

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інтерактивна кросплатформена система
візуального моделювання інформації»**

Виконав студент групи ІПЗ-22-2
АНДРІЮК Денис Русланович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

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: **АНДРІЮК Денис Русланович**

Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**

Тема роботи: **«Інтерактивна кросплатформена система візуального моделювання інформації»**,

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

Вихідні дані для роботи: Об'єктом дослідження є процеси створення, редагування, зберігання та візуалізації структурованої інформації у вигляді графів, дерев, часових ліній та інших інформаційних моделей. Предметом дослідження є програмна архітектура, моделі даних, клієнтська та серверна логіка кросплатформеної системи візуального моделювання інформації.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Левківський В.Л.	27.02.2026р.	27.02.2026р.
2	Левківський В.Л.	27.03.2026р	27.03.2026р
3	Левківський В.Л.	27.04.2026р	27.04.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра на тему «Інтерактивна кросплатформена система для візуального моделювання інформації» складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Текстова частина викладена на 108 сторінках друкованого тексту. Пояснювальна записка має 23 сторінки додатків. Список використаних джерел містить 33 найменувань і займає 4 сторінки. В роботі наведено 39 рисунків та 37 таблиць. Загальний обсяг роботи 131 сторінок.

Метою роботи є розробка інтерактивної системи для створення, редагування та візуалізації структурованої інформації у вигляді графів, дерев, часових ліній та інших моделей. Система орієнтована на забезпечення зручного інтерфейсу для роботи зі складними інформаційними структурами та їх наочним представленням.

В результаті розробки було створено програмний продукт, який реалізує функціонал візуального моделювання даних та підтримує інтерактивну взаємодію користувача з моделями. Система побудована на основі сучасних вебтехнологій із використанням клієнт-серверної архітектури та може бути використана як основа для подальшого розвитку та масштабування.

КЛЮЧОВІ СЛОВА: ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ, ГРАФИ, ДЕРЕВА, ВЕБДОДАТОК, REACT, NESTJS, POSTGRESQL, API, КРОСПЛАТФОРМЕНІСТЬ.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інтерактивна кросплатформена система візуального моделювання інформації Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.	Андріюк Д. Р.						1	131
Керівник	Левківський В.Л.					ДУ «Житомирська політехніка», група ІПЗ-22-2		
Рецензент	Локтікова Т.М.							
Н. контр.								
Зав. каф.	Вакалюк Т.А.							

ABSTRACT

The bachelor's thesis titled "Interactive Cross-Platform System for Visual Information Modeling" consists of an introduction, three chapters, conclusions, references, and appendices.

The main text comprises 108 pages of printed text. The explanatory note includes 23 pages of appendices. The list of references contains 33 entries and spans 4 pages. The work includes 39 figures and 37 tables. The total length of the work is 131 pages.

The purpose of the work is to develop an interactive system for creating, editing, and visualizing structured information in the form of graphs, trees, timelines, and other models. The system is aimed at providing a user-friendly interface for working with complex data structures and their visual representation.

As a result of the development, a software product was created that implements visual modeling functionality and supports interactive user interaction with models. The system is based on modern web technologies using a client-server architecture and can serve as a foundation for further development and scalability.

KEYWORDS: VISUAL MODELING, GRAPHS, TREES, WEB APPLICATION, REACT, NESTJS, POSTGRESQL, API, CROSS-PLATFORM.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА	
ЗАДАЧІ	9
1.1. Аналіз предметної області візуального моделювання інформації.....	9
1.2. Аналіз існуючих програмних рішень та аналогів	12
1.3. Визначення вимог до системи.....	17
1.4. Вибір інструментів розробки	21
Висновки до першого розділу.....	25
РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ	26
2.1. Визначення варіантів використання системи.....	26
2.2. Загальна архітектура системи	31
2.3. Проектування програмної архітектури.....	38
2.4. Проектування структури даних	44
2.5. Програмна реалізація системи	50
Висновки до другого розділу.	65
РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ ..	67
3.1. Структура та основні елементи інтерфейсу системи.....	67
3.2. Порядок роботи користувача із системою.	72
3.3. Налаштування, встановлення та розгортання системи.....	80
3.4. Керування доступом та ролями користувачів.	86
3.5. Тестування роботи системи.	94
Висновки до третього розділу.....	101
ВИСНОВКИ	102
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	105
ДОДАТКИ.....	109

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface / Програмний інтерфейс прикладного програмування.

CI/CD – Continuous Integration / Continuous Deployment / Безперервна інтеграція та доставка.

CRUD – Create, Read, Update, Delete / Основні операції з даними.

ERD – Entity-Relationship Diagram / Модель «сутність – зв'язок».

HTTP – HyperText Transfer Protocol / Гіпертекстовий протокол передачі.

JSON – JavaScript Object Notation / Запис об'єктів JavaScript.

JWT – JSON Web Token / Веб-токен у форматі JSON.

MVC – Model-View-Controller / Модель–представлення–контролер.

ORM – Object-Relational Mapping / Об'єктно-реляційне відображення.

RBAC - Role-Based Access Control / Контроль доступу на основі ролей.

REST - Representational State Transfer / Технологія архітектури обміну даними

SaaS - Software as a Service / Програмне забезпечення як послуга

UI – User Interface / Користувацький інтерфейс.

UML – Unified Modeling Language.

UX – User Experience / Користувацький досвід.

WS – WebSocket.

БД – База даних.

СКБД – Система керування базами даних.

ІТ – Інформаційні технології.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій значно зростає обсяг даних, що потребують обробки, структуризації та подальшого аналізу. Відповідно, актуальним завданням є створення ефективних інструментів для представлення інформації у наочній та зрозумілій формі. Одним із найбільш перспективних підходів до вирішення даної проблеми є використання методів візуального моделювання, які дозволяють відображати складні взаємозв'язки між об'єктами у вигляді графічних структур.

Візуальне моделювання інформації знаходить широке застосування у різних сферах діяльності, зокрема в управлінні бізнес-процесами, побудові організаційних структур, аналізі даних, генеалогічних дослідженнях, розробці програмного забезпечення та системах підтримки прийняття рішень. Використання графів, дерев, часових ліній та інших структур дозволяє не лише спростити сприйняття інформації, але й підвищити ефективність її аналізу.

Незважаючи на наявність значної кількості програмних рішень у сфері візуалізації даних, більшість з них мають обмежену функціональність, не забезпечують достатнього рівня інтерактивності або не підтримують роботу з різними типами інформаційних структур у межах єдиної системи. Крім того, існуючі інструменти часто орієнтовані на вузькоспеціалізовані задачі та не враховують потребу у кросплатформеності та масштабованості.

У зв'язку з цим виникає необхідність розробки універсальної системи, яка поєднуватиме можливості створення, редагування та візуалізації різних типів структурованої інформації, забезпечуючи при цьому зручний користувацький інтерфейс та підтримку сучасних технологій взаємодії.

Актуальність теми полягає у необхідності створення інтерактивних програмних засобів, здатних ефективно працювати з великими обсягами структурованих даних та забезпечувати їх наочне представлення у режимі реального часу. Розробка такої системи дозволяє підвищити продуктивність

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

роботи користувачів, покращити якість аналізу інформації та створити основу для подальшого розвитку інтелектуальних систем обробки даних.

Метою даної роботи є розробка інтерактивної кросплатформеної системи для візуального моделювання інформації, яка забезпечує створення, редагування та відображення структурованих даних у вигляді графів, дерев, часових ліній та інших моделей.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести аналіз предметної області візуального моделювання інформації;
- дослідити існуючі програмні рішення та визначити їх недоліки;
- сформулювати функціональні та нефункціональні вимоги до системи;
- розробити архітектуру програмного забезпечення;
- спроектувати структуру бази даних;
- реалізувати серверну та клієнтську частини системи;
- забезпечити інтерактивну взаємодію користувача з моделями;
- провести тестування та оцінку працездатності системи.

Об'єктом дослідження є процеси візуального представлення та моделювання інформації.

Предметом дослідження є методи, засоби та програмні технології реалізації інтерактивних систем візуального моделювання.

Практичне значення отриманих результатів полягає у створенні програмного продукту, який може бути використаний для побудови та аналізу різних типів інформаційних структур у різних предметних областях. Запропоноване рішення може слугувати основою для подальшого розвитку систем візуалізації даних, а також інтеграції з іншими інформаційними системами.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз предметної області візуального моделювання інформації

У сучасному інформаційному суспільстві обсяг даних, що генеруються, обробляються та зберігаються, постійно зростає. Це обумовлює необхідність використання ефективних підходів до їх представлення, аналізу та інтерпретації. Одним із ключових інструментів, що дозволяє вирішувати зазначені задачі, є візуальне моделювання інформації. У працях з інформаційної візуалізації підкреслюється, що графічне представлення даних дозволяє користувачу швидше виявляти структури, закономірності, зв'язки та відхилення, які складніше сприймати у табличній або текстовій формі [1–3].

Візуальне моделювання інформації – це процес представлення даних та зв'язків між ними у графічній формі з метою полегшення їх сприйняття, аналізу та обробки. Такий підхід дозволяє перетворити складні абстрактні структури у зрозумілі візуальні образи, що підвищує ефективність роботи з інформацією [1, 2].

Особливість візуального моделювання полягає в тому, що воно поєднує два взаємопов'язані аспекти: інформаційний та графічний. Інформаційний аспект визначає, які саме об'єкти предметної області відображаються в моделі, які властивості вони мають і які відношення існують між ними. Графічний аспект відповідає за спосіб подання цих об'єктів у візуальному середовищі, зокрема за розташування елементів, форму вузлів, типи зв'язків, підписи, масштабування та загальну читабельність структури. Тому ефективна система візуального моделювання повинна забезпечувати не лише відображення даних, а й збереження їх змістової структури.

Графи є універсальною структурою, що складається з множини вершин і зв'язків між ними. Вони широко застосовуються для моделювання соціальних мереж, транспортних систем, інформаційних потоків та залежностей між об'єктами. Для графових структур важливими є не лише самі зв'язки, а й спосіб

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

їх візуального розміщення, оскільки якість компоновання впливає на читабельність моделі [4, 5].

Дерева є окремим випадком графів, що мають ієрархічну структуру. Вони використовуються для представлення підпорядкованих систем, таких як організаційні структури, файлові системи, генеалогічні дерева та класифікації. Для таких моделей важливим є чітке відображення рівнів ієрархії та зв'язків між батьківськими й дочірніми елементами [4, 6].

Ієрархічні моделі є важливими для тих предметних областей, де інформація має чітко виражену структуру підпорядкування або походження. Наприклад, в організаційних моделях така структура дозволяє показати зв'язки між підрозділами, посадами та відповідальними особами, а в генеалогічних моделях – відобразити родинні відносини між поколіннями. Незважаючи на спільну ієрархічну природу, ці типи моделей мають різну семантику: для організаційної структури основним є підпорядкування, тоді як для генеалогії важливими є родинні зв'язки, партнерські відносини та спадковість між поколіннями. Це свідчить про необхідність підтримки різних правил інтерпретації зв'язків у межах єдиного редактора.

Часові лінії застосовуються для відображення подій у часовому контексті. Вони дозволяють аналізувати послідовність подій, визначати їх тривалість і взаємозв'язки в часі. Такий підхід широко використовується в історичних дослідженнях, управлінні проєктами та аналітиці [1, 3].

Окрему складність становить робота з великими моделями. Зі збільшенням кількості вузлів і зв'язків зростає ризик візуального перевантаження, накладання елементів, ускладнення навігації та зниження швидкодії інтерфейсу. Для збереження зрозумілості моделі необхідно застосовувати механізми масштабування, панорамування, автоматичного або напівавтоматичного компоновання, а також оптимізованого відображення графічних елементів. У графових моделях особливо важливими є мінімізація перетинів зв'язків, збереження логічної відстані між пов'язаними елементами та стабільність розташування вузлів під час редагування [4, 5, 7].

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Візуальне моделювання інформації має важливе значення не лише як спосіб подання даних, а і як інструмент їх структурного осмислення. На відміну від простого графічного зображення, модель повинна відображати сутності предметної області, їх властивості та зв'язки між ними. Завдяки цьому користувач отримує можливість працювати не з окремими ізольованими елементами, а з цілісною інформаційною структурою, у якій кожен вузол і кожен зв'язок мають визначене змістове навантаження. Такий підхід є особливо важливим для складних предметних областей, де інформація має багаторівневий характер і не може бути ефективно представлена лише у вигляді текстового опису або таблиці.

Однією з переваг візуального моделювання є можливість зменшення когнітивного навантаження на користувача. Коли дані представлені у вигляді взаємопов'язаних об'єктів, користувач швидше розуміє загальну структуру інформації, бачить центральні елементи моделі, виявляє залежності та може оцінити складність системи в цілому. Це має значення для освітніх, аналітичних, управлінських і проєктних задач, де важливо не лише зберегти інформацію, а й забезпечити її зрозуміле сприйняття. З погляду людино-комп'ютерної взаємодії, візуальна модель повинна бути не перевантаженою, логічно організованою та доступною для інтерактивної роботи користувача [8, 9].

Ще однією важливою характеристикою сучасних програмних систем є кросплатформеність. Відповідно до моделі якості програмного забезпечення ISO/IEC 25010, для прикладних систем важливими є зручність використання, сумісність, надійність, переносимість і підтримуваність [10].

Незважаючи на значну кількість існуючих інструментів для візуалізації даних, більшість із них мають ряд обмежень. Зокрема, вони часто підтримують лише один тип структур, мають обмежені можливості інтерактивної взаємодії або не забезпечують достатньої гнучкості у налаштуванні моделей. Крім того, багато рішень не орієнтовані на інтеграцію з іншими системами або масштабування.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

Важливим аспектом предметної області є також збереження еволюції інформаційної моделі. У практичних сценаріях користувач не лише створює модель один раз, а постійно її уточнює: додає нові елементи, змінює властивості, видаляє застарілі зв'язки, імпортує або експортує дані. Тому система повинна підтримувати не тільки поточний стан моделі, а й механізми збереження, відновлення та контролю змін. Це дозволяє підвищити надійність роботи з даними та зменшити ризик втрати важливої інформації під час редагування.

Отже, предметна область візуального моделювання інформації охоплює не лише побудову графічного представлення даних, а й формалізацію структури, підтримку різних типів моделей, інтерактивну взаємодію, збереження даних, контроль доступу та забезпечення прийнятної швидкодії. Саме поєднання цих характеристик визначає потребу у створенні універсальної інтерактивної системи, яка дозволяє користувачам працювати зі складними інформаційними структурами в єдиному програмному середовищі.

1.2. Аналіз існуючих програмних рішень та аналогів

У сфері візуального моделювання інформації існують як універсальні інструменти побудови діаграм, так і спеціалізовані системи, орієнтовані на конкретні типи структур, такі як генеалогічні дерева або часові лінії. У межах даного дослідження доцільно розглядати саме такі спеціалізовані веб-системи як найближчі аналоги розроблюваного програмного продукту, такі як: Family Tree Builder, Time.Graphics, Miro (як універсальний інструмент візуалізації).

Family Tree Builder є спеціалізованою системою для побудови генеалогічних дерев [11]. Основний функціонал включає створення та редагування сімейних зв'язків, додавання інформації про осіб, візуалізацію ієрархічної структури родинних відносин. Система забезпечує зручне представлення даних, однак обмежується лише одним типом структури – деревом, що значно звужує сферу її застосування.

Time.Graphics є веб-сервісом для створення часових ліній [12]. Користувач має можливість додавати події, визначати часові інтервали та відображати їх на шкалі часу. Система є ефективною для аналізу хронологічних даних, однак не

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

підтримує складні графові структури або взаємозв'язки між об'єктами поза часовим контекстом.

Універсальні інструменти, навпаки, надають користувачу більшу свободу у створенні схем і візуальних представлень. Прикладом такого підходу є Miro, який дозволяє створювати різні типи діаграм і підтримує спільну роботу користувачів у реальному часі [13]. Однак універсальність таких платформ часто досягається за рахунок меншої формалізації даних. Об'єкти на дошці можуть бути зручними для візуального сприйняття, але не завжди мають чітко визначені типи, атрибути, зв'язки, правила доступу або історію змін. У результаті такі системи більше підходять для вільного проектування схем, ніж для довготривалого збереження та обробки структурованих інформаційних моделей.

Окрему категорію становлять бібліотеки для побудови візуалізацій, зокрема D3. Такі інструменти мають високу гнучкість і дозволяють розробнику створювати складні графічні представлення даних [14]. Водночас вони не є готовими прикладними системами для кінцевого користувача. Для отримання повноцінного програмного продукту на основі такої бібліотеки необхідно додатково реалізовувати збереження моделей, авторизацію, ролі доступу, імпорт, експорт, адміністрування, інтерфейс редагування та механізми спільної роботи. Тому D3 доцільно розглядати не як прямий аналог Graphorі, а як технологічну альтернативу для реалізації окремих компонентів візуалізації.

До універсальних інструментів побудови діаграм також належить Draw.io, відомий також як diagrams.net. Це веборієнтований і десктопний інструмент для створення блок-схем, UML-діаграм, ER-діаграм, мережевих схем, організаційних структур та інших типів візуальних представлень. Його перевагою є доступність, можливість локального використання та широкий набір шаблонів для різних типів діаграм. Водночас Draw.io переважно орієнтований на ручне створення схем і не забезпечує повноцінної роботи з інформаційною моделлю як із набором структурованих сутностей, що мають власні атрибути, правила доступу, версії та спеціалізовану логіку редагування для різних типів моделей.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Ще одним поширеним рішенням є Lucidchart – хмарний інструмент для побудови діаграм і візуальної співпраці. Він дає змогу створювати блок-схеми, організаційні структури, UML-діаграми, ER-діаграми та інші типи схем, а також підтримує спільну роботу користувачів і інтеграції з іншими сервісами. Перевагою Lucidchart є зручність командної роботи, готові шаблони та орієнтація на бізнес-процеси й системне проектування. Проте, як і більшість універсальних diagramming-платформ, Lucidchart насамперед зосереджений на створенні візуальних схем, тоді як розроблювана система Graphori орієнтована на роботу зі структурованими інформаційними моделями, де кожен елемент має тип, властивості, зв'язки, правила доступу та може використовуватися в різних режимах візуального подання.

Для порівняння функціональних можливостей розглянутих систем та розроблюваного програмного продукту наведено таблицю 1.1.

Таблиця 1.1.

Порівняння функціональних можливостей систем

Функціональність	Family Tree Builder	Time.Graphics	Miro	Draw.io	Lucidchart	Розроблювана система
Підтримка графів	—	—	частково	частково	частково	+
Підтримка дерев	+	—	частково	частково	частково	+
Таймлайни	—	+	частково	частково	частково	+
Робота з різними типами структур	—	—	частково	частково	частково	+
Генеалогічні структури	+	—	частково	частково	частково	+
Організаційні ієрархії	частково	—	+	+	+	+
Інтерактивне редагування	+	+	+	+	+	+
Робота в реальному часі	—	+	+	частково	+	+
Кросплатформеність	частково	лише web	+	+	+	+
API / інтеграції	—	—	+	частково	+	+

Формалізоване збереження структурованих моделей	частково	частково	—	—	частково	+
Ролі та права доступу	—	—	частково	частково	частково	+
Версіонування моделей	—	—	частково	частково	частково	+

Аналіз таблиці 1.1 показує, що Draw.io та Lucidchart мають ширші можливості побудови діаграм, ніж вузькоспеціалізовані інструменти, однак їх основне призначення полягає у створенні візуальних схем, а не в роботі з інформаційною моделлю як із формалізованою структурою даних. Miro, Draw.io та Lucidchart можуть використовуватися для побудови окремих графічних представлень, проте вони не забезпечують повного поєднання різних типів моделей, спеціалізованої логіки редагування, версіонування, керування доступом і предметно-орієнтованого збереження структурованих даних у межах єдиного середовища.

Під час аналізу існуючих рішень важливо враховувати не лише наявність окремих функцій, а й те, наскільки ці функції утворюють цілісне середовище для роботи зі структурованими інформаційними моделями. Спеціалізовані системи, такі як Family Tree Builder або Time.Graphics, добре розв'язують окремі прикладні задачі, однак їх функціональність обмежена конкретним типом структури. Це є перевагою для вузьких сценаріїв, оскільки інтерфейс і логіка системи максимально адаптовані до однієї предметної області. Проте така спеціалізація ускладнює використання цих інструментів для моделювання інших типів даних, наприклад залежностей між об'єктами, організаційних структур або довільних графів.

