = graph.neighbors(targetId);
  setSearchNeighbors(new Set(neighbors));
  const transactions = [];
  graph.edges(targetId).forEach(e => {
    const src = graph.source(e);
    const tgt = graph.target(e);
    const amount = graph.getEdgeAttribute(e, 'weight') || 0;
    transactions.push({ src, tgt, amount });
  });
  setNodeDetails({ id: targetId, transactions });
  if (viewMode === "3D" && fgRef.current) {
    const node3D = graphData3D.nodes.find(n => String(n.id) === String(targetId));
    setHoveredNode3D(node3D || { id: targetId });
    hoveredNeighbors3DRef.current.clear();
    neighbors.forEach(n => hoveredNeighbors3DRef.current.add(n));
  } else {
    if (sigmaRef.current) sigmaRef.current.refresh();
  }
};
```

Алгоритм оптимізації модулярності Лувена (через Web Worker)

```

const runLouvain = () => {
const graph = graphRef.current;
  if (graph.order === 0) return alert("Граф порожній!");
  setIsLoading(true);
  if (workerRef.current) {
    workerRef.current.postMessage({
      type: 'Louvain',
      serializedGraph: graph.export()
    });
  }
};
// worker.js
let communities = {};
try { communities = louvain(graph); } catch(e) {}
const clusterColors = ["#BF616A", "#D08770", "#EBCB8B", "#A3BE8C", "#8FBCBB", "#88C0D0",
"#81A1C1", "#5E81AC", "#B48EAD"];
const results = {};
graph.forEachNode((node) => {
  const clusterId = communities[node] || 0;
  results[node] = {
    cluster: clusterId,
    color: clusterColors[clusterId % clusterColors.length]
  };
});
postMessage({ type: 'Louvain_Done', results });

```

Метод розрахунку центральності за ступенем

```

const runCentrality = () => { const graph = graphRef.current;
  if (graph.order === 0) return alert("Граф порожній!");
  let centralities = {};
  try { centralities = degreeCentrality(graph); } catch(e) {}
  let maxScore = Math.max(...Object.values(centralities), 0.0001);
  graph.forEachNode((node) => {const baseSize = 3;
    const centralityScore = (centralities[node] / maxScore) * 15;
    graph.setNodeAttribute(node, "size", baseSize + centralityScore);
  });
  triggerRenderUpdate();
};

```

Програмний код розробленого вебзастосунку знаходиться у репозиторії за посиланням: <https://git.ztu.edu.ua/ipz/2022-2026/ipz-22-3/kupriyanov-danylo/web-app-for-interactive-large-social-graph-visualization.git>