Окремо доцільно розглянути переваги та недоліки досліджених систем представлених в таблиці 1.2.

Переваги та недоліки існуючих рішень

Система	Переваги	Недоліки
Family Tree Builder	Простота використання, зручна робота з генеалогією	Обмеження лише деревами, відсутність універсальності
Time.Graphics	Зручна візуалізація часових даних	Відсутність графових структур та зв'язків
Miro	Гнучкість, колаборація, сучасний інтерфейс	Відсутність формалізованого моделювання даних
Draw.io	Доступність, підтримка великої кількості типів діаграм, можливість web- і desktop-використання, зручність для створення схем і технічної документації	Орієнтація переважно на ручне створення діаграм, відсутність спеціалізованої моделі даних, обмежена підтримка ролей, версіонування та предметної логіки редагування
Lucidchart	Зручна побудова діаграм, підтримка командної роботи, наявність шаблонів, інтеграції з іншими сервісами	Основний акцент на створенні схем, а не на моделюванні структурованих даних; обмежена підтримка спеціалізованих типів моделей, версіонування та гнучкого керування доступом

Порівняння переваг і недоліків існуючих рішень показує, що кожна система ефективно розв'язує окремий клас задач, однак не забезпечує повного набору можливостей, необхідних для універсального візуального моделювання інформації. Family Tree Builder і Time.Graphics є зручними для вузьких сценаріїв, Miro, Draw.io та Lucidchart забезпечують ширші можливості побудови схем, проте не орієнтовані на комплексну роботу зі структурованими моделями, що мають типи, атрибути, зв'язки, правила доступу, історію змін і різні режими візуального подання. Це підтверджує доцільність розробки системи Graphori як окремого програмного продукту, що поєднує інтерактивне редагування, формалізоване збереження даних і підтримку різних типів інформаційних структур.

Таким чином, аналіз аналогів підтверджує доцільність розробки системи Graphori як проміжного рішення між вузькоспеціалізованими інструментами та універсальними візуальними дошками. На відміну від спеціалізованих систем, Graphori має підтримувати різні типи моделей у межах єдиного середовища. На відміну від універсальних дошок, система повинна працювати не лише з візуальними об'єктами, а зі структурованими даними, які можна зберігати, редагувати, експортувати, імпортувати, версіонувати та захищати за допомогою механізмів доступу. Саме ці особливості визначають функціональну відмінність розроблюваного програмного продукту від розглянутих аналогів.

Як назву розроблювального проекту обрано «Graphori» та в подальшому будуть звернення до назви замість повної назви кваліфікаційної роботи.

1.3. Визначення вимог до системи

Проведений аналіз предметної області візуального моделювання інформації, а також дослідження існуючих програмних рішень (розділи 1.1, 1.2) дозволяють чітко визначити постановку задачі та цілі розробки.

Основною проблемою, що була виявлена в ході дослідження, є відсутність універсальних програмних засобів, які б дозволяли працювати з різними типами структурованої інформації у межах єдиного середовища. Більшість існуючих рішень орієнтовані на вузьку спеціалізацію, наприклад, побудову генеалогічних дерев або часових ліній, що значно обмежує їх застосування. Універсальні інструменти, у свою чергу, не забезпечують достатнього рівня формалізації, інтерактивності та роботи з даними як структурованими моделями.

Таким чином, постає задача розробки інтерактивної кросплатформеної системи, яка забезпечує можливість створення, редагування та візуалізації різних типів інформаційних моделей, включаючи графи, дерева, часові лінії та мережі залежностей, із забезпеченням єдиного підходу до роботи з ними.

Сутність поставленої задачі полягає у створенні програмного продукту, який дозволяє користувачам не лише відображати інформацію, але й активно взаємодіяти з нею, змінювати структуру моделей, аналізувати зв'язки між об'єктами та отримувати миттєвий результат змін.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

Метою роботи є розробка інтерактивної кросплатформеної системи для візуального моделювання інформації, яка забезпечує універсальність, інтерактивність, масштабованість та зручність використання. Для досягнення поставленої мети необхідно вирішити задачі, представлені у вступі.

Очікуваним результатом виконання роботи є створення програмного продукту, який відповідає сучасним вимогам до веб-застосунків, забезпечує ефективну роботу з візуальними моделями та може бути використаний у різних предметних областях.

Практична значущість роботи полягає у можливості використання розробленої системи як універсального інструменту для моделювання інформації, а також як основи для подальшого розвитку та інтеграції з іншими інформаційними системами.

Сформовані вимоги до системи безпосередньо впливають із визначеної проблеми та обмежень розглянутих аналогів. Насамперед система повинна забезпечувати базові операції над елементами інформаційної моделі: створення, редагування та видалення вузлів і зв'язків між ними. Вузли відображають сутності предметної області, тоді як зв'язки визначають відношення між цими сутностями. Користувач повинен мати можливість змінювати властивості елементів, переміщувати їх у робочій області, встановлювати зв'язки між вузлами та редагувати структуру моделі у візуальному режимі [7].

Ключовою функціональною вимогою є підтримка різних типів структур у межах єдиного середовища. Система повинна дозволяти працювати з графами, деревами, часовими лініями, генеалогічними структурами, організаційними ієрархіями та мережами взаємозв'язків. Така вимога обумовлена тим, що графові та ієрархічні структури є базовими формами представлення зв'язків між об'єктами, а їх ефективне відображення потребує спеціалізованих підходів до компонування та взаємодії з елементами [4-6].

Важливою вимогою є інтерактивність роботи з моделлю. Користувач повинен мати можливість масштабувати робочу область, виконувати панорамування, обирати елементи, відкривати панель властивостей, редагувати

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

дані та бачити результат змін без повного перезавантаження сторінки. Такий підхід відповідає принципам ефективної людино-комп'ютерної взаємодії, згідно з якими система повинна забезпечувати зрозумілий зворотний зв'язок, передбачуваність дій і зменшення зайвого навантаження на користувача [8, 9].

Окремою функціональною вимогою є збереження, завантаження, імпорт, експорт і версіонування моделей. Оскільки інформаційні моделі можуть містити значну кількість взаємопов'язаних елементів, система повинна забезпечувати надійне довготривале зберігання даних, можливість повернення до попередніх станів і перенесення моделей між середовищами. З погляду якості програмного забезпечення такі вимоги пов'язані з надійністю, підтримуваністю та зручністю використання системи [10].

Система повинна підтримувати керування доступом до моделей. Для цього необхідно передбачити приватні, публічні та спільні моделі, а також різні рівні дозволів для користувачів. Користувач має мати можливість керувати доступністю власних моделей, надавати іншим користувачам право перегляду або редагування, а адміністратор — виконувати модерацію публічного контенту. Такі вимоги пов'язані із захистом користувацьких даних, автентифікацією, авторизацією та запобіганням несанкціонованому доступу [15, 16].

Для забезпечення обміну даними між клієнтською та серверною частинами система повинна мати програмний інтерфейс взаємодії. REST-підхід є доцільним для реалізації стандартних операцій створення, читання, оновлення та видалення даних, оскільки передбачає чітке розділення клієнта і сервера та використання ресурсно-орієнтованої моделі взаємодії [17, 18].

З архітектурного погляду система повинна бути модульною та підтримуваною. Це означає, що окремі частини програмного продукту мають відповідати за різні аспекти роботи: автентифікацію, користувачів, моделі, вузли, зв'язки, атрибути, спільний доступ, адміністрування та взаємодію в реальному часі. Такий підхід відповідає принципам чистої архітектури та розділення відповідальностей, що спрощує супровід і розвиток програмного продукту [19, 20].

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Для узагальнення ключових вимог до системи наведено таблицю 1.3.

Таблиця 1.3.

Узагальнення вимог до системи

Категорія	Вимога	Опис
Функціональна	Робота з вузлами та зв'язками	Створення, редагування та видалення елементів моделі
Функціональна	Підтримка різних структур	Графи, дерева, таймлайни, генеалогії, ієрархії та мережі
Функціональна	Інтерактивне редагування	Масштабування, панорамування, вибір і редагування елементів
Функціональна	Збереження даних	Збереження, завантаження, імпорт, експорт і версіювання моделей
Функціональна	Керування доступом	Приватні, публічні та спільні моделі, ролі й дозволи
Функціональна	Взаємодія в реальному часі	Оновлення стану моделі без перезавантаження сторінки
Нефункціональна	Продуктивність	Швидка реакція інтерфейсу при роботі зі складними моделями
Нефункціональна	Масштабованість	Можливість розвитку системи та збільшення навантаження
Нефункціональна	Кросплатформеність	Робота у вебсередовищі та підтримка десктопного застосунку
Нефункціональна	Безпека	Захист користувацьких даних, автентифікація та авторизація
Архітектурна	Клієнт-серверна структура	Розділення інтерфейсу користувача та серверної логіки
Архітектурна	Модульність	Незалежний розвиток окремих частин системи
Архітектурна	API-взаємодія	Наявність програмного інтерфейсу для обміну даними
Архітектурна	Контейнеризація	Спрощення розгортання та перенесення системи між середовищами

Таким чином, сформовані вимоги визначають основу для подальшого вибору архітектури, технологічного стеку та способів реалізації програмного

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

продукту. Вони охоплюють як функціональні можливості системи, так і якісні характеристики, необхідні для створення повноцінної інтерактивної платформи візуального моделювання інформації.

1.4. Вибір інструментів розробки

Вибір технологічного стеку для розробки інтерактивної системи візуального моделювання інформації є важливим етапом, оскільки він безпосередньо впливає на продуктивність, масштабованість, зручність розробки, підтримуваність і можливість подальшого розвитку програмного продукту. Обрані технології повинні відповідати вимогам, сформованим у підрозділі 1.3, а також забезпечувати ефективну реалізацію усіх передбачених частин системи.

Для реалізації клієнтської частини було обрано бібліотеку React із використанням TypeScript. React забезпечує компонентний підхід до побудови інтерфейсу, що дозволяє розділяти систему на незалежні та повторно використововувані частини [21, 22].

Це є особливо важливим для системи, яка містить складні інтерактивні компоненти: редактор моделей, панель властивостей, навігаційні елементи, адміністративну панель і сторінку профілю користувача. Використання TypeScript підвищує надійність коду завдяки статичній типізації та спрощує підтримку великих проєктів [22].

Додатковою перевагою React у межах розроблюваної системи є його придатність для побудови динамічних інтерфейсів, у яких стан сторінки часто змінюється внаслідок дій користувача. Для візуального редактора моделей це має особливе значення, оскільки користувач постійно взаємодіє з полотном: додає вузли, переміщує елементи, створює зв'язки, відкриває панель властивостей і змінює параметри моделі. Компонентна структура React дозволяє ізолювати окремі частини інтерфейсу та оновлювати лише ті елементи, стан яких змінився. Це підвищує зручність супроводу коду та створює основу для подальшої оптимізації продуктивності редактора під час роботи зі складними графовими структурами.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Альтернативами React є Angular і Vue. Angular є повноцінним фреймворком із жорсткою структурою, що може бути перевагою для великих корпоративних систем, проте ускладнює початкову розробку та має вищий поріг входу. Vue є простішим у використанні, однак менш формалізованим для побудови складних масштабованих систем. Порівняння зазначених технологій наведено в таблиці 1.4.

Таблиця 1.4.

Порівняння frontend-технологій

Критерій	React	Angular	Vue
Тип	Бібліотека	Фреймворк	Фреймворк
Складність	Середня	Висока	Низька
Масштабованість	Висока	Висока	Середня
Гнучкість	Висока	Середня	Висока
Підтримка TypeScript	Вбудована	Вбудована	Часткова

Для побудови та запуску клієнтського застосунку використовується Vite, який забезпечує швидке середовище розробки та ефективну збірку проєкту. Для маршрутизації між сторінками використовується React Router, а для взаємодії з backend API – Axios. Для побудови інтерактивного редактора моделей доцільним є використання React Flow, який надає готові механізми для роботи з вузлами, зв'язками, полотном, масштабуванням і переміщенням елементів [23].

Для зберігання даних обрано PostgreSQL, що є реляційною системою керування базами даних і забезпечує надійність, підтримку транзакцій, складних зв'язків між сутностями та JSON-полів [24]. Альтернативним підходом є використання NoSQL-баз даних, наприклад MongoDB. Такі рішення забезпечують високу гнучкість схеми, однак можуть ускладнювати роботу зі складними зв'язками між об'єктами, що є характерними для графів, ієрархій і моделей залежностей.

Порівняння СКБД та підходів до зберігання даних наведено у таблиці 1.5.

Порівняння підходів до зберігання даних

Критерій	PostgreSQL	MongoDB
Тип	Реляційна база даних	NoSQL-база даних
Робота зі зв'язками	Ефективна	Обмежена
Транзакції	Підтримуються	Підтримуються частково залежно від сценарію
Гнучкість структури	Середня	Висока
Відповідність графовим та ієрархічним моделям	Висока	Середня
Критерій	PostgreSQL	MongoDB

Для взаємодії з базою даних використовується Prisma ORM, що дозволяє описувати схему даних у формалізованому вигляді, виконувати міграції та працювати з базою даних через типізований клієнт [25]. Це зменшує кількість помилок під час роботи з SQL-запитами та підвищує підтримуваність backend-коду.

Серверну частину системи реалізовано на основі NestJS. Цей фреймворк побудований поверх Node.js і підтримує TypeScript, модульну архітектуру, dependency injection, контролери, сервіси, guards і middleware [26]. NestJS добре відповідає вимогам системи, оскільки дозволяє чітко розділити функціональні модулі: автентифікацію, користувачів, моделі, вузли, зв'язки, атрибути, спільний доступ, адміністрування та взаємодію в реальному часі.

Альтернативою NestJS є Express або Spring Boot. Express є простішим і менш формалізованим рішенням, однак потребує самостійної організації архітектури, що може ускладнювати підтримку великого проєкту. Spring Boot є потужним фреймворком для Java-екосистеми, проте для даного проєкту його використання збільшило б складність розробки та вимагало б окремого технологічного стеку.

Порівняння backend-технологій наведено в таблиці 1.6.

Таблиця 1.6.

Порівняння backend-технологій

Критерій	NestJS	Express	Spring Boot
Мова розробки	TypeScript	JavaScript/ TypeScript	Java
Архітектурна структура	Вбудована модульність	Потребує ручної організації	Вбудована модульність
Підтримка dependency injection	Вбудована	Обмежена	Вбудована
Масштабованість	Висока	Середня	Висока

Для реалізації взаємодії в реальному часі використовується WebSocket, що дозволяє встановити постійне двонаправлене з'єднання між клієнтом і сервером [27]. Такий підхід є доцільним для інтерактивного редактора моделей, оскільки дає змогу миттєво передавати оновлення стану моделі без періодичних HTTP-запитів. Альтернативним підходом є polling, однак він створює додаткове навантаження на сервер і не забезпечує такої швидкої реакції. Порівняння методів взаємодії наведено в таблиці 1.7.

Таблиця 1.7.

Порівняння методів взаємодії в реальному часі

Критерій	WebSocket	HTTP polling
Затримка	Низька	Вища
Навантаження на сервер	Нижче	Вище
Реальний час	Підтримується	Обмежено
Двонаправлена взаємодія	Підтримується	Не є природною
Доцільність для інтерактивного редактора	Висока	Низька

Для створення десктопної версії системи обрано Electron [27]. Це дозволяє повторно використовувати React-інтерфейс у настільному застосунку та забезпечити локальну роботу з моделями. Для локального зберігання даних у десктопній версії доцільним є використання SQLite, оскільки ця база даних не потребує окремого серверного процесу та добре підходить для локального зберігання структурованих даних [29].

Для контейнеризації системи використовується Docker. Його застосування дозволяє уніфікувати середовище запуску frontend, backend і бази даних, спростити розгортання, ізолювати компоненти системи та забезпечити перенесення проєкту між різними середовищами [30].

Таким чином, обраний технологічний стек, що включає React, TypeScript, NestJS, PostgreSQL, Prisma ORM, WebSocket, Electron і Docker, відповідає функціональним, нефункціональним та архітектурним вимогам системи. Він забезпечує інтерактивність, надійне зберігання даних, кросплатформеність, масштабованість, безпеку та створює основу для подальшого розвитку програмного продукту.

Висновки до першого розділу.

У першому розділі було проведено комплексний аналіз предметної області візуального моделювання інформації, що дозволило визначити основні підходи до представлення структурованих даних, зокрема у вигляді графів, дерев та часових ліній. Встановлено, що використання візуальних моделей значно підвищує ефективність сприйняття та аналізу складної інформації.

У процесі дослідження існуючих програмних рішень було встановлено, що сучасні системи здебільшого орієнтовані на вузькоспеціалізовані задачі або не забезпечують достатнього рівня формалізації та інтерактивності.

На основі проведеного аналізу було сформовано сукупність функціональних та нефункціональних вимог до розроблюваної системи. Визначено, що система повинна забезпечувати інтерактивну роботу з моделями, підтримку різних типів структур, кросплатформеність, масштабованість та високий рівень продуктивності.

У підсумку було сформульовано постановку задачі, яка полягає у розробці інтерактивної кросплатформеної системи для візуального моделювання інформації, а також визначено мету та основні задачі розробки. Отримані результати створюють теоретичну та практичну основу для подальшого проєктування архітектури системи, що буде розглянуто у наступному розділі.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Визначення варіантів використання системи

Визначення варіантів використання системи є важливим етапом проектування, оскільки дозволяє формалізувати основні сценарії взаємодії користувачів із програмним продуктом. Для інтерактивної системи візуального моделювання інформації такі сценарії охоплюють створення, редагування, збереження, перегляд, поширення, адміністрування та синхронізацію інформаційних моделей. На відміну від опису інтерфейсу, який буде наведено у третьому розділі, у цьому підрозділі розглядається саме функціональна поведінка системи з погляду акторів і доступних їм дій.

Система GraphoGі передбачає декілька типів користувачів, які мають різний рівень доступу до функціональності. Основними акторами системи є гість, авторизований користувач, користувач десктопної версії, адміністратор і суперадміністратор. Гість має мінімальний набір можливостей і може працювати лише з відкритими даними системи. Авторизований користувач є основним актором, оскільки саме він створює та редагує інформаційні моделі. Адміністратор виконує функції модерації та керування користувачами, а суперадміністратор має повний контроль над адміністративними ролями та дозволами. Узагальнену характеристику акторів системи наведено в таблиці 2.1.

Таблиця 2.1.

Актори системи та їх призначення

Актор	Характеристика	Основна роль у системі
Гість	Неавторизований користувач	Перегляд публічних моделей, реєстрація та вхід до системи
Авторизований користувач	Користувач із власним обліковим записом	Створення, редагування, збереження, імпорт, експорт і поширення моделей
Користувач десктопної версії	Авторизований користувач, який працює через Electron-застосунок	Локальне збереження моделей і синхронізація з сервером

Адміністратор	Користувач із розширеними правами	Модерація публічних моделей і керування користувачами
Суперадміністратор	Користувач із найвищим рівнем доступу	Керування адміністраторами, ролями та дозволами

Основні варіанти використання системи пов'язані з життєвим циклом інформаційної моделі. Користувач створює модель, визначає її тип, додає вузли, встановлює зв'язки між елементами, редагує властивості, зберігає зміни, за потреби експортує модель або надає доступ іншим користувачам. Для десктопного сценарію додатково передбачено локальне збереження та синхронізацію моделі з серверною частиною (див. додаток Б табл. Б.1).

Діаграма варіантів використання системи наведена у додатку А рис. А.2. Вона відображає основних акторів системи та сценарії, які вони можуть виконувати. Наведена діаграма демонструє, що система орієнтована не лише на базове створення моделей, а й на повний цикл роботи з ними. Основний актор – авторизований користувач – може створювати моделі, редагувати їх структуру, працювати з вузлами та зв'язками, виконувати імпорт, експорт, версіонування і керування доступом. Гість має обмежені можливості, пов'язані з переглядом публічного контенту та автентифікацією. Адміністратор і суперадміністратор мають додаткові сценарії, пов'язані з керуванням користувачами, модерацією та адміністративними дозволами.

Варіант використання “Створення моделі” є початковим сценарієм для більшості подальших дій користувача. У межах цього сценарію користувач задає назву моделі, опис і тип структури. Тип моделі визначає подальшу логіку роботи редактора: для таймлайну важливими є часові параметри, для генеалогії – персональні дані та родинні зв'язки, для організаційної ієрархії – підпорядкування та посади, а для графової моделі – довільні вузли та зв'язки.

Варіант використання “Редагування моделі” є комплексним, оскільки включає декілька підсценаріїв. Користувач може додавати нові вузли, змінювати

їх назву, тип, атрибути, координати та спеціалізовані властивості. Також користувач може створювати зв'язки між вузлами, змінювати їх тип, напрям, підпис і метадані. Редагування моделі передбачає постійну взаємодію клієнтської частини із серверною логікою та базою даних.

Варіант використання “Керування спільним доступом” дозволяє власнику моделі визначати, хто може переглядати або редагувати модель. У системі передбачено приватні, публічні та спільні моделі. Приватна модель доступна лише власнику, публічна може бути доступна для перегляду іншим користувачам, а спільна модель надається конкретним користувачам із визначеним рівнем доступу. Такий підхід дозволяє гнучко керувати доступністю інформаційних моделей.

Варіант використання “Локальне збереження моделі” характерний для десктопної версії системи. У цьому сценарії користувач може працювати з моделлю на власному пристрої та зберігати її в локальному сховищі. Варіант використання “Синхронізація із сервером” доповнює локальну роботу та дозволяє передати модель до серверної частини або оновити локальний стан відповідно до хмарної версії.

Адміністративні варіанти використання пов'язані з підтримкою порядку в системі та контролем доступу. Адміністратор може переглядати публічні моделі, змінювати їх статус модерації, обмежувати користувачів або забороняти їм створення публічних моделей. Суперадміністратор має можливість керувати адміністраторами та призначати окремі дозволи, що забезпечує гнучке розмежування адміністративних функцій (див. додат. Б табл. Б.3).

Окрему увагу слід приділити сценаріям, пов'язаним із різними типами моделей. Незважаючи на те, що базова логіка роботи з моделями є спільною, конкретна поведінка системи залежить від обраного типу структури. Для мережевої моделі основним сценарієм є довільне створення об'єктів і зв'язків між ними. Для генеалогічної моделі важливими є сценарії створення осіб, визначення родинних зв'язків і редагування персональних даних. Для організаційної ієрархії ключовим є створення працівників, визначення

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

керівників і фільтрація за підрозділами. Для таймлайну основними є створення подій, часових проміжків і періодів.

Відповідність типів моделей ключовим сценаріям роботи наведено в таблиці 2.2.

Таблиця 2.2.

Варіанти використання для різних типів моделей

Тип моделі	Основні дії користувача	Результат
Мережева модель	Створення об'єктів, встановлення довільних зв'язків, переміщення вузлів	Побудована мережа взаємопов'язаних об'єктів
Генеалогічна модель	Додавання осіб, створення родинних зв'язків, редагування персональних даних	Побудоване генеалогічне дерево
Організаційна ієрархія	Додавання працівників, визначення керівників, фільтрація за відділами	Побудована структура підпорядкування
Таймлайн	Створення подій, періодів і часових проміжків	Побудована хронологічна послідовність
Ієрархічна структура	Створення батьківських і дочірніх вузлів	Побудована деревоподібна структура
Модель залежностей	Створення напрямлених зв'язків між елементами	Відображено залежності між об'єктами

Варіанти використання також пов'язані з моделлю доступу. Одна й та сама дія може бути дозволена або заборонена залежно від ролі користувача, типу доступу до моделі та її статусу. Наприклад, гість може переглядати лише публічні моделі, авторизований користувач може редагувати власні моделі, а спільний доступ може надавати право лише на перегляд або також на редагування. Адміністративні дії доступні лише користувачам із відповідними ролями та дозволами.

Такий підхід дозволяє поєднати гнучкість спільної роботи з необхідним рівнем захисту даних. Кожна модель має визначеного власника, який контролює її доступність для інших користувачів. При цьому система повинна розмежовувати дії перегляду, редагування, публікації, модерації та

адміністрування, оскільки вони мають різний рівень впливу на дані. Саме тому для формалізації прав доступу доцільно використати матрицю, у якій відображено, які дії доступні кожному типу користувача.

У таблиці 2.3 наведено узагальнену матрицю доступу до основних дій.

Таблиця 2.3.

Матриця доступу до основних функцій системи

Дія	Гість	Користувач	Адміністратор	Суперадміністратор
Перегляд публічних моделей	Так	Так	Так	Так
Реєстрація та вхід	Так	Так	Так	Так
Створення власних моделей	Ні	Так	Так	Так
Редагування власних моделей	Ні	Так	Так	Так
Керування спільним доступом	Ні	Так	Так	Так
Імпорт та експорт моделей	Ні	Так	Так	Так
Локальне збереження в desktop-версії	Ні	Так	Так	Так
Модерація публічних моделей	Ні	Ні	Так	Так
Керування користувачами	Ні	Ні	Так	Так
Керування адміністраторами	Ні	Ні	Ні	Так
Керування адміністративними дозволами	Ні	Ні	Ні	Так

Визначені варіанти використання впливають на архітектуру системи та її реалізацію. Сценарії створення та редагування моделей потребують наявності візуального редактора, API для роботи з вузлами й зв'язками та механізму збереження даних. Сценарії спільного доступу та модерації вимагають наявності рольової моделі, адміністративних дозволів і статусів публікації моделей.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

Сценарії локального збереження та синхронізації визначають необхідність окремої десктопної частини, локального сховища й сервісу обміну даними із сервером.

Таким чином, визначення варіантів використання дозволяє формалізувати функціональну поведінку системи GraphoGі та встановити зв'язок між ролями користувачів, доступними діями, типами моделей і архітектурними компонентами. Надалі ці сценарії використовуються як основа для демонстрації роботи системи, опису порядку взаємодії користувача з інтерфейсом і формування тестових сценаріїв у третьому розділі.

2.2. Загальна архітектура системи

Для розроблюваної системи передбачено клієнт-серверну архітектуру з додатковою підтримкою десктопного режиму роботи. Така структура дозволяє розділити інтерфейс користувача, серверну логіку, рівень зберігання даних і локальну роботу десктопного застосунку.

У межах цієї архітектури клієнтська частина відповідає за відображення візуального редактора та обробку дій користувача, серверна частина – за бізнес-логіку, перевірку доступу та обробку даних, а база даних – за централізоване зберігання інформації про користувачів, моделі, вузли, зв'язки, атрибути та версії моделей.

Система орієнтована на роботу з різними типами інформаційних моделей: графами, ієрархічними структурами, часовими лініями, генеалогічними моделями, організаційними структурами та мережами об'єктів. Тому архітектура повинна підтримувати не лише стандартні операції створення, читання, оновлення та видалення даних, а й інтерактивне редагування елементів, збереження координат вузлів, роботу з типами зв'язків, керування доступом, модерацію публічних моделей і синхронізацію десктопної версії із сервером.

Основні сценарії використання системи пов'язані з різними компонентами програмної архітектури (див. додат. Б.4). Для виконання кожного сценарію залучаються клієнтський інтерфейс, серверна логіка, база даних, модулі

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

автентифікації, редактор моделей або адміністративна підсистема. Такий розподіл дозволяє показати, які частини системи забезпечують виконання конкретних дій користувача.

Наведена таблиця не дублює перелік акторів системи, а демонструє зв'язок між функціональними сценаріями та архітектурними компонентами, які забезпечують їх виконання. Це дозволяє простежити, які частини системи беруть участь у реалізації кожного сценарію: від клієнтського інтерфейсу та редактора моделей до серверних сервісів, бази даних, адміністративної підсистеми та десктопного застосунку.

Загальна структура системи включає чотири логічні рівні: клієнтський рівень, серверний рівень, рівень зберігання даних і локальний рівень десктопного застосунку. Їх призначення наведено в таблиці 2.4.

Таблиця 2.4.

Логічні рівні архітектури системи

Рівень	Складові	Призначення
Клієнтський рівень	Web client, Desktop client	Відображення інтерфейсу, робота з редактором моделей, обробка дій користувача
Серверний рівень	REST API, WebSocket Gateway, модулі backend	Обробка запитів, бізнес-логіка, перевірка доступу, робота з моделями
Рівень зберігання даних	PostgreSQL, Prisma Service	Централізоване зберігання користувачів, моделей, вузлів, зв'язків і службових даних
Локальний рівень desktop	SQLite, Electron IPC, Sync Service	Локальне збереження моделей і синхронізація з сервером

Клієнтський рівень представлений вебзастосунком і десктопним застосунком. Вебзастосунок забезпечує роботу користувача через браузер і є основним способом доступу до системи. Десктопний застосунок використовує той самий підхід до побудови інтерфейсу, але додатково має можливість

зберігати моделі локально. Такий поділ дозволяє використовувати систему як у звичайному вебсередовищі, так і у форматі настільного застосунку.

Серверний рівень реалізує прикладну логіку системи. До його складу входять підсистеми автентифікації та керування користувачами, керування моделями, адміністрування, взаємодії в реальному часі та доступу до даних. Підсистема керування моделями є центральною, оскільки саме вона обробляє операції над моделями, вузлами, зв'язками, атрибутами, параметрами відображення та версіями. Підсистема адміністрування забезпечує керування користувачами, модерацію публічних моделей і налаштування адміністративних дозволів.

Рівень зберігання даних представлений PostgreSQL. У базі даних зберігається інформація про облікові записи користувачів, створені моделі, вузли, зв'язки, атрибути, налаштування відображення, версії моделей, правила спільного доступу та адміністративні дозволи. Серверна частина взаємодіє з базою даних через Prisma Service, що виконує роль єдиного рівня доступу до даних.

Локальний рівень десктопного застосунку призначений для збереження моделей безпосередньо на пристрої користувача. Для цього використовується локальна база SQLite. Взаємодія між інтерфейсом Electron-застосунку та локальними можливостями операційної системи відбувається через preload API та IPC-механізм. Синхронізація локальних моделей із сервером виконується через окремий сервіс синхронізації.

Компонентна діаграма системи подана на рисунку 2.1. Вона відображає основні підсистеми та напрями взаємодії між ними.



Рис. 2.1. Компонентна діаграма системи

Компонентна діаграма демонструє, що система не є монолітною. Клієнтські застосунки взаємодіють із серверною частиною через чітко визначені інтерфейси. REST API використовується для основних операцій над даними, а WebSocket – для оновлення стану моделі в реальному часі. Такий поділ дозволяє не змішувати стандартну обробку запитів із механізмами інтерактивної взаємодії.

На серверному рівні підсистеми згруповані за функціональним призначенням. Підсистема автентифікації та користувачів обробляє реєстрацію, вхід, перевірку JWT-токенів і профілі користувачів [30]. Підсистема керування моделями відповідає за основну предметну логіку: створення моделей, редагування вузлів, формування зв'язків, збереження атрибутів, імпорт, експорт і версіонування. Підсистема адміністрування охоплює керування користувачами, модерацію публічних моделей і роботу з адміністративними дозволами.

Важливим архітектурним елементом є рівень доступу до даних. Усі серверні підсистеми не працюють із PostgreSQL безпосередньо, а використовують Prisma Service. Це дозволяє централізувати роботу з базою даних, зменшити дублювання коду та забезпечити єдиний підхід до виконання операцій читання і запису.

Діаграма розгортання системи подана на рисунку 2.2. Вона демонструє фізичне розміщення основних компонентів системи у середовищі виконання.

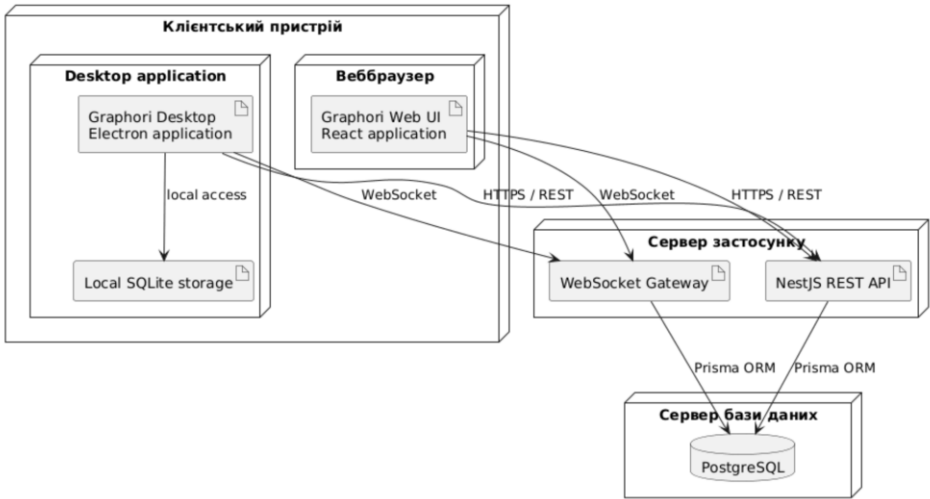


Рис. 2.2. Діаграма розгортання (Deployment Diagram)

Діаграма розгортання уточнює, що клієнтська частина може функціонувати у двох формах: як вебзастосунок у браузері та як десктопний застосунок. У першому випадку всі операції збереження та отримання даних виконуються через сервер. У другому випадку частина даних може зберігатися локально, що дозволяє користувачу працювати з моделями на власному пристрої та виконувати синхронізацію із сервером за потреби.

Сервер застосунку включає REST API та WebSocket Gateway. REST API обробляє запити, пов'язані з користувачами, моделями, вузлами, зв'язками, атрибутами та адміністративними діями. WebSocket Gateway використовується для передавання повідомлень про зміни стану моделі. Сервер бази даних винесений в окремий вузол, оскільки зберігання даних є самостійним рівнем системи та може масштабуватися незалежно від прикладної логіки.

Діаграма послідовності, наведена на рисунку 2.3, описує сценарій створення вузла в моделі. Цей сценарій є базовим для більшості типів моделей, оскільки вузли є основними елементами графів, дерев, таймлайнів, генеалогій і мережевих структур.

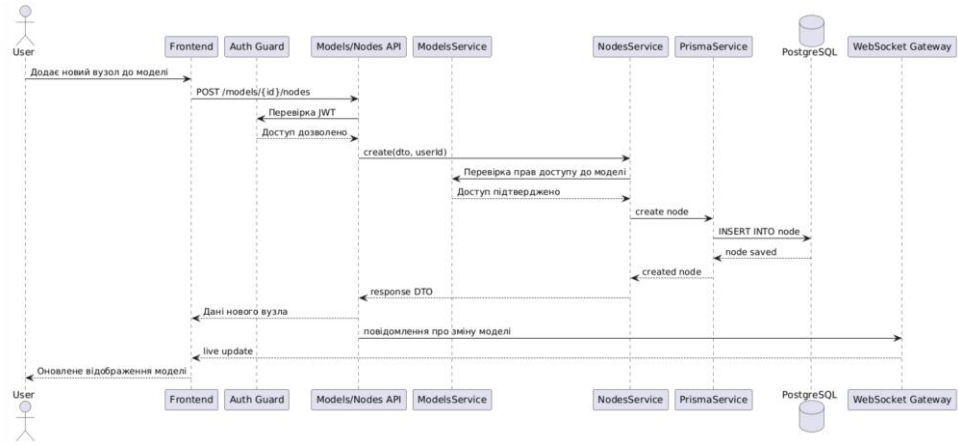


Рис. 2.3. Діаграма послідовності взаємодії

Послідовність створення вузла демонструє проходження запиту через основні архітектурні рівні системи. Спочатку користувач виконує дію у візуальному редакторі. Клієнтська частина формує HTTP-запит і передає його до серверної частини. Перед виконанням операції сервер перевіряє автентифікацію користувача та його право доступу до відповідної моделі.

Після успішної перевірки NodesService виконує створення вузла через PrismaService, а запис зберігається у PostgreSQL. Після цього результат повертається до клієнта. Окремо формується повідомлення через WebSocket Gateway, яке дозволяє оновити стан моделі в реальному часі. Такий сценарій показує поєднання REST-взаємодії, перевірки доступу, збереження даних і механізму live-оновлення.

Рух даних між основними підсистемами відображено на рисунку 2.4.

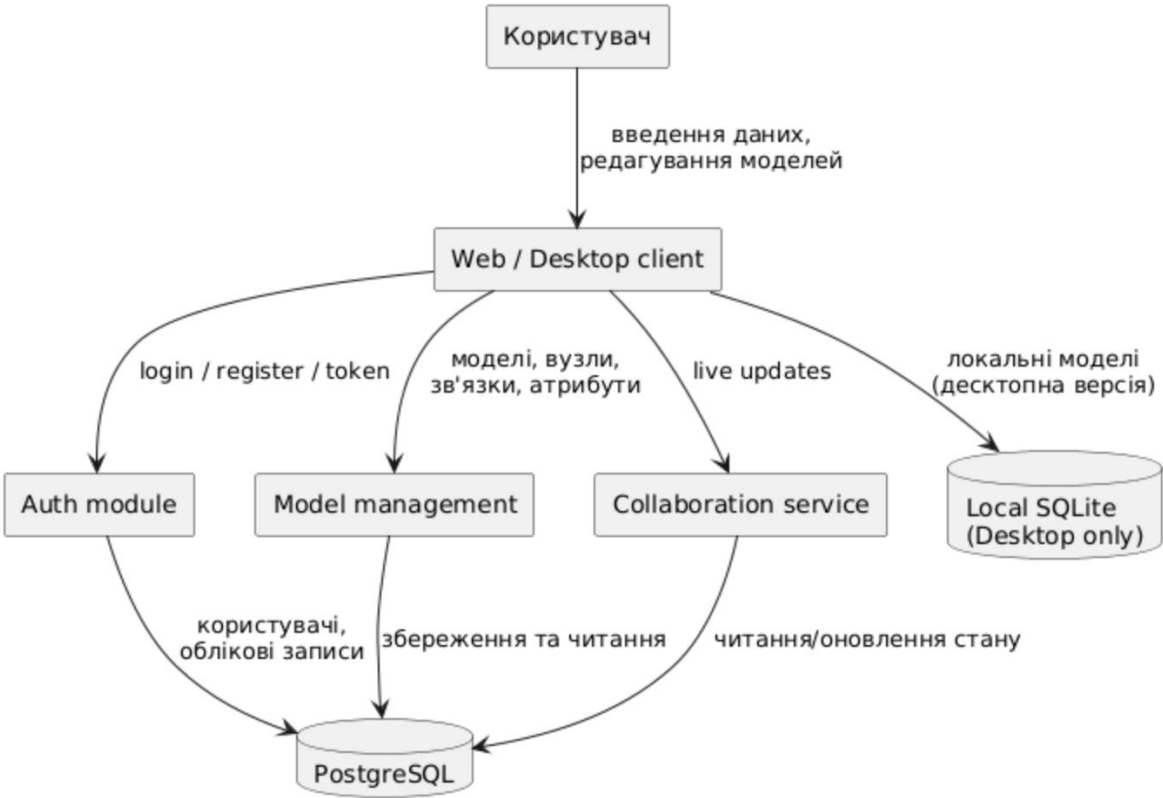


Рис. 2.4. Діаграма потоків даних (Data Flow)

Діаграма потоків даних показує, які групи інформації переміщуються між користувачем, клієнтською частиною, серверними модулями та сховищами даних. Дані автентифікації передаються до модуля автентифікації, після чого зберігаються або перевіряються у PostgreSQL. Дані моделей, вузлів, зв'язків і атрибутів обробляються підсистемою керування моделями та також зберігаються в централізованій базі даних.

Для інтерактивного режиму окремо виділено потік live-оновлень. Він не замінює основне збереження даних, а доповнює його, забезпечуючи швидке оновлення стану редактора. Для десктопної версії додатково передбачено

локальний потік даних між клієнтом і SQLite-сховищем. Це дозволяє відокремити локальну роботу користувача від серверної синхронізації.

Узагальнення відповідності архітектурних компонентів функціональним частинам системи наведено в таблиці 2.5.

Таблиця 2.5.

**Відповідність компонентів архітектури
функціональним частинам системи**

Компонент	Основна відповідальність	Дані, з якими працює компонент
Вебклієнт	Робота користувача через браузер	Моделі, вузли, зв'язки, профіль користувача
Десктопний клієнт	Локальна робота з моделями та синхронізація	Локальні моделі, токени, дані синхронізації
REST API	Обробка стандартних запитів клієнта	Користувачі, моделі, вузли, зв'язки, атрибути
WebSocket Gateway	Передавання оновлень у реальному часі	Події зміни моделей і стан редактора
Підсистема автентифікації	Реєстрація, вхід, перевірка користувача	Облікові записи, токени, OAuth-дані
Підсистема керування моделями	Основна робота з інформаційними моделями	Моделі, вузли, зв'язки, атрибути, версії
Підсистема адміністрування	Модерація та керування користувачами	Користувачі, ролі, дозволи, статуси моделей
Prisma Service	Єдиний рівень доступу до бази даних	Усі сутності PostgreSQL
PostgreSQL	Централізоване зберігання даних	Дані користувачів, моделей, зв'язків і налаштувань
SQLite	Локальне зберігання в desktop-версії	Локальні копії моделей

Запропонована архітектура забезпечує узгоджену взаємодію між усіма частинами системи. Веб і десктопний клієнти використовують однакову серверну логіку, що дозволяє уникнути дублювання правил обробки даних.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Серверна частина централізує бізнес-логіку, перевірку доступу та роботу з базою даних. Локальне сховище desktop-версії розширює можливості системи, але не порушує загальну модель централізованого зберігання, оскільки синхронізація виконується через backend.

Таким чином, у підрозділі визначено загальну архітектуру системи, основні сценарії використання, логічні рівні, компоненти, потоки даних і взаємодію між ними.

2.3. Проектування програмної архітектури

Архітектура системи Graphorі проектується за модульним принципом. Кожна частина системи має власну зону відповідальності та взаємодіє з іншими частинами через визначені інтерфейси. Такий підхід дозволяє уникнути надмірної зв'язаності коду, спростити підтримку програмного продукту та забезпечити можливість подальшого розширення функціональності. Для системи, яка повинна підтримувати різні типи інформаційних моделей, спільний доступ, адміністрування, десктопну роботу та синхронізацію, модульна організація є необхідною умовою стабільного розвитку.

Серверна частина системи побудована як набір функціональних модулів. Кожен модуль відповідає за окрему частину предметної області та містить відповідні контролери, сервіси, DTO, guards або допоміжні класи. Контролери приймають HTTP-запити, сервіси реалізують основну прикладну логіку, DTO визначають структуру вхідних даних, а guards забезпечують перевірку автентифікації та прав доступу. Доступ до бази даних централізовано через Prisma Service, що дозволяє відокремити бізнес-логіку від конкретних операцій збереження даних (див. табл. 2.6).

Таблиця 2.6.

Основні модулі серверної частини системи

Модуль	Основне призначення
AuthModule	Реєстрація, вхід, JWT-автентифікація, Google OAuth
UsersModule	Керування профілями користувачів і персональними налаштуваннями

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

ModelsModule	Створення, оновлення, видалення, імпорт, експорт і версіонування моделей
NodesModule	Керування вузлами моделей та їх базовими властивостями
RelationsModule	Керування зв'язками між вузлами та перевірка допустимості відношень
NodeAttributesModule	Робота із системними та користувацькими атрибутами вузлів
GraphSharingModule	Надання спільного доступу до моделей іншим користувачам
GraphLayoutModule	Обробка параметрів розміщення вузлів і зв'язків
AdminModule	Адміністративні функції, модерація публічних моделей, керування користувачами
PermissionsModule	Керування адміністративними дозволами
CollaborationModule	Підтримка взаємодії в реальному часі через WebSocket
PrismaModule	Централізований доступ до бази даних

Серверна архітектура передбачає розділення на рівень представлення, рівень прикладної логіки, рівень предметної логіки та інфраструктурний рівень. Рівень представлення відповідає за прийом HTTP-запитів і WebSocket-повідомлень. Рівень прикладної логіки містить сервіси, які виконують основні операції системи. Рівень предметної логіки охоплює правила роботи з моделями, вузлами, зв'язками та доступом. Інфраструктурний рівень забезпечує взаємодію з базою даних.

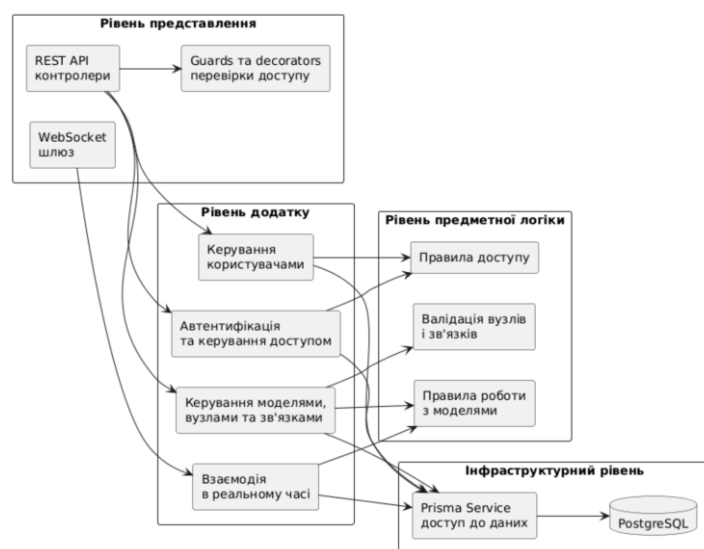


Рис. 2.5. UML Component Diagram (Backend)

Компонентна діаграма серверної архітектури демонструє, що backend-частина не є набором незалежних контролерів, які напряду взаємодіють із базою даних. Запит проходить через рівень представлення, де виконується маршрутизація, автентифікація та перевірка доступу. Після цього дані передаються до прикладного рівня, де розташовані сервіси, відповідальні за основну логіку системи. Предметна логіка використовується для перевірки бізнес-обмежень: доступу до моделей, допустимості типів вузлів, правил створення зв'язків і обмежень для різних типів моделей. Інфраструктурний рівень ізолює роботу з PostgreSQL через Prisma Service.

Клієнтська частина також має модульну структуру. Вона складається зі сторінок застосунку, UI-компонентів, API-сервісів, модуля авторизації, модуля локалізації, підсистеми візуального редактора та WebSocket-клієнта [26]. Основним функціональним центром frontend є редактор моделей, який забезпечує відображення вузлів і зв'язків, обробку дій користувача, роботу з панеллю властивостей та синхронізацію зі станом backend.

Структурні елементи клієнтської частини наведено в таблиці 2.7.

Таблиця 2.7.

Основні складові клієнтської частини системи

Складова	Призначення
Сторінки застосунку	Відображення основних екранів: головної сторінки, редактора моделей, профілю, адміністративної панелі
Спільні UI-компоненти	Повторно використовувані елементи інтерфейсу
Модуль авторизації	Збереження стану користувача, обробка входу, виходу та перевірки сесії
Модуль локалізації	Підтримка української та англійської мов інтерфейсу
API-сервіси	Ізольована взаємодія з backend через HTTP-запити
Редактор моделей	Побудова, редагування та відображення візуальних моделей
WebSocket-клієнт	Отримання подій оновлення в реальному часі

Важливим рішенням клієнтської архітектури є ізоляція доступу до backend API. Компоненти інтерфейсу не повинні самостійно формувати всі HTTP-запити,

тому ця логіка винесена в окремі API-сервіси. Це дозволяє змінювати спосіб взаємодії з сервером без суттєвого впливу на візуальні компоненти. Аналогічно, логіка авторизації та локалізації винесена в окремі модулі, що зменшує дублювання коду та спрощує підтримку застосунку.

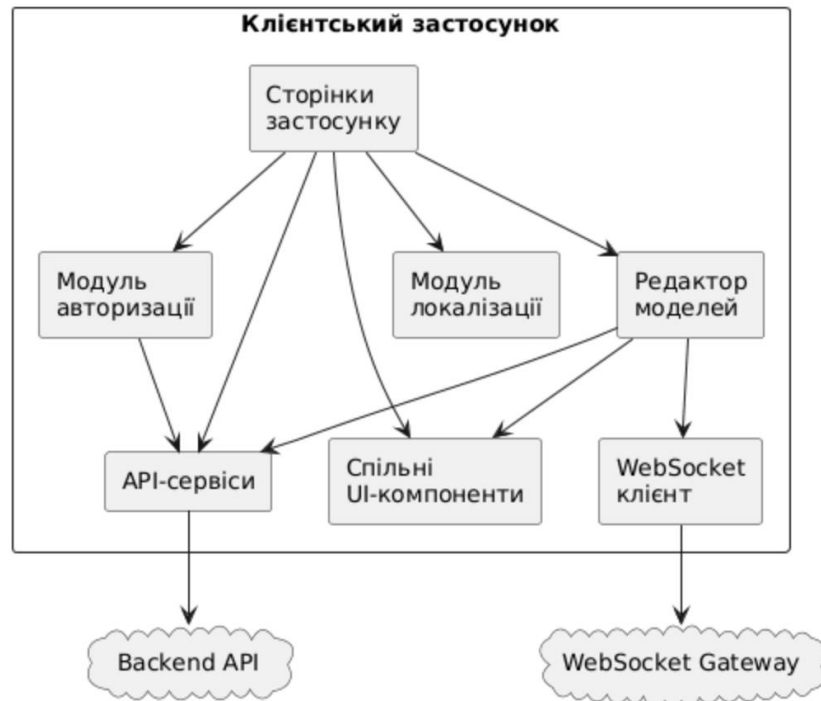


Рис. 2.6. Компонентна діаграма клієнтської архітектури

Діаграма клієнтської архітектури показує, що frontend-рівень складається з декількох взаємопов'язаних частин. Сторінки застосунку формують загальну структуру інтерфейсу, а спільні UI-компоненти забезпечують повторне використання елементів. Редактор моделей взаємодіє з API-сервісами для збереження змін і з WebSocket-клієнтом для отримання live-оновлень. Така організація дозволяє відокремити візуальну логіку від логіки обміну даними.

Окремим архітектурним елементом є десктопна версія системи. Вона використовує React-інтерфейс, але має додатковий Electron-рівень, що складається з renderer-процесу, preload API, IPC-обробників, локального сховища SQLite, сховища токенів, сервісу синхронізації та cloud-клієнта. Така структура дозволяє користувачу працювати з моделями локально та передавати їх на сервер за потреби.

Особливість десктопної архітектури полягає у тому, що renderer-процес не отримує прямого доступу до можливостей Node.js або операційної системи. Усі операції з локальними файлами, токенами, SQLite-сховищем і синхронізацією виконуються через preload API та IPC-виклики. Це зменшує ризики безпеки та дозволяє ізолювати інтерфейс користувача від системного рівня.

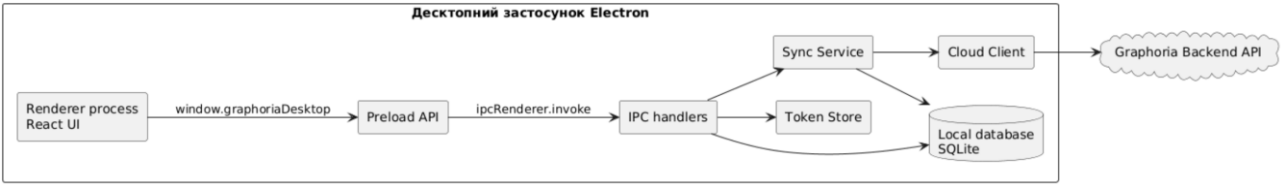


Рис. 2.7. Компонентна діаграма десктопної архітектури

Діаграма десктопної архітектури відображає внутрішню організацію Electron-застосунку. Renderer process відповідає за інтерфейс користувача. Preload API є контрольованим мостом між інтерфейсом і системними можливостями застосунку. IPC handlers обробляють запити на локальне збереження, читання моделей, роботу з токенами та синхронізацію. SQLite використовується для локального зберігання моделей, а Cloud Client забезпечує взаємодію із серверною частиною. Узагальнену відповідність архітектурних частин програмної системи їх відповідальності наведено в таблиці 2.8.

Таблиця 2.8.

Розподіл відповідальностей між частинами програмної архітектури

Частина системи	Основна відповідальність	Що не входить до її відповідальності
Frontend	Відображення інтерфейсу, обробка дій користувача, робота з редактором	Прямий доступ до бази даних і виконання серверної бізнес-логіки
Backend controllers	Прийом і маршрутизація HTTP-запитів	Зберігання складної бізнес-логіки
Backend services	Обробка бізнес-логіки, перевірка правил, робота з моделями	Відображення інтерфейсу користувача

Guards та decorators	Автентифікація, авторизація, отримання поточного користувача	Обробка предметної логіки моделей
Prisma Service	Централізований доступ до PostgreSQL	Формування UI або прийняття рішень інтерфейсу
WebSocket Gateway	Передавання подій у реальному часі	Заміна основного механізму збереження даних
Electron preload / IPC	Безпечний доступ до локальних можливостей desktop-застосунку	Безпосереднє відображення інтерфейсу

Важливим принципом програмної архітектури є розділення відповідальностей. Контролери не повинні містити складної бізнес-логіки, оскільки їх завдання полягає у прийомі запиту та передачі даних до сервісів. Сервіси реалізують основні правила роботи системи, але не відповідають за відображення інтерфейсу. Frontend-компоненти працюють із даними через API-сервіси та не мають прямого доступу до бази даних. У десктопній версії renderer-процес не працює напряму з локальним сховищем, а використовує контрольований preload API.

Застосована архітектура дозволяє розвивати систему поетапно. Додавання нового типу моделі може виконуватися через розширення логіки редактора, правил валідації, типів вузлів і зв'язків, не змінюючи базову структуру взаємодії між frontend, backend і базою даних. Додавання нових адміністративних функцій може бути реалізоване в межах AdminModule та PermissionsModule без впливу на основну логіку редагування моделей. Розвиток десктопної версії також може відбуватися незалежно, оскільки локальне збереження та синхронізація відокремлені від вебінтерфейсу.

Таким чином, програмна архітектура системи Graphori забезпечує чітке розділення відповідальностей між клієнтською, серверною та десктопною частинами. Серверна частина реалізує бізнес-логіку та доступ до даних,

клієнтська частина забезпечує інтерактивну взаємодію користувача з моделями, а десктопна частина розширює систему можливістю локальної роботи й синхронізації. Такий підхід підвищує підтримуваність програмного продукту та створює основу для подальшого розширення функціональності.

2.4. Проектування структури даних

Проектування структури даних є одним із ключових етапів створення інтерактивної системи візуального моделювання інформації, оскільки саме модель даних визначає спосіб зберігання інформаційних моделей, їх елементів, зв'язків, атрибутів, налаштувань відображення та прав доступу. На відміну від простих інформаційних систем, де основні дані мають лінійну або табличну форму, система Graphori повинна зберігати складні взаємопов'язані структури: графи, дерева, таймлайни, генеалогічні моделі, організаційні ієрархії та мережі об'єктів.

Основним завданням проектування структури даних є створення такої схеми, яка одночасно забезпечує цілісність інформації, підтримку зв'язків між сутностями, можливість розширення системи та адаптацію до різних типів моделей.

Для цього використовується реляційний підхід до організації даних, у межах якого основні сутності винесено в окремі таблиці, а зв'язки між ними реалізовано через зовнішні ключі. Такий підхід дозволяє зберігати моделі у формалізованому вигляді та забезпечує контроль узгодженості даних.

Центральною сутністю системи є користувач. Користувач створює інформаційні моделі, редагує їх, керує доступом і може мати власні налаштування профілю.

Окремо передбачено зберігання OAuth-акаунтів, що дозволяє використовувати зовнішні механізми автентифікації, зокрема Google OAuth [31]. Такий підхід відокремлює основні дані користувача від даних зовнішніх провайдерів входу.

Основною предметною сутністю є модель. Модель представляє окрему інформаційну структуру, наприклад граф, таймлайн, генеалогічне дерево або

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

організаційну ієрархію. Вона містить назву, опис, тип, параметри видимості, статус модерації, службові поля для м'якого видалення, а також пов'язується з вузлами, зв'язками, версіями, налаштуваннями відображення та правилами спільного доступу. Саме тип моделі визначає, які правила редагування та візуалізації будуть застосовуватися до її елементів.

Вузол є базовим елементом інформаційної моделі. Він може представляти об'єкт, особу, подію, часовий проміжок, документ, категорію або іншу сутність залежно від типу моделі.

Для того щоб не перевантажувати таблицю вузлів великою кількістю специфічних полів, у структурі даних передбачено окремі таблиці для деталізованої інформації. Наприклад, персональні дані для генеалогічних моделей зберігаються в таблиці PersonDetails, дані працівників організаційної структури – у HierarchyPersonDetails, а часові характеристики подій і періодів – у TimelineItem.

Зв'язок між вузлами представлений сутністю Relation. Вона містить посилання на початковий і кінцевий вузол, тип відношення, напрям, підпис і додаткові метадані. Завдяки цьому система може підтримувати різні види відношень: родинні зв'язки, підпорядкування, залежності, асоціації, послідовність подій та інші типи взаємозв'язків. Така модель є універсальною для різних типів структур і дозволяє розширювати набір підтримуваних зв'язків без повної зміни базової схеми даних.

Для підтримки візуального редактора окремо зберігаються дані розташування вузлів і параметри відображення зв'язків. Таблиця NodeLayout містить координати вузла, інформацію про фіксацію позиції та інші параметри розміщення. Таблиця RelationLayout використовується для зберігання параметрів візуального підключення зв'язків. Це дозволяє відокремити змістовну інформацію про модель від її графічного представлення.

Загальну логіку організації ключових сутностей наведено в таблиці 2.9.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Основні групи сутностей системи

Група даних	Сутності	Призначення
Користувацькі дані	User, UserOAuthAccount, UserEmailChange	Зберігання облікових записів, профілів, OAuth-зв'язків і змін електронної пошти
Дані моделей	Model, ModelDisplaySettings, TimelineSettings, ModelVersion	Опис інформаційних моделей, їх налаштувань, часових параметрів і версій
Елементи моделей	Node, NodeLayout, PersonDetails, HierarchyPersonDetails, TimelineItem	Зберігання вузлів, їх позицій і спеціалізованих даних для різних типів моделей
Зв'язки між елементами	Relation, RelationLayout	Зберігання логічних і візуальних параметрів зв'язків між вузлами
Атрибути	NodeAttributeDefinition, NodeAttributeValue	Опис і зберігання системних та користувацьких атрибутів вузлів
Доступ і адміністрування	GraphShare, AdminPermission	Керування спільним доступом, ролями та адміністративними дозволами

Для забезпечення розширюваності системи використовується механізм атрибутів вузлів. Сутність NodeAttributeDefinition описує доступні атрибути, їх назву, внутрішній ідентифікатор, тип поля, область застосування та ознаку обов'язковості. Сутність NodeAttributeValue зберігає конкретні значення атрибутів для окремих вузлів. Значення можуть мати різні типи: текстові, числові, логічні, датовані або JSON-значення. Такий підхід дозволяє додавати нові поля до моделей без зміни основної таблиці вузлів.

Окрему роль у структурі даних відіграє версіонування моделей. Сутність ModelVersion зберігає номер версії, опис, кількість вузлів і зв'язків, а також знімок стану моделі у форматі JSON. Це дозволяє фіксувати стан моделі на

певному етапі, відновлювати попередні варіанти та забезпечувати контроль змін. Такий підхід є важливим для систем, у яких користувачі працюють зі складними структурами та можуть виконувати значну кількість операцій редагування.

Механізм спільного доступу реалізовано через сутність GraphShare. Вона пов'язує модель із користувачем і визначає рівень доступу до неї. Це дозволяє надавати іншим користувачам право перегляду або редагування моделі без зміни її власника. Для адміністративного керування використовується сутність AdminPermission, яка зберігає набір дозволів для користувачів із адміністративними ролями.

Основні сутності, їх ключові атрибути та призначення наведено в додатку Б табл. Б.2.

Для формалізації зв'язків між сутностями створена ER-діаграма. Вона дозволяє відобразити структуру бази даних, основні таблиці та кардинальність зв'язків між ними (див. додат. А рис. А.2).

Наведена ER-діаграма демонструє, що структура даних системи побудована навколо сутності Model. Кожна модель належить конкретному користувачу та містить набір вузлів і зв'язків. Вузли є універсальними елементами моделі, але за потреби можуть мати спеціалізовані дані. Такий підхід дозволяє підтримувати різні предметні області без створення окремої незалежної структури бази даних для кожного типу моделі.

Зв'язки між вузлами зберігаються окремо від вузлів, що є важливим для роботи з графовими структурами. Один зв'язок має початковий і кінцевий вузол, тому таблиця relations містить два посилання на таблицю nodes. Це дозволяє описувати як спрямовані, так і неспрямовані відношення між елементами.

Спеціалізовані таблиці PersonDetails, HierarchyPersonDetails і TimelineItem пов'язані з вузлом у форматі один до одного. Це означає, що один вузол може мати додаткові дані відповідного типу, якщо вони потрібні для конкретної моделі. Наприклад, вузол генеалогічної моделі може мати дані про дату народження, стать і біографію, а вузол організаційної структури – посаду, відділ і контактні дані працівника.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

Для узагальнення кардинальності основних зв'язків між сутностями наведено таблицю 2.10.

Таблиця 2.10.

Основні зв'язки між сутностями

Зв'язок	Кардинальність	Опис
User – Model	1:N	Один користувач може створити багато моделей
Model – Node	1:N	Одна модель містить багато вузлів
Model – Relation	1:N	Одна модель містить багато зв'язків
Node – Relation	1:N	Один вузол може бути початковим або кінцевим елементом багатьох зв'язків
Node – NodeLayout	1:1	Один вузол має один набір layout-даних
Node – PersonDetails	1:0..1	Один вузол може мати персональні дані
Node – HierarchyPersonDetails	1:0..1	Один вузол може мати дані працівника
Node – TimelineItem	1:0..1	Один вузол може мати часові параметри
Model – ModelVersion	1:0..N	Одна модель може мати багато версій
Model – GraphShare	1:0..N	Одна модель може бути надана в доступ багатьом користувачам
Node – NodeAttributeValue	1:0..N	Один вузол може мати багато значень атрибутів
NodeAttributeDefinition – NodeAttributeValue	1:0..N	Одне визначення атрибута може використовуватися в багатьох значеннях

Структура бази даних відповідає принципам нормалізації. Основні об'єкти предметної області зберігаються в окремих таблицях, що дозволяє уникнути дублювання інформації та забезпечити узгодженість даних. Наприклад, модель не зберігає всі вузли в одному полі, а пов'язується з ними через окрему таблицю nodes. Аналогічно, зв'язки, атрибути, версії та правила доступу зберігаються у власних таблицях.

Для додаткового представлення структури предметної області доцільно використати UML-діаграму класів. Якщо ER-діаграма орієнтована на

відображення таблиць бази даних, зовнішніх ключів і кардинальності зв'язків, то діаграма класів дозволяє показати логічну модель системи з погляду об'єктів, їх атрибутів і взаємозалежностей. Це є важливим для розуміння того, як сутності бази даних відповідають об'єктам програмної системи.

Діаграма класів відображає основні сутності предметної області та зв'язки між ними. Центальною сутністю є Model, яка належить користувачу та містить вузли, зв'язки, налаштування, версії та правила спільного доступу. Вузли можуть мати layout-дані, спеціалізовані персональні, ієрархічні або часові властивості. Зв'язки поєднують два вузли та визначають тип відношення між ними (див. додат. А, рис. А.3).

Наведена діаграма класів демонструє, що структура даних системи побудована навколо моделі, яка об'єднує вузли та зв'язки. Окремі спеціалізовані сутності дозволяють підтримувати різні типи моделей без перевантаження базової сутності Node. Наприклад, персональні дані зберігаються окремо від загальних даних вузла, а часові параметри використовуються лише для елементів таймлайну. Такий підхід забезпечує гнучкість, нормалізацію структури даних і можливість подальшого розширення системи.

Використання діаграми класів разом з ER-діаграмою дозволяє розглянути структуру даних з двох різних позицій. ER-діаграма показує фізичну організацію таблиць у базі даних, тоді як діаграма класів відображає логічні сутності, з якими працює програмна система. У межах системи Graphori ці два представлення узгоджені між собою, оскільки основні класи предметної області відповідають ключовим таблицям бази даних.

Водночас у системі передбачено використання JSON-полів. Вони застосовуються там, де структура даних може змінюватися залежно від типу моделі або сценарію використання. Наприклад, JSON-поля використовуються для метаданих вузлів, налаштувань моделі, параметрів атрибутів і знімків версій. Це дозволяє поєднати переваги реляційної моделі з гнучкістю зберігання додаткових параметрів.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

З погляду цілісності даних важливо, щоб усі елементи моделі належали одній моделі. Зв'язок не повинен поєднувати вузли з різних моделей, а вузол не повинен існувати без відповідної моделі. Такі обмеження забезпечують коректність структури даних і запобігають виникненню неузгоджених зв'язків у графі. Крім того, для окремих типів моделей можуть застосовуватися додаткові правила. Наприклад, в організаційній ієрархії один працівник не повинен мати кількох безпосередніх керівників, а в генеалогічній моделі зв'язки мають відповідати родинній логіці.

Для підтримки операцій імпорту, експорту та відновлення моделей структура даних передбачає можливість збереження повного стану моделі. Під час експорту модель може бути представлена як сукупність вузлів, зв'язків, атрибутів, layout-даних і налаштувань. Під час імпорту ці дані відтворюються у відповідних таблицях. Для забезпечення узгодженості таких операцій доцільним є використання транзакцій, оскільки створення моделі, вузлів, зв'язків і атрибутів повинно виконуватися як єдина логічна операція.

Окремо слід зазначити підтримку м'якого видалення моделей. Для цього в сутності Model використовуються службові поля, які дозволяють позначити модель як видалену без негайного фізичного вилучення з бази даних. Це дає змогу реалізувати відновлення моделі та зменшує ризик втрати даних у разі випадкового видалення.

Таким чином, спроектована структура даних забезпечує зберігання складних інформаційних моделей, підтримку різних типів вузлів і зв'язків, розширення атрибутів, версіонування, спільний доступ, адміністративне керування та збереження візуальних параметрів. Вона є достатньо гнучкою для подальшого розвитку системи та водночас зберігає формалізовану реляційну основу, необхідну для забезпечення цілісності й надійності даних.

2.5. Програмна реалізація системи

Програмна реалізація системи Graphori виконана відповідно до архітектурних рішень, описаних у попередніх підрозділах. Основна увага приділялась побудові модульної структури, розділенню відповідальностей між

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

клієнтською, серверною та десктопною частинами, забезпеченню надійного зберігання даних, реалізації інтерактивного візуального редактора та підтримці кросплатформеної роботи.

Реалізація системи охоплює декілька взаємопов'язаних частин: серверний застосунок, клієнтський вебінтерфейс, підсистему візуального моделювання, механізми безпеки, систему зберігання даних, десктопний застосунок і засоби контейнеризації. Такий поділ дозволяє розглядати систему не як один монолітний програмний продукт, а як комплекс взаємодіючих модулів, кожен з яких виконує окрему функцію в загальній архітектурі.

У процесі реалізації важливо було забезпечити узгоджену взаємодію між усіма частинами системи. Клієнтський інтерфейс повинен коректно передавати дії користувача до серверної частини, серверна логіка – перевіряти права доступу, обробляти дані та зберігати їх у базі, а десктопна версія – підтримувати локальну роботу з моделями та подальшу синхронізацію з backend. Саме тому програмна реалізація побудована навколо чітко визначених інтерфейсів взаємодії, що зменшує залежність між модулями та спрощує подальше розширення функціональності.

Загальна відповідність частин програмної реалізації їх призначенню наведена в таблиці 2.11.

Таблиця 2.11.

Основні частини програмної реалізації системи

Частина системи	Основні елементи реалізації	Призначення
Серверна частина	NestJS-модулі, контролери, сервіси, guards, DTO, Prisma Service	Обробка запитів, бізнес-логіка, доступ до даних, автентифікація
Клієнтська частина	React-компоненти, сторінки, API-сервіси, контексти, хуки	Відображення інтерфейсу та взаємодія користувача із системою
Візуальний редактор	ModelEditorPage, React Flow, layout-модулі, панелі властивостей	Побудова, редагування та відображення інформаційних моделей

Підсистема безпеки	JWT, Google OAuth, RolesGuard, PermissionsGuard, OptionalJwtAuthGuard	Автентифікація, авторизація та контроль доступу
Система зберігання	PostgreSQL, Prisma ORM, транзакції, версії моделей, soft delete	Збереження користувачів, моделей, вузлів, зв'язків і службових даних
Десктопна частина	Electron, preload API, IPC, SQLite, Sync Service	Локальна робота з моделями та синхронізація з сервером
Інфраструктура	Docker, Docker Compose, окремі Dockerfile для backend і frontend	Контейнеризоване розгортання системи

Серверна частина реалізована як NestJS-застосунок із модульною структурою, склад якої було визначено під час проєктування програмної архітектури (див. розд. 2.2). У межах програмної реалізації основна увага приділяється не повторному переліку модулів, а принципу їх взаємодії під час виконання основних операцій системи.

У загальному випадку обробка запиту на серверному рівні відбувається за єдиною логічною схемою. Спочатку HTTP-запит надходить до відповідного контролера, де виконується маршрутизація та первинне приймання вхідних даних. Далі за потреби застосовуються guards, які перевіряють JWT-токен, роль користувача або наявність адміністративних дозволів. Після цього дані передаються до сервісного рівня, де виконується основна прикладна логіка: перевірка прав доступу до моделі, валідація даних, створення або оновлення сутностей, формування результату та підготовка відповіді клієнту. Усі операції з базою даних виконуються через PrismaService, що забезпечує централізований доступ до PostgreSQL

Для роботи з інформаційними моделями основну роль відіграють сервіси моделей, вузлів, зв'язків, атрибутів і спільного доступу. Вони взаємодіють між собою не як ізольовані частини, а як єдина прикладна підсистема. Наприклад, під час створення вузла сервіс вузлів звертається до сервісу моделей для перевірки

існування моделі та прав користувача на її редагування. Під час створення зв'язку додатково перевіряється належність обох вузлів одній моделі, допустимість типу зв'язку та відсутність конфліктів із правилами конкретного типу моделі. Такий підхід дозволяє зберігати цілісність даних і не допускати появи некоректних інформаційних структур.

Окрему роль у серверній реалізації відіграють операції, що потребують узгодженого створення або зміни кількох пов'язаних сутностей. До таких операцій належать імпорт моделі, повне видалення моделі, створення версії та відновлення стану моделі. Для них використовується транзакційний підхід, оскільки модель складається з багатьох взаємопов'язаних елементів: метаданих, вузлів, зв'язків, layout-даних, атрибутів і спеціалізованих властивостей. Якщо під час такої операції виникає помилка, транзакція дозволяє уникнути частково збереженого стану.

Одним із центральних серверних сервісів є ModelsService, який реалізує роботу з інформаційними моделями. Він відповідає за створення нової моделі, отримання списку моделей користувача, оновлення метаданих, зміну видимості, м'яке видалення, відновлення, повне видалення, імпорт, експорт і створення версій. Під час створення моделі формується базовий запис у таблиці models, після чого для окремих типів моделей можуть створюватися додаткові налаштування, наприклад параметри відображення або параметри часової лінії.

Для запобігання втраті даних у системі використовується механізм м'якого видалення. При видаленні модель не вилучається фізично з бази даних, а позначається як видалена за допомогою службових полів isDeleted і deletedAt. Це дозволяє реалізувати відновлення моделі, якщо користувач помилково її видалив. Повне видалення може виконуватися окремою операцією, під час якої пов'язані вузли, зв'язки, атрибути та інші залежні дані видаляються узгоджено.

NodesService реалізує логіку роботи з вузлами. Під час створення вузла перевіряється існування моделі, право користувача на її редагування та коректність типу вузла для відповідного типу моделі. Наприклад, у генеалогічних моделях вузли можуть мати персональні дані, в організаційній

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

ієрархії – дані працівника, а в таймлайні – часові параметри. Завдяки такій структурі базова сутність Node залишається універсальною, а специфічні дані зберігаються в окремих таблицях.

RelationsService відповідає за створення та перевірку зв'язків між вузлами. Перед створенням зв'язку перевіряється, чи належать обидва вузли одній моделі, чи не створюється зв'язок вузла із самим собою, чи не існує дубліката такого зв'язку, а також чи відповідає тип зв'язку правилам конкретної моделі. Для організаційних ієрархій додатково важливо запобігати циклічним залежностям і ситуаціям, коли один працівник має кількох безпосередніх керівників.

Послідовність обробки операції створення вузла на серверному рівні подано в додатку А рис. 1.

Наведена діаграма демонструє, що операція створення вузла складається з декількох етапів: отримання дії користувача, перевірки автентифікації, перевірки доступу до моделі, валідації даних, збереження у базі даних і повідомлення клієнта про зміну стану. Така послідовність забезпечує не лише створення графічного елемента, а й підтримку цілісності даних та актуального стану редактора.

Для кращого відображення графів було реалізовано автоматичне вирівнювання положень вершин графу відповідно до типу графу. Кожний тип графу має особливе відображення та спосіб візуального зв'язку між об'єктами. Для автоматичного визначення положень вершин та їх оптимальне розміщення у інтерфейсі системи, було реалізовано метод «solve» у класі GraphLayoutService (див. додат. В).

Для операцій імпорту, експорту та версіонування використовується узгоджене представлення моделі як сукупності метаданих, вузлів, зв'язків, атрибутів, layout-даних і спеціалізованих властивостей. Під час експорту система формує структурований об'єкт, який може бути збережений у файл. Під час імпорту цей об'єкт відтворюється у відповідних таблицях бази даних.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

Оскільки імпорт моделі передбачає створення великої кількості взаємопов'язаних записів, ця операція виконується як транзакція. Це дозволяє уникнути частково створених моделей у випадку помилки.

Окремим важливим механізмом системи є версіонування моделей. Система дозволяє не лише зберігати поточний стан моделі, а й відновлювати попередні версії. При цьому перед відновленням створюється резервна копія поточного стану, що зменшує ризик втрати даних. Відповідний механізм реалізовано в методі `restoreVersion` сервісу `ModelsService`. Для кращого пояснення логіки роботи методу доцільно розглянути його не як один великий фрагмент коду, а як послідовність окремих етапів.

Першим етапом є перевірка права користувача на редагування моделі та пошук версії, яку потрібно відновити. Це необхідно для того, щоб запобігти відновленню версій користувачами, які не мають доступу до відповідної моделі, а також унеможливити роботу з неіснуючою або сторонньою версією.

```
async restoreVersion(modelId: string, versionId: string, userId: string) {  
  const model = await this.assertCanWrite(modelId, userId);  
  const version = await this.prisma.modelVersion.findFirst({  
    where: { id: versionId, modelId },  
  });  
  
  if (!version) {  
    throw new NotFoundException(`Version with id ${versionId} was not found`);  
  }  
}
```

У наведеному фрагменті метод `assertCanWrite` перевіряє, чи має користувач право змінювати модель. Така перевірка є важливою частиною механізму контролю доступу, оскільки відновлення версії фактично змінює поточний стан моделі. Після цього виконується пошук запису версії за ідентифікатором моделі та ідентифікатором версії. Якщо відповідний запис не знайдено, система генерує виняток `NotFoundException`, і операція відновлення припиняється.

Наступним етапом є перевірка збереженого стану моделі. Оскільки версія зберігається у вигляді `snapshot`, перед відновленням необхідно переконатися, що

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

його структура є коректною, відповідає підтримуваний схемі та належить до того самого типу моделі.

```
const snapshot = version.snapshot as Record<string, unknown>;
const schemaVersion = this.getString(
  snapshot.schemaVersion ?? snapshot.version,
);

if (schemaVersion !== '1.0') {
  throw new BadRequestException('Unsupported model version schema');
}

const normalized = this.normalizeImport({
  format: ModelImportFormat.JSON,
  content: JSON.stringify(snapshot),
  fallbackModelType: model.type,
});

if (!normalized.valid || !normalized.normalized) {
  throw new BadRequestException({
    message: 'Version snapshot is not valid',
    errors: normalized.errors,
    warnings: normalized.warnings,
  });
}

if (normalized.normalized.model.type !== (model.type as ModelType)) {
  throw new BadRequestException(
    'Cannot restore a version with a different model type',
  );
}
```

Цей фрагмент демонструє перевірку сумісності збереженої версії перед її застосуванням. Спочатку визначається версія схеми snapshot, після чого система перевіряє, чи підтримується вона поточною реалізацією. Далі виконується нормалізація імпортованої структури, що дозволяє привести дані до внутрішнього формату системи. Якщо snapshot має некоректну структуру, містить несумісний тип моделі або не може бути нормалізований, відновлення не виконується. Такий підхід запобігає пошкодженню поточної моделі через застосування неправильних або несумісних даних.

Після успішної перевірки snapshot система формує резервну копію поточного стану моделі. Це необхідно для того, щоб користувач мав можливість повернутися до стану, який існував безпосередньо перед відновленням старішої версії.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

```
const imported = normalized.normalized;

const backupSnapshot = await this.getExportSnapshot(modelId, {
  includeLayout: true,
  includeAttributes: true,
  includeDisplaySettings: true,
  includeTimelineSettings: true,
});
```

У цьому фрагменті поточний стан моделі зчитується як експортований snapshot. До резервної копії включаються не лише вузли та зв'язки, а й параметри розміщення, користувацькі атрибути, налаштування відображення та специфічні параметри окремих типів моделей. Завдяки цьому резервна версія зберігає повний стан моделі, а не лише її базову структуру.

Завершальним етапом є транзакційне відновлення моделі. У межах однієї транзакції система створює резервну версію поточного стану, після чого замінює дані моделі даними з обраного snapshot.

```
await this.prisma.$transaction(async (tx) => {
  await this.createVersionFromSnapshot(
    modelId,
    userId,
    backupSnapshot,
    {
      name: `Backup before restore v${version.versionNumber}`,
      description:
        'Automatically created before restoring a previous version.',
    },
    tx,
  );

  await this.replaceModelWithImportedSnapshot(
    tx,
    modelId,
    userId,
    imported,
  );
});

return this.findById(modelId);
}
```

Наведений фрагмент демонструє використання транзакції для операції, яка змінює значну кількість взаємопов'язаних даних. Спочатку створюється резервна копія поточного стану моделі, після чого виконується заміна даних на основі обраної версії. Якщо під час виконання будь-якого з цих етапів виникає помилка, транзакція відкочується, а модель залишається у попередньому

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

узгодженому стані. Це дозволяє уникнути часткового відновлення, втрати вузлів, зв'язків або атрибутів.

Таким чином, механізм відновлення версії складається з декількох послідовних етапів: перевірки прав доступу, пошуку версії, валідації snapshot, створення резервної копії та транзакційної заміни поточного стану моделі. Такий підхід підвищує надійність роботи з даними та є доцільним для системи, у якій модель складається з багатьох взаємопов'язаних сутностей.

Окремим елементом реалізації є система атрибутів вузлів. Вона дозволяє створювати додаткові поля без зміни основної таблиці nodes. Для цього використовується пара сутностей NodeAttributeDefinition і NodeAttributeValue. Перша описує атрибут, його тип і область застосування, а друга зберігає конкретне значення для окремого вузла. Такий підхід дозволяє адаптувати систему до різних предметних областей і створювати розширювані моделі.

Ще одним прикладом практичної реалізації є нормалізація значень користувацьких атрибутів вузлів. Оскільки система підтримує різні типи атрибутів, зокрема текст, число, телефон, email, URL, дату, логічне значення, список і множинний список, серверна частина повинна перевіряти коректність значення перед збереженням у базу даних (див додат. Г).

У цьому фрагменті показано, що система не зберігає довільні неструктуровані значення без перевірки. Для кожного типу поля застосовується власна логіка валідації та приведення до відповідного формату. Наприклад, email перевіряється за шаблоном, URL нормалізується через стандартний об'єкт URL, числове значення перетворюється на число з перевіркою коректності, а для множинного списку перевіряється, що всі значення належать до дозволених варіантів. Це підвищує якість даних і зменшує ризик появи некоректних значень у базі.

Основним компонентом клієнтської частини є редактор моделей. Він відповідає за завантаження моделі, перетворення серверних даних у формат, придатний для візуального відображення, обробку дій користувача, створення

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

вузлів і зв'язків, відкриття панелі властивостей, зміну координат елементів і синхронізацію змін із сервером.

Редактор підтримує різні типи моделей, тому його внутрішня логіка побудована так, щоб загальні механізми роботи з вузлами та зв'язками могли використовуватися для різних сценаріїв (див. табл. 2.12).

Таблиця 2.12.

Основні частини клієнтського редактора моделей

Частина редактора	Призначення
ModelEditorPage	Центральна сторінка редагування моделі та координації стану редактора
Компоненти полотна	Відображення вузлів, зв'язків, таймлайнів і спеціалізованих структур
Панель властивостей	Редагування параметрів вибраного вузла або зв'язку
Модулі layout	Обчислення або збереження координат елементів моделі
API-сервіси	Виклик backend-операцій для моделей, вузлів, зв'язків і атрибутів
WebSocket-клієнт	Отримання подій оновлення моделі в реальному часі
Утиліти перетворення даних	Перетворення серверних сутностей у формат візуального редактора

Реалізація візуального моделювання базується на універсальному підході до вузлів і зв'язків. Для мережевих моделей основну роль відіграє довільне розташування вузлів на полотні та візуальне відображення зв'язків між ними. Для ієрархічних моделей важливим є розташування елементів за рівнями підпорядкування. Для генеалогічних моделей враховуються зв'язки між батьками, дітьми та партнерами. Для таймлайнів ключовим є коректне відображення подій, періодів і часових проміжків на горизонтальній осі.

Підтримка різних типів моделей реалізується не через створення повністю незалежних редакторів, а через спеціалізацію спільного редакторського механізму. Базові операції над вузлами, зв'язками, збереженням і панеллю властивостей залишаються спільними, тоді як спосіб відображення та набір специфічних полів залежать від типу моделі. Це дозволяє уникнути дублювання логіки й одночасно забезпечити різні сценарії візуалізації.

У таблиці 2.13 наведено відповідність типів моделей особливостям їх реалізації.

Таблиця 2.13.

Реалізація різних типів візуальних моделей

Тип моделі	Особливості реалізації
Мережева модель	Довільне розміщення об'єктів, зв'язки між вузлами, підтримка графових структур
Генеалогічна модель	Вузли-особи, родинні зв'язки, партнери, батьки, діти, персональні дані
Організаційна ієрархія	Підпорядкування, посади, відділи, прямі керівники, фільтрація працівників
Таймлайн	Події, періоди, часові проміжки, масштабування часової осі
Ієрархічна структура	Батьківські та дочірні вузли, рівневе розташування елементів
Модель залежностей	Напрявлені зв'язки, залежності між елементами, перевірка циклів за потреби

Для підтримки взаємодії в реальному часі реалізовано WebSocket-рівень. Основні операції створення та збереження даних виконуються через HTTP-запити, а WebSocket використовується для повідомлення клієнтів про зміни. Такий підхід дозволяє уникнути періодичних запитів до сервера та забезпечити актуалізацію стану редактора. Нижче наведено приклад програмної реалізації перевірки конфлікту паралельного редагування.

```
private assertVersion(
    entityType: string,
    entityId: string,
    updatedAt: Date,
    expectedUpdatedAt: string,
    serverValue: unknown,
) {
    if (updatedAt.toISOString() === new Date(expectedUpdatedAt).toISOString()) {
        return;
    }

    throw new ConflictException({
        entityType,
        entityId,
        reason: 'ENTITY_WAS_MODIFIED',
        serverValue,
    });
}
```

```
});
}
```

Метод `assertVersion` порівнює актуальне значення часу оновлення сутності на сервері з тим значенням, яке було відоме клієнту під час виконання дії. Якщо ці значення не збігаються, система визначає, що сутність уже була змінена іншим користувачем, і повертає помилку конфлікту. Це запобігає ситуації, коли один користувач непомітно перезаписує зміни іншого.

```
private async handleCommand(
  client: AuthenticatedSocket,
  modelId: string,
  clientMutationId: string | undefined,
  command: () => Promise<void>,
) {
  if (!client.rooms.has(modelRoom(modelId))) {
    this.emitError(
      client,
      new BadRequestException('Join the model room before editing.'),
      clientMutationId,
    );
    return;
  }

  try {
    await command();
  } catch (error) {
    this.emitError(client, error, clientMutationId);
  }
}

private broadcastConfirmed<TPayload>(
  client: AuthenticatedSocket,
  eventName: string,
  modelId: string,
  clientMutationId: string | undefined,
  payload: TPayload,
) {
  const event = {
    ...this.baseEvent(modelId, clientMutationId, client),
    payload,
  };

  this.server.to(modelRoom(modelId)).emit(eventName, event);
}
```

Метод `handleCommand` перевіряє, чи користувач приєднаний до кімнати відповідної моделі, і лише після цього дозволяє виконання команди редагування. Якщо користувач не приєднався до кімнати моделі, система повертає помилку. Метод `broadcastConfirmed` використовується для поширення підтвердженої

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

зміни всім клієнтам, які працюють із цією моделлю. Завдяки цьому всі відкриті редактори отримують актуальний стан без необхідності постійно виконувати повторні HTTP-запити.

Механізми безпеки реалізовано на рівні автентифікації, авторизації та перевірки прав доступу до моделей. Після входу користувач отримує JWT-токен, який передається у подальших запитах до backend. JwtStrategy перевіряє токен і отримує дані користувача, а JwtAuthGuard захищає маршрути, доступні лише авторизованим користувачам. Для маршрутів, які працюють з авторизованим та неавторизованим користувачем, використовується OptionalJwtAuthGuard.

Рольова модель доступу включає звичайного користувача, адміністратора та суперадміністратора. Для перевірки ролей використовується RolesGuard, а для точнішого контролю адміністративних дій – PermissionsGuard. Адміністративні дозволи зберігаються в базі даних, що дозволяє гнучко визначати, які саме дії може виконувати конкретний адміністратор.

У таблиці 2.14 наведено основні механізми безпеки системи.

Таблиця 2.14.

Реалізовані механізми безпеки

Механізм	Призначення
JWT-автентифікація	Підтвердження особи користувача під час запитів до API
Хешування паролів	Захист паролів користувачів у базі даних
Google OAuth	Альтернативний спосіб входу через зовнішнього провайдера
JwtAuthGuard	Захист приватних маршрутів
OptionalJwtAuthGuard	Робота маршрутів із необов'язковою автентифікацією
RolesGuard	Перевірка ролі користувача
PermissionsGuard	Перевірка адміністративних дозволів
Graph visibility	Розмежування приватних, публічних і спільних моделей
Graph moderation	Модерація публічних моделей адміністраторами

Десктопна версія системи реалізована на основі Electron. Вона використовує React-інтерфейс, але має додаткову системну частину, яка відповідає за створення вікна застосунку, локальне сховище, роботу з токенами, IPC-виклики та синхронізацію з backend. Такий підхід дозволяє повторно використовувати клієнтський інтерфейс і водночас забезпечити можливості, характерні для настільних застосунків.

У десктопній частині використовується розділення на renderer process, main process і preload API. Renderer process відповідає за інтерфейс користувача. Main process керує вікном застосунку, локальною базою, токенами та системними ресурсами. Preload API виступає контрольованим посередником між інтерфейсом і системними можливостями Electron. Це дозволяє не надавати інтерфейсу прямий доступ до Node.js API.

Локальне збереження моделей у десктопній версії виконується за допомогою SQLite. Користувач може створювати або редагувати модель локально, після чого за потреби виконувати синхронізацію з сервером. Синхронізація передбачає порівняння локального та хмарного стану моделі, передавання змін і обробку можливих конфліктів. Послідовність локального збереження та синхронізації моделі наведено на рисунку 2.8.

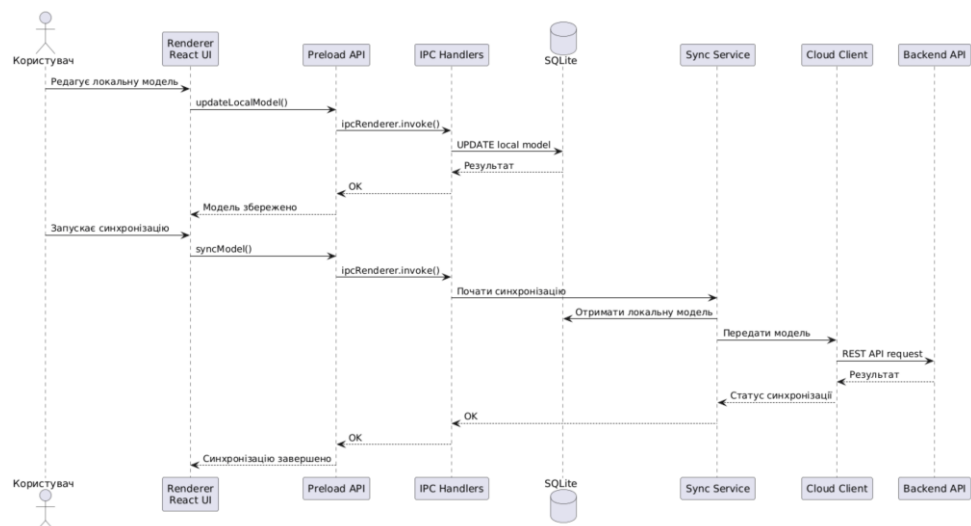


Рис. 2.8. Послідовність локального збереження та синхронізації моделі

Дана послідовність демонструє, що десктопний застосунок не звертається безпосередньо з інтерфейсу до локальної бази даних. Усі операції проходять через preload API та IPC handlers. Це забезпечує контрольований доступ до локального сховища й дозволяє відокремити візуальний рівень від системної логіки. Під час синхронізації локальна модель передається до cloud-клієнта, який взаємодіє із серверною частиною через REST API.

Для спрощення розгортання система контейнеризується за допомогою Docker. Окремі контейнери використовуються для PostgreSQL, backend API та frontend-застосунку. Docker Compose описує їх взаємодію, змінні середовища, порти, volume для збереження даних і залежності між сервісами. Це дозволяє запускати систему в однаковому середовищі на різних машинах і спрощує подальше розгортання. Інфраструктурна частина реалізації наведена в таблиці 2.15.

Таблиця 2.15.

Інфраструктурні елементи реалізації

Елемент	Призначення
Dockerfile backend	Збірка та запуск серверної частини
Dockerfile frontend	Збірка та запуск клієнтської частини
Docker Compose	Одночасний запуск backend, frontend і PostgreSQL

продовж. табл. 2.15

PostgreSQL volume	Постійне збереження даних бази між перезапусками контейнера
Environment variables	Передавання конфігурації без жорсткого закріплення параметрів у коді
Healthcheck PostgreSQL	Перевірка готовності бази перед запуском backend

З погляду програмної реалізації важливим є те, що основна логіка системи не залежить від конкретного способу запуску. Backend може працювати локально, у контейнері або на сервері. Frontend може запускатися у режимі розробки або збиратися як статичний застосунок. Desktop-застосунок може взаємодіяти з тим самим backend API, що й вебклієнт. Це забезпечує

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Левківський В.Л.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

узгодженість поведінки системи у різних середовищах. Desktopна частина додає локальне збереження та синхронізацію. Усі ці частини разом формують повноцінну кросплатформену систему візуального моделювання інформації.

Таким чином, програмна реалізація системи Graphori охоплює серверну логіку, клієнтський інтерфейс, інтерактивний редактор моделей, механізми безпеки, систему зберігання даних, десктопну версію та контейнеризоване середовище запуску. Реалізована структура відповідає спроектованій архітектурі та забезпечує основу для подальшої демонстрації, перевірки роботи й тестування системи.

Висновки до другого розділу.

Було виконано проектування архітектури інтерактивної кросплатформеної системи для візуального моделювання інформації. Визначено загальну клієнт-серверну структуру системи, описано її основні логічні рівні, компоненти, потоки даних та взаємодію між вебклієнтом, десктопним застосунком, серверною частиною, централізованою базою даних і локальним сховищем. Запропонована архітектура забезпечує підтримку роботи з різними типами інформаційних моделей.

У межах проектування програмної архітектури було визначено внутрішню організацію серверної, клієнтської та десктопної частин системи. Серверна частина побудована за модульним принципом і включає підсистеми автентифікації, керування користувачами, моделями, вузлами, зв'язками, атрибутами, спільним доступом, адмініструванням та взаємодією в реальному часі. Клієнтська частина організована як компонентний застосунок із візуальним редактором моделей, API-сервісами, модулем авторизації та локалізації. Десктопна частина передбачає локальне збереження даних, взаємодію через preload API та IPC-механізм, а також синхронізацію з сервером.

Також у розділі було спроектовано структуру даних системи. Визначено основні сутності предметної області, зокрема користувачів, моделі, вузли,

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

зв'язки, атрибути, версії моделей, налаштування відображення, правила спільного доступу та адміністративні дозволи.

Окрему увагу приділено програмній реалізації системи. Було описано принципи взаємодії контролерів, сервісів, guards, Prisma Service, frontend-компонентів, візуального редактора, WebSocket-рівня, механізмів безпеки, десктопного застосунку та інфраструктурних засобів розгортання. Це дозволило показати, як спроектована архітектура переходить у практичну реалізацію програмного комплексу.

У підрозділі визначення варіантів використання було формалізовано основні сценарії взаємодії користувачів із системою. Розглянуто ролі гостя, авторизованого користувача, користувача десктопної версії, адміністратора та суперадміністратора, а також визначено доступні для них дії. Побудована діаграма варіантів використання та матриця доступу стали основою для подальшої демонстрації роботи системи й формування тестових сценаріїв у третьому розділі.

Таким чином, у другому розділі було сформовано повну проєктну та реалізаційну основу системи Graphori. Описані архітектурні рішення, структура даних, програмні компоненти та варіанти використання створюють цілісне уявлення про побудову системи та забезпечують перехід до демонстрації її роботи, налаштування, експлуатації й тестування.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1. Структура та основні елементи інтерфейсу системи.

Інтерфейс системи Graphori спроектований таким чином, щоб забезпечити зручну роботу користувача з різними типами інформаційних моделей. Основну увагу приділено простоті навігації, зрозумілому розміщенню елементів керування, доступності основних функцій і можливості швидкого переходу від списку моделей до їх редагування. Оскільки система підтримує декілька типів структур, інтерфейс повинен бути універсальним, але водночас адаптованим до особливостей конкретної моделі.

До основних елементів інтерфейсу системи належать головна сторінка, форма створення моделі, редактор моделей, панель властивостей, сторінка профілю користувача та адміністративна панель. Кожен із цих елементів виконує окрему функцію в загальному сценарії роботи користувача із системою. Головна сторінка забезпечує доступ до створених моделей, редактор відповідає за побудову та зміну структури моделі, панель властивостей дозволяє редагувати дані окремих елементів, а адміністративна панель використовується для керування користувачами та публічними моделями.

Загальну структуру інтерфейсу системи наведено в таблиці 3.1.

Таблиця 3.1.

Основні елементи інтерфейсу системи

Елемент інтерфейсу	Призначення	Основні можливості
Головна сторінка	Відображення доступних моделей користувача	Перегляд списку моделей, створення нової моделі, відкриття моделі для редагування
Форма створення моделі	Створення нової інформаційної структури	Введення назви, опису, вибір типу моделі та рівня доступу
Редактор моделей	Основна робоча область системи	Додавання вузлів, створення зв'язків, переміщення елементів, масштабування, редагування структури

Панель властивостей	Редагування параметрів вибраного елемента	Зміна назви, типу, атрибутів, персональних, часових або ієрархічних даних
Сторінка профілю	Керування персональними даними користувача	Оновлення імені, контактних даних, фото профілю, мови інтерфейсу
Адміністративна панель	Керування системою адміністраторами	Перегляд користувачів, модерація публічних моделей, керування адміністраторами та дозволами

Після входу до системи користувач потрапляє на головну сторінку, де відображається перелік доступних моделей. Кожна модель представлена у вигляді окремої картки, що містить назву, тип моделі, короткий опис і службову інформацію. Такий формат дозволяє швидко орієнтуватися серед створених структур і відкривати потрібну модель для подальшого редагування. На головній сторінці також розміщено елемент створення нової моделі, що є початковою точкою для більшості сценаріїв роботи користувача.

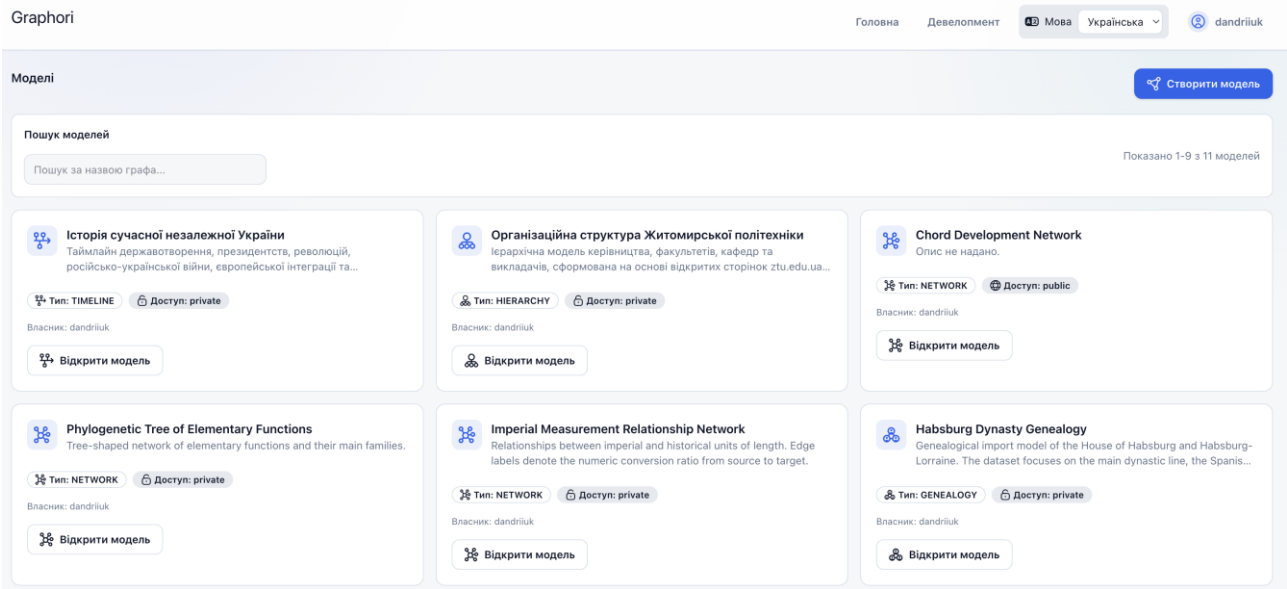


Рис. 3.1. Головна сторінка системи зі списком моделей

Форма створення моделі використовується для визначення початкових параметрів нової інформаційної структури. Користувач задає назву моделі, за

потреби додає опис, обирає тип моделі та визначає параметри доступу. Вибір типу моделі є важливим, оскільки від нього залежить подальша логіка роботи редактора, набір доступних елементів і спосіб візуалізації даних. Наприклад, для таймлайну основним елементом є подія або часовий проміжок, для генеалогічної структури – особа, а для організаційної ієрархії – працівник або посада.

Основним елементом системи є редактор моделей. Він містить робочу область, на якій відображаються вузли, зв'язки та інші візуальні елементи моделі. У редакторі користувач може додавати нові елементи, переміщувати їх, створювати зв'язки, масштабувати робочу область, змінювати положення елементів і відкривати панель властивостей для детального редагування. Інтерфейс редактора побудований таким чином, щоб користувач міг працювати з моделлю без необхідності переходити між великою кількістю сторінок.

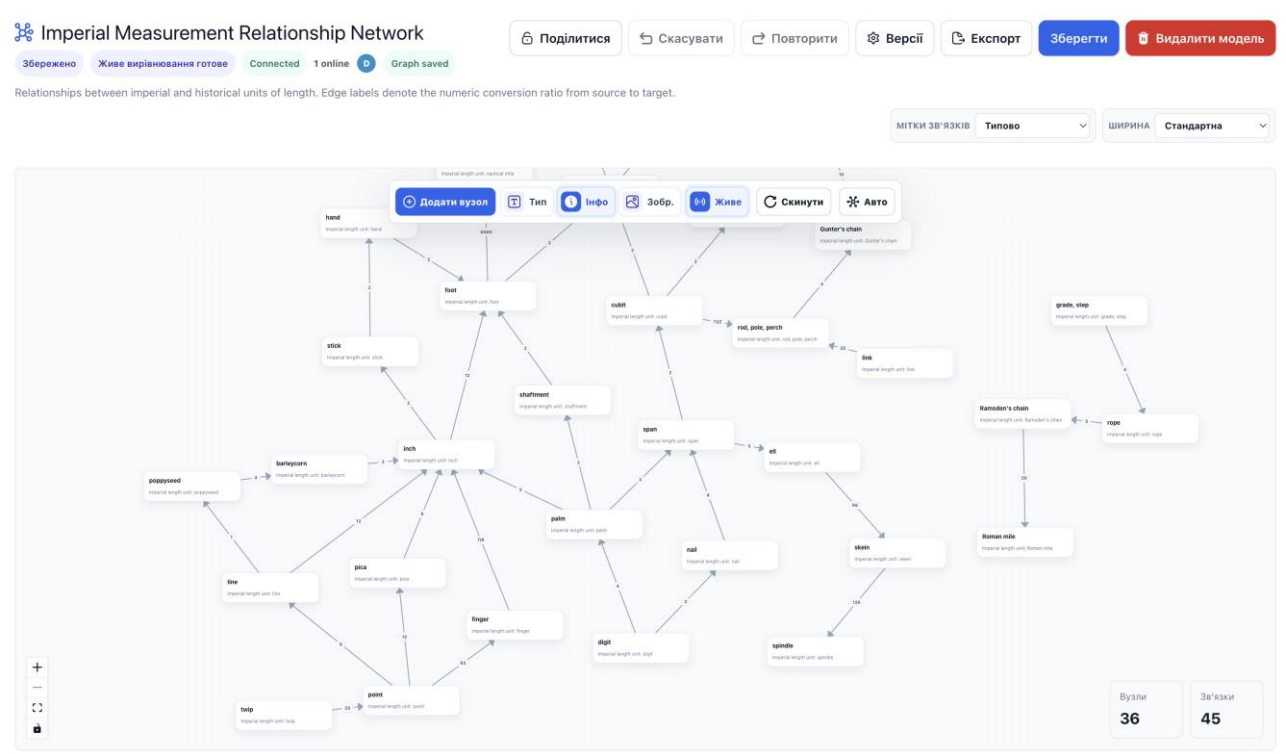


Рис. 3.2. Інтерфейс редактора мережевої моделі

Редактор моделей є універсальною робочою областю, однак його відображення адаптується до типу створеної інформаційної структури. Для мережевих моделей основну роль відіграють вузли та довільні зв'язки між ними,

для генеалогічних моделей – особи та родинні відношення, для організаційних ієрархій – рівневе підпорядкування працівників або посад, а для часових моделей – події та періоди, розміщені на часовій осі. У цьому підрозділі ці режими розглядаються лише як складові інтерфейсу, тоді як порядок практичної роботи з ними подано в підрозділі 3.2.

Панель властивостей є важливою частиною редактора моделей. Вона відкривається після вибору вузла або зв'язку та дозволяє редагувати детальні параметри елемента.

Для звичайного вузла це можуть бути назва, опис, тип і додаткові атрибути. Для особи в генеалогічній моделі – ім'я, прізвище, дата народження, стать, біографія та інші персональні дані. Для елемента таймлайну – дата події, початок і кінець періоду, тип часової сутності та колір.

Такий підхід дозволяє використовувати один загальний принцип редагування для різних типів моделей.

Рис. 3.3. Панель властивостей вибраного елемента моделі

Окремим елементом інтерфейсу є сторінка профілю користувача. Вона призначена для перегляду та редагування персональних даних, зокрема імені, прізвища, контактної інформації, фото профілю та мови інтерфейсу. Наявність сторінки профілю дозволяє персоналізувати роботу із системою та зберігати індивідуальні налаштування користувача.

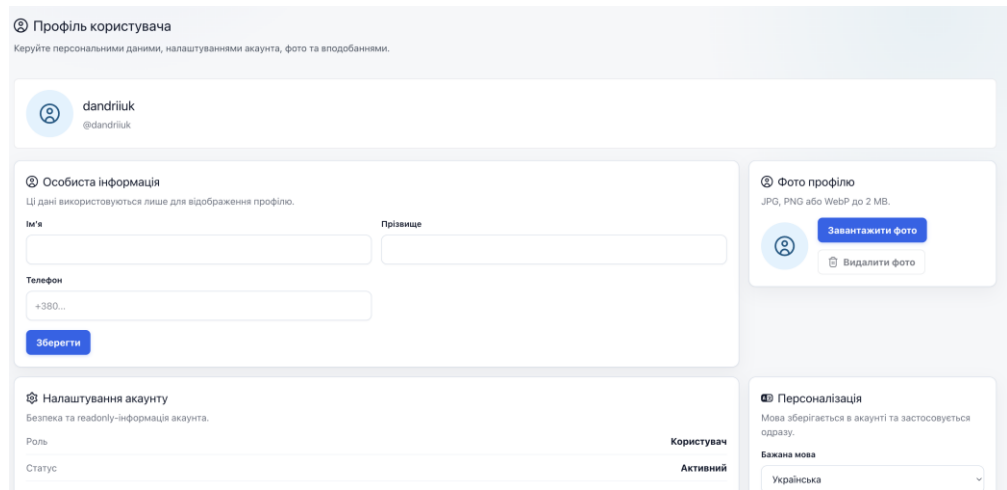


Рис. 3.4. Сторінка профілю користувача

Адміністративна панель доступна користувачам із відповідними правами. Вона містить інструменти для перегляду користувачів, керування їх статусом, модерації публічних моделей, перегляду адміністраторів і налаштування дозволів. Завдяки цьому система підтримує не лише індивідуальну роботу з моделями, а й адміністративне керування публічним контентом і користувацькими обліковими записами.

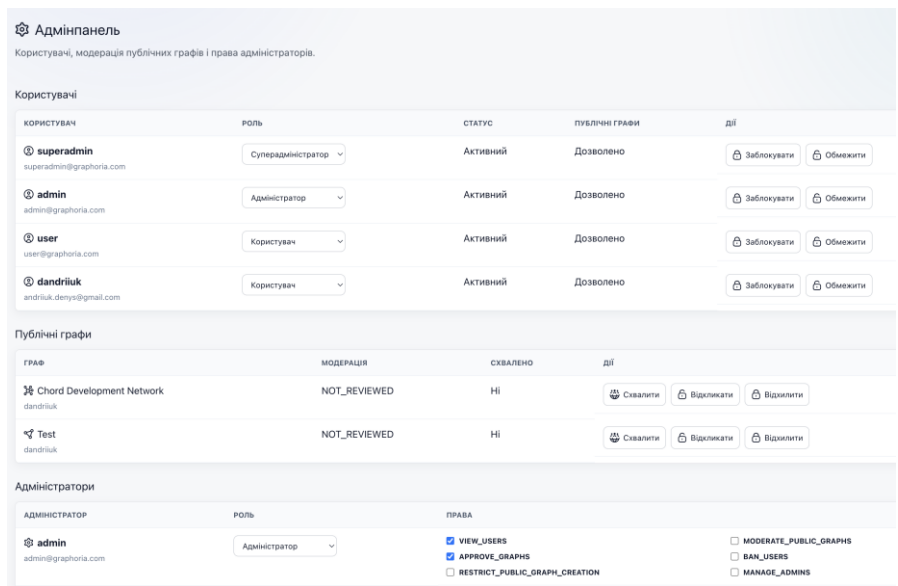


Рис. 3.5. Адміністративна панель системи

Важливою характеристикою інтерфейсу є підтримка локалізації. Система передбачає можливість використання української та англійської мов. Це дозволяє адаптувати інтерфейс до потреб різних користувачів і підвищує зручність роботи з програмним продуктом. Мова інтерфейсу може бути змінена користувачем і збережена в його налаштуваннях.

Інтерфейс системи також враховує потребу в розмежуванні доступу. Залежно від ролі користувача та типу моделі в інтерфейсі можуть відображатися різні елементи керування. Наприклад, гість має обмежений доступ і може переглядати лише публічні моделі, звичайний користувач може створювати та редагувати власні моделі, а адміністратор отримує доступ до адміністративної панелі. Такий підхід дозволяє уникнути відображення недоступних функцій і робить інтерфейс більш зрозумілим.

Таким чином, інтерфейс системи Graphorі складається з набору взаємопов'язаних екранів і компонентів, які забезпечують повний цикл роботи з інформаційними моделями: від створення та редагування до керування доступом і адміністрування. Наявність спеціалізованих режимів для різних типів моделей дозволяє використовувати систему в різних предметних областях, а підтримка профілю, локалізації та адміністративної панелі робить програмний продукт придатним для практичного використання.

3.2. Порядок роботи користувача із системою.

Порядок роботи користувача із системою Graphorі розглянемо на прикладі типового сценарію створення та редагування інформаційної моделі. Такий сценарій демонструє основні можливості програмного продукту: вхід до системи, створення моделі, вибір її типу, роботу з візуальним редактором, додавання вузлів, створення зв'язків, редагування властивостей елементів, збереження результату та подальше використання моделі.

Після запуску системи користувач переходить на головну сторінку, де відображається список доступних моделей. На цій сторінці користувач може переглянути вже створені моделі, відкрити одну з них для редагування або створити нову. Кожна модель подається у вигляді картки, що містить її назву,

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

тип і короткий опис. Такий підхід спрощує навігацію між різними інформаційними структурами.

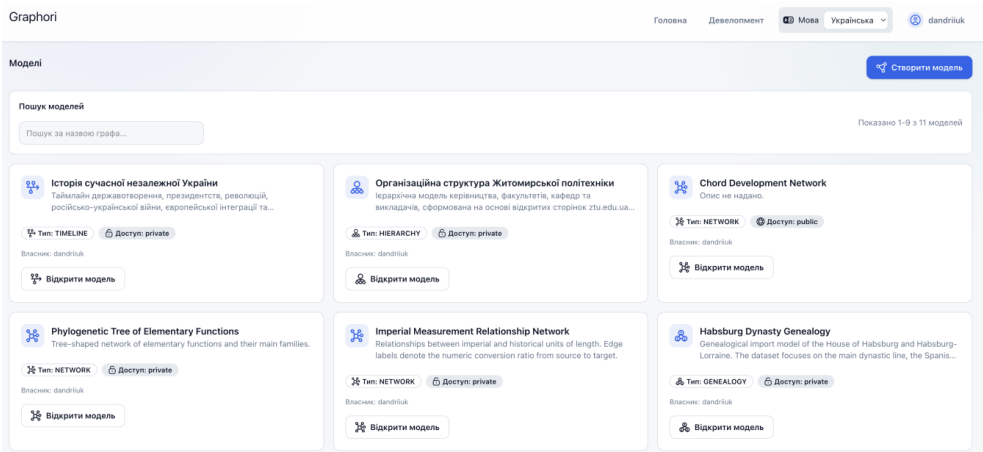


Рис. 3.6. Головна сторінка системи зі списком доступних моделей

Для створення нової моделі користувач натискає кнопку створення моделі. Після цього відкривається форма, у якій потрібно вказати назву, опис і тип моделі. На цьому етапі користувач визначає, з якою структурою він буде працювати: мережею об’єктів, генеалогічною моделлю, організаційною ієрархією, таймлайном або іншим типом інформаційної структури. Вибір типу моделі впливає на подальший вигляд редактора та доступні властивості елементів.

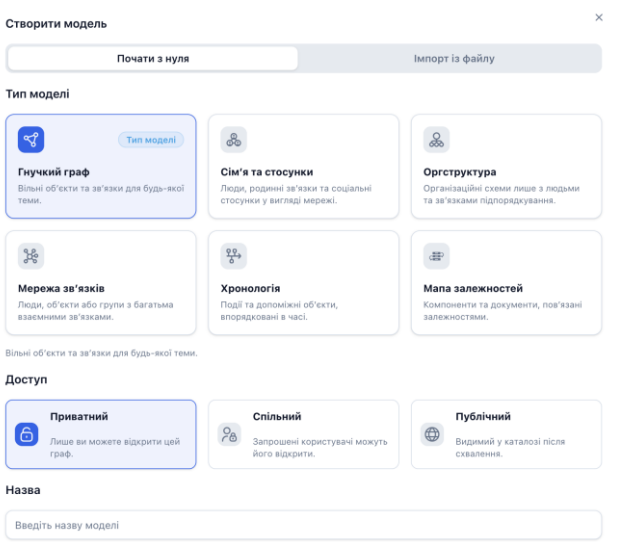


Рис. 3.7. Форма створення нової інформаційної моделі

Після створення модель відкривається у редакторі. Редактор є основним робочим середовищем користувача. У центральній частині розміщується

полотно, на якому відображаються вузли та зв'язки. У верхній або бічній частині інтерфейсу розміщуються елементи керування, що дозволяють додавати нові вузли, змінювати масштаб, виконувати пошук, переміщуватися по полотну та відкривати додаткові налаштування.

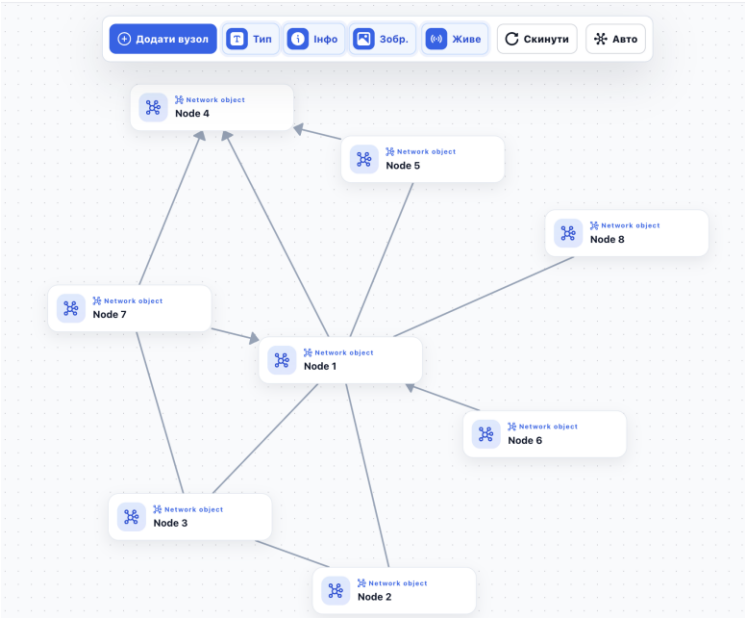


Рис. 3.8. Відкрита мережева модель у редакторі

На прикладі мережевої моделі користувач може додати декілька об'єктів і встановити між ними зв'язки. Для цього спочатку створюються вузли, які представляють окремі сутності предметної області. Після додавання вузлів користувач може змінювати їх положення на полотні, формуючи зручне візуальне розташування. Далі між вузлами створюються зв'язки, які відображають логічні або смислові відношення між об'єктами.

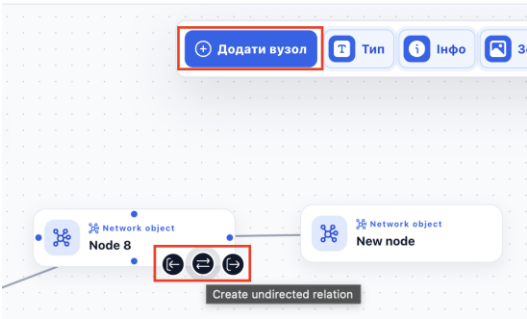


Рис. 3.9. Додавання вузлів і зв'язків у мережевій моделі

Після вибору певного елемента відкривається панель властивостей. Вона дозволяє змінювати назву вузла, опис, тип, додаткові атрибути або спеціалізовані параметри. Для різних типів моделей набір властивостей

відрізняється. Наприклад, для звичайного вузла доступні базові атрибути, для особи в генеалогічній моделі – персональні дані, а для елемента таймлайну – дата, часовий проміжок або період.

Рис. 3.10. Панель властивостей вибраного елемента моделі

Для генеалогічної моделі порядок роботи користувача має іншу предметну логіку. У цьому випадку вузли представляють окремих осіб, а зв’язки між ними відображають родинні відношення. Користувач додає осіб, заповнює їх персональні дані, після чого встановлює зв’язки партнерства, батьківства або спорідненості. У результаті система формує візуальне родинне дерево.

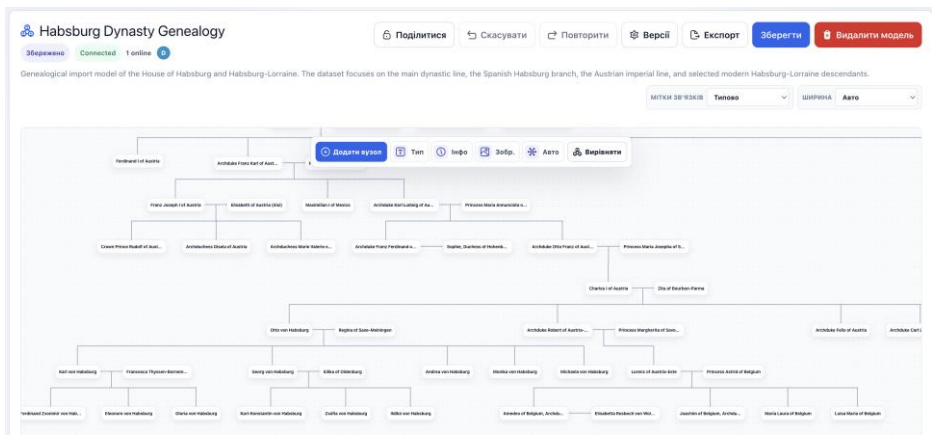


Рис. 3.11. Приклад генеалогічної моделі в редакторі

Під час роботи з генеалогічною моделлю користувач може вибрати конкретну особу та відредагувати її дані у панелі властивостей. Наприклад, можна вказати ім’я, прізвище, дату народження, стать, біографічні відомості та

інші персональні характеристики. Таким чином, модель поєднує візуальну структуру родинних зв'язків і детальну інформацію про кожного учасника дерева.

Рис. 3.12. Редагування персональних даних особи у генеалогічній моделі

Для організаційної ієрархії користувач працює зі структурою підпорядкування. У такій моделі вузли представляють працівників або посади, а зв'язки показують відношення керівник–підлеглий. Користувач може додавати працівників, визначати їх посади, відділи, контактні дані та встановлювати ієрархічні зв'язки між ними. Для зручності роботи з великими структурами можуть використовуватися пошук і фільтрація.

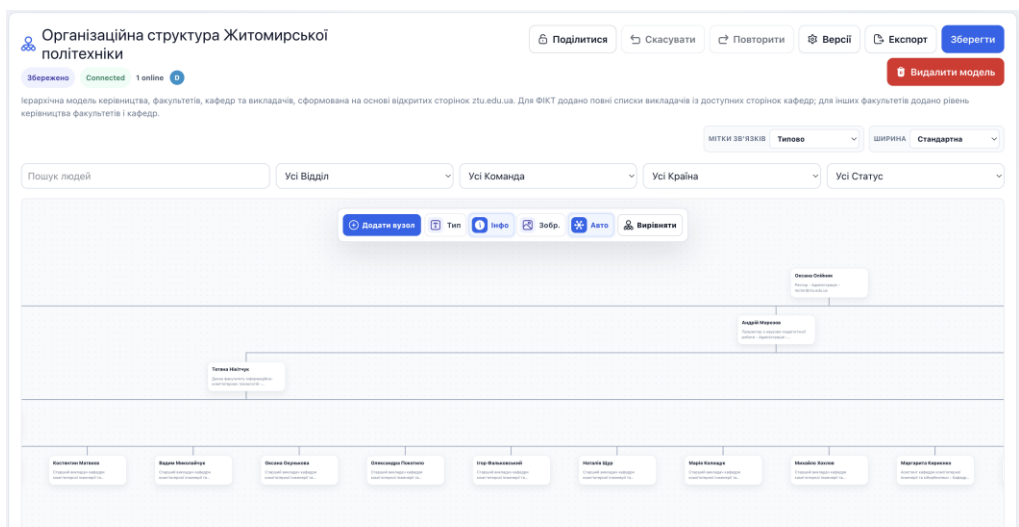


Рис. 3.13. Приклад організаційної ієрархії

Для часової моделі користувач створює події, періоди або часові проміжки. Кожен елемент має часові параметри, які визначають його положення на часовій осі. Користувач може масштабувати шкалу, переміщуватися по таймлайну, додавати нові події та змінювати їх властивості. Такий режим роботи є зручним для побудови історичних послідовностей, планів або хронологічних структур.

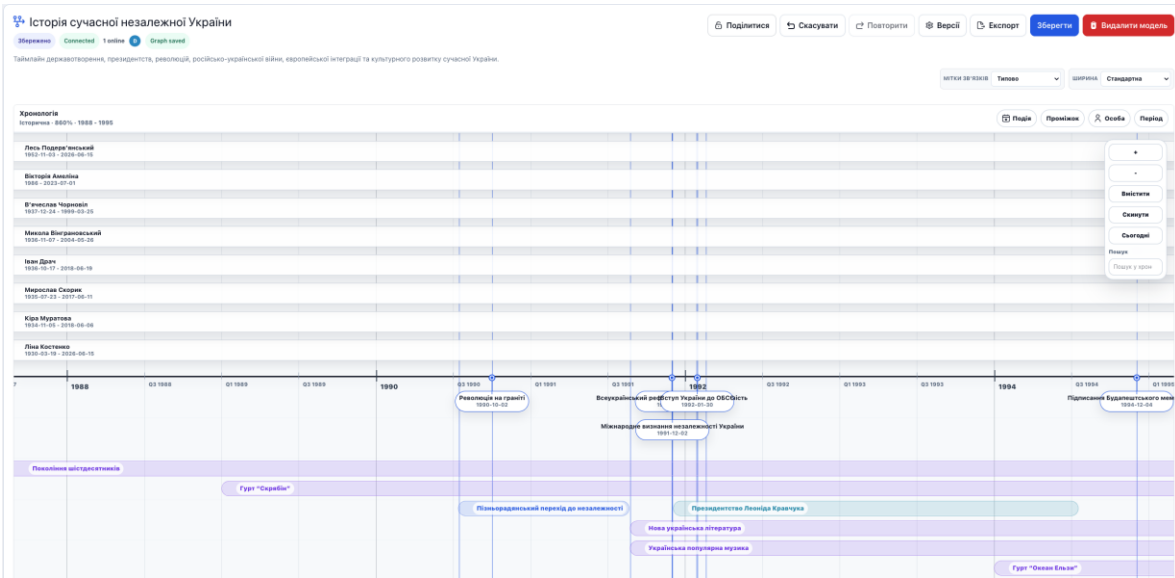


Рис. 3.14. Приклад часової моделі у вигляді таймлайну

Загальний сценарій створення та редагування моделі наведено в таблиці 3.2.

Таблиця 3.2.

Демонстраційний сценарій роботи користувача з моделлю.

Етап	Дія користувача	Відображення в інтерфейсі	Результат
Вхід до системи	Користувач авторизується	Відкривається головна сторінка	Користувач отримує доступ до власних моделей
Створення моделі	Користувач натискає кнопку створення моделі	Відображається форма створення	Створюється нова модель обраного типу
Відкриття редактора	Користувач відкриває створену модель	Відображається робоче поле	Модель готова до редагування
Додавання вузлів	Користувач додає елементи на поле	На полі з'являються вузли	Створюється початкова структура

Створення зв'язків	Користувач поєднує вузли	Між елементами з'являються лінії зв'язків	Формується інформаційна модель
Редагування властивостей	Користувач вибирає елемент і змінює дані	Відкривається панель властивостей	Дані елемента оновлюються
Збереження результату	Користувач завершує редагування	Система фіксує зміни	Поточний стан моделі збережено
Експорт або поширення	Користувач обирає відповідну дію	Відкриваються параметри експорту або доступу	Модель можна використати поза поточним сеансом

Після завершення редагування користувач може виконати додаткові дії з моделлю. До таких дій належать експорт, імпорт, створення версії або надання спільного доступу іншим користувачам. Експорт дозволяє зберегти модель у зовнішньому форматі, імпорт – завантажити підготовлені дані, а спільний доступ надати іншому користувачу право перегляду або редагування.



Рис. 3.15. Елементи керування експортом або спільним доступом до моделі

Окремий сценарій роботи передбачений для десктопної версії системи. У цьому випадку користувач може працювати з моделями локально, зберігаючи їх на власному пристрої. Після внесення змін модель може бути синхронізована із серверною частиною. Такий режим дозволяє використовувати систему не лише

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

як вебзастосунок, а й як настільну програму для локальної роботи з інформаційними структурами.

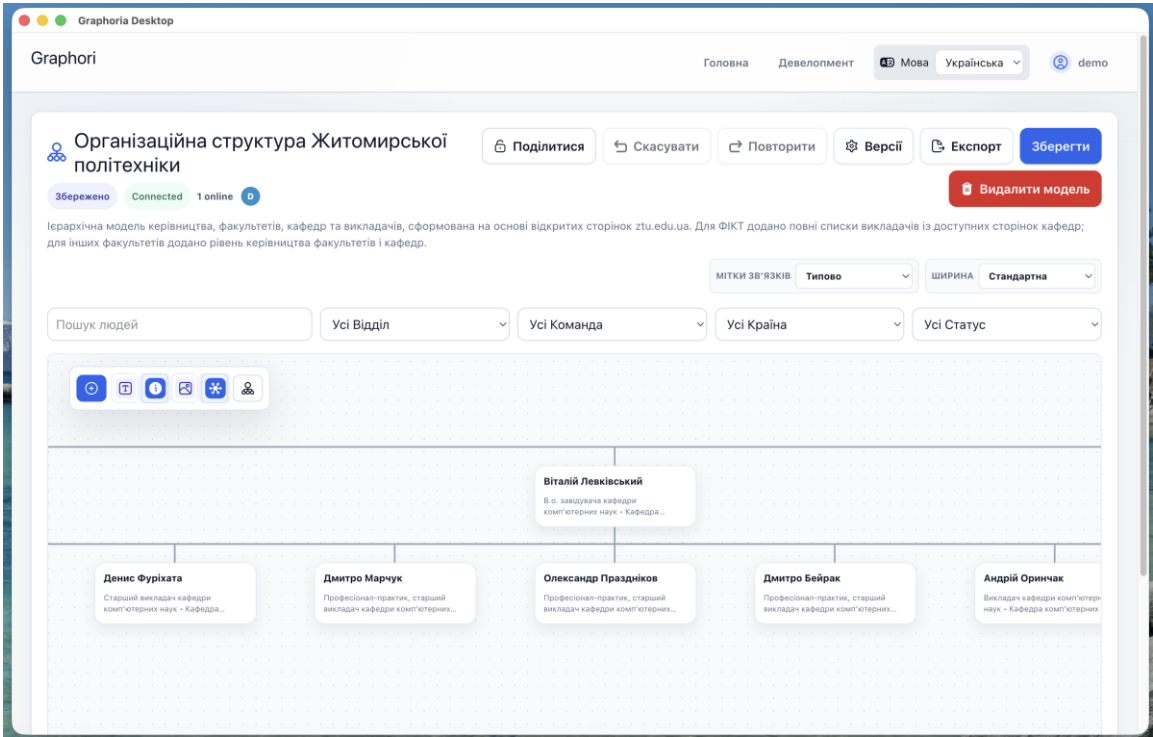


Рис. 3.16. Робота з моделлю у десктопній версії системи

Сценарій роботи з десктопною версією наведено в таблиці 3.3.

Таблиця 3.3.

Сценарій роботи з десктопною версією системи

Крок	Дія користувача	Результат
1	Користувач запускає десктопний застосунок	Відкривається локальний інтерфейс системи
2	Користувач створює або відкриває локальну модель	Модель завантажується з локального сховища
3	Користувач редагує вузли, зв’язки та властивості	Структура моделі змінюється
4	Користувач зберігає модель локально	Дані записуються у локальне сховище
5	Користувач запускає синхронізацію	Локальна модель узгоджується із серверною версією

Таким чином, порядок роботи користувача із системою Graphori охоплює повний цикл взаємодії з інформаційними моделями. Користувач може створити

модель, обрати її тип, побудувати структуру з вузлів і зв'язків, відредагувати властивості елементів, зберегти результат, експортувати модель або надати до неї доступ. Наявність декількох режимів роботи редактора дозволяє застосовувати систему для різних типів структурованої інформації, а десктопна версія додатково забезпечує можливість локальної роботи та синхронізації.

3.3. Налаштування, встановлення та розгортання системи.

Налаштування, встановлення та розгортання системи Graphorі є важливим етапом перевірки її працездатності, оскільки система складається з декількох взаємопов'язаних частин: клієнтського вебзастосунку, серверної частини, бази даних PostgreSQL, десктопного застосунку та допоміжних конфігураційних файлів. Для спрощення запуску використовується контейнеризація, що дозволяє розгорнути основні компоненти системи в ізолюваному середовищі без необхідності ручного налаштування кожної залежності окремо.

Особливістю розгортання такої системи є необхідність узгодженого налаштування всіх компонентів. Некоректно вказана адреса серверної частини, помилка у змінних середовища, неправильні параметри підключення до бази даних або відсутність необхідних міграцій можуть призвести до того, що окремі частини системи будуть запущені, але не зможуть взаємодіяти між собою. Тому процес встановлення має включати не лише запуск сервісів, а й перевірку конфігурації, доступності API, підключення до бази даних, коректності змінних середовища та працездатності основних користувацьких сценаріїв.

У процесі розгортання системи необхідно забезпечити коректну взаємодію між frontend, backend і базою даних. Клієнтська частина надсилає HTTP-запити до серверної частини, серверна частина обробляє бізнес-логіку та звертається до PostgreSQL, а база даних зберігає інформацію про користувачів, моделі, вузли, зв'язки, атрибути, версії та права доступу. Десктопна версія системи додатково використовує локальне сховище для автономної роботи з моделями та може синхронізувати дані із сервером..

Основні компоненти, які беруть участь у запуску системи, наведено в таблиці 3.4.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

Компоненти розгортання системи

Компонент	Призначення	Спосіб запуску
PostgreSQL	Зберігання основних даних системи	Docker Compose
Backend	Обробка API-запитів, автентифікація, бізнес-логіка	Docker Compose або локальний запуск
Frontend	Вебінтерфейс користувача	Docker Compose або локальний запуск
Desktop	Настільна версія застосунку	Локальний запуск або збірка Electron
Prisma	Робота зі схемою бази даних і міграціями	Команди backend-проекту

Перед запуском системи необхідно підготувати змінні середовища. Вони використовуються для передавання конфігураційних параметрів без жорсткого закріплення значень у програмному коді. Такий підхід відповідає практиці розділення конфігурації та реалізації, а також спрощує перенесення системи між локальним, тестовим і продуктивним середовищами.

Для серверної частини основними параметрами є адреса підключення до бази даних, секрет для підпису JWT-токенів, порт запуску. Приклад структури .env для backend наведено нижче.

```

DATABASE_URL="postgresql://Graphori_user:Graphori_password@postgres:5432/Graphori?schema=publish"

POSTGRES_USER=Graphori_user
POSTGRES_PASSWORD=Graphori_password
POSTGRES_DB=Graphori

JWT_SECRET=change_this_secret_value
PORT=3000

```

Для клієнтської частини також використовуються змінні середовища. Вони визначають адресу backend API, адресу WebSocket-з'єднання та ідентифікатор Google-клієнта для авторизації через браузер. Приклад конфігурації frontend наведено нижче.

```

VITE_API_URL=http://localhost:3000/api/v1
VITE_SOCKET_URL=http://localhost:3000

```

Важливо враховувати, що значення адрес backend API залежить від способу запуску системи. Якщо всі компоненти запускаються локально на одному комп'ютері, для frontend можна використовувати localhost. Якщо вебзастосунок відкривається з іншого пристрою у локальній мережі, замість localhost необхідно вказати IP-адресу комп'ютера або домен, на якому запущено backend. Це потрібно для того, щоб запити з клієнтської частини надсилалися на правильний сервер, а не на локальну адресу пристрою користувача.

Для контейнеризованого запуску використовується Docker Compose. Він дозволяє одночасно запустити базу даних, backend і frontend як окремі сервіси. Кожен сервіс має власну конфігурацію, але всі вони працюють у спільній Docker-мережі, що забезпечує взаємодію між контейнерами за іменами сервісів.

Приклад логіки запуску через Docker Compose передбачає наявність трьох основних сервісів: postgres, api і web. Сервіс postgres запускає базу даних, сервіс api відповідає за серверну частину, а сервіс web – за клієнтський застосунок. Для бази даних використовується volume, який дозволяє зберігати дані між перезапусками контейнерів (див. додат. Г).

Наведений приклад демонструє загальну структуру запуску, однак конкретні значення портів, назв сервісів і змінних середовища можуть відрізнитися залежно від конфігурації проєкту. У межах локального розгортання достатньо підготувати .env файл, переконавшись в наявності Docker і Docker Compose, після чого виконати команду запуску:

```
docker compose up --build
```

Після виконання цієї команди Docker Compose збирає образи backend і frontend, запускає PostgreSQL та створює необхідну мережу між контейнерами. У разі успішного запуску користувач може відкрити вебінтерфейс у браузері, а backend API стає доступним за налаштованою адресою.

☐	▼	●	graphoria	-	-	-	8.36%	9 ho				
☐		●	postgres	97bee3dd8cff	postgres:1	5432:5432	3.9%	9 ho				
☐		●	api	063b29f4e423	graphoria-1	3000:3000	0.56%	9 ho				
☐		●	web	1961c9403c03	graphoria-1	5173:5173	3.9%	9 ho				

Рис. 3.17. Запуск контейнерів системи через Docker Compose

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

Після запуску контейнерів необхідно перевірити стан сервісів. Для цього можна використати команду перегляду активних контейнерів.

```
docker compose ps
```

У результаті повинні відображатися запущені сервіси бази даних, backend і frontend. Якщо один із контейнерів не запущений або завершив роботу з помилкою, слід переглянути його журнали.

```
docker compose logs api
```

```
docker compose logs web
```

```
docker compose logs postgres
```

```
[11:13:40 PM] File change detected. Starting incremental compilation...
graphoria-api |
graphoria-api | [11:13:40 PM] Found 0 errors. Watching for file changes.
graphoria-api |
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [NestFactory] Starting Nest application...
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] AppModule dependencies initialized +10ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] PassportModule dependencies initialized +0ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] ConfigHostModule dependencies initialized +1ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] HealthModule dependencies initialized +0ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] ConfigModule dependencies initialized +4ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] PrismaModule dependencies initialized +1ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] PermissionsModule dependencies initialized +0ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] JwtModule dependencies initialized +0ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] JwtModule dependencies initialized +1ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] GraphLayoutModule dependencies initialized +1ms
graphoria-api | [Nest] 121 - 06/14/2026, 11:13:40 PM    LOG [InstanceLoader] AdminModule dependencies initialized +0ms
```

Рис. 3.18. Перевірка стану контейнерів системи

Для роботи з базою даних використовується Prisma ORM. Після налаштування змінної DATABASE_URL необхідно виконати генерацію Prisma Client і застосувати міграції бази даних. Ці дії забезпечують відповідність структури PostgreSQL описаній схемі даних.

```
npx prisma generate
```

```
npx prisma migrate deploy
```

У середовищі розробки може використовуватися команда створення та застосування нової міграції.

```
npx prisma migrate dev
```

Результатом виконання міграцій є створення таблиць, необхідних для роботи системи. До них належать таблиці користувачів, моделей, вузлів, зв'язків, атрибутів, версій, налаштувань відображення та правил доступу. Після цього

backend може коректно виконувати операції створення, читання, оновлення та видалення даних.

Після запуску backend і frontend користувач може перевірити роботу системи через браузер. Для локального запуску frontend відкривається за адресою, визначеною конфігурацією, наприклад `http://localhost:5173`. Після відкриття застосунку користувач може зареєструватися, увійти до системи, створити модель і перевірити її збереження. Якщо авторизація, створення моделі та відображення редактора працюють коректно, це підтверджує правильність базового розгортання.

Десктопна версія системи запускається окремо від контейнеризованої серверної інфраструктури. Вона використовує Electron і містить той самий користувацький інтерфейс, що й вебверсія, але додатково підтримує локальне збереження моделей. Перед запуском desktop-застосунку необхідно встановити залежності проєкту та виконати команду запуску у режимі розробки.

```
npm install
```

```
npm run desktop:dev
```

Для створення інсталяційного пакета або готової збірки десктопного застосунку використовується команда збірки.

```
npm run desktop:build
```

Після запуску десктопної версії користувач може працювати з моделями локально. Якщо налаштовано адресу backend API, десктопний застосунок також може виконувати синхронізацію локальних моделей із сервером. Це дозволяє використовувати систему як локальний інструмент і водночас зберігати можливість обміну даними з хмарною версією.

Під час розгортання системи важливо перевірити не лише запуск окремих компонентів, а й їхню взаємодію. Backend повинен мати доступ до PostgreSQL, frontend повинен надсилати запити до правильного API, WebSocket-з'єднання повинно встановлюватися з backend, а desktop-застосунок повинен коректно працювати з локальним сховищем і серверною синхронізацією.

Контрольний перелік перевірки після розгортання наведено в таблиці 3.5.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

Перевірка працездатності системи після розгортання

Перевірка	Очікуваний результат
Запуск PostgreSQL	Контейнер бази даних працює, підключення доступне
Запуск backend	API відповідає на запити
Запуск frontend	Вебінтерфейс відкривається у браузері
Підключення frontend до backend	Запити авторизації та моделей виконуються без помилок
Виконання міграцій	У базі даних створено необхідні таблиці
Реєстрація користувача	Новий користувач створюється в системі
Вхід до системи	Користувач отримує доступ до головної сторінки
Створення моделі	Модель зберігається у базі даних
Відкриття редактора	Редактор моделі відображається коректно
WebSocket-з'єднання	Оновлення моделі передаються в реальному часі
Google OAuth	Вхід через Google працює після налаштування Client ID
Desktop-застосунок	Локальний інтерфейс запускається
Синхронізація desktop із backend	Локальні моделі передаються на сервер або оновлюються з нього

У разі розгортання системи не на локальному комп'ютері, а на віддаленому сервері, необхідно додатково налаштувати домен, HTTPS-сертифікат, змінні середовища для продуктивного середовища та правила доступу до портів. У продуктивному режимі не рекомендується використовувати значення localhost у клієнтській конфігурації, оскільки браузер користувача буде намагатися звертатися до власного пристрою, а не до серверної частини. Тому для production-розгортання потрібно вказувати публічний домен або IP-адресу backend API.

Також доцільно відокремлювати конфігурації для локального та продуктивного середовища. У локальному середовищі можуть використовуватися тестові значення змінних, відкриті порти та спрощені параметри запуску. У продуктивному середовищі необхідно використовувати

надійні секрети, обмежений доступ до бази даних, HTTPS, стабільний домен і безпечно зберігання конфігураційних параметрів.

Таким чином, налаштування та розгортання системи Graphori передбачає підготовку змінних середовища, запуск PostgreSQL, backend і frontend через Docker Compose, застосування міграцій бази даних, перевірку роботи вебінтерфейсу та запуск десктопної версії. Використання контейнеризації спрощує процес встановлення системи, зменшує залежність від локального середовища розробника та забезпечує можливість подальшого перенесення застосунку на віддалений сервер.

3.4. Керування доступом та ролями користувачів.

Керування доступом є важливою частиною роботи системи Graphori, оскільки інформаційні моделі можуть містити як особисті, так і публічні або спільні дані. Система повинна забезпечувати можливість розмежування прав між різними категоріями користувачів, контролювати доступ до моделей, обмежувати адміністративні дії та підтримувати модерацію публічного контенту. У межах користувацького інтерфейсу це проявляється через різні доступні дії, видимість елементів керування та можливість виконувати або не виконувати певні операції залежно від ролі користувача.

Розмежування прав доступу дозволяє захистити приватні моделі користувачів і водночас підтримати спільну роботу з інформаційними структурами. Права визначають не лише можливість перегляду моделі, а й доступність окремих дій: редагування, зміну видимості, надання доступу іншим користувачам та виконання адміністративних операцій.

У системі передбачено декілька категорій користувачів: авторизований користувач, гість, адміністратор і суперадміністратор. Гість може працювати лише з відкритою частиною системи, авторизований користувач має доступ до власного робочого середовища, адміністратор має додаткові інструменти керування користувачами та публічними моделями, а суперадміністратор може керувати адміністраторами й дозволами (див. табл. 3.6).

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

Ролі користувачів у системі Graphori.

Роль	Призначення	Основні можливості
Гість	Неавторизований користувач системи	Перегляд публічних моделей, реєстрація, вхід до системи
Користувач	Основний учасник системи	Створення, редагування, збереження, імпорт, експорт і поширення власних моделей
Адміністратор	Користувач із розширеними правами	Модерація публічних моделей, перегляд користувачів, обмеження окремих дій користувачів
Суперадміністратор	Користувач із повним адміністративним доступом	Керування адміністраторами, ролями та адміністративними дозволами

Для гостя система доступна в обмеженому режимі. Такий користувач може переглядати публічні моделі, якщо вони схвалені для відображення, а також перейти до сторінки входу або реєстрації. Гість не може створювати власні моделі, редагувати існуючі структури, відкривати приватні моделі або користуватися адміністративними можливостями. Такий підхід забезпечує базовий доступ до відкритого контенту, але не дозволяє змінювати дані системи без автентифікації.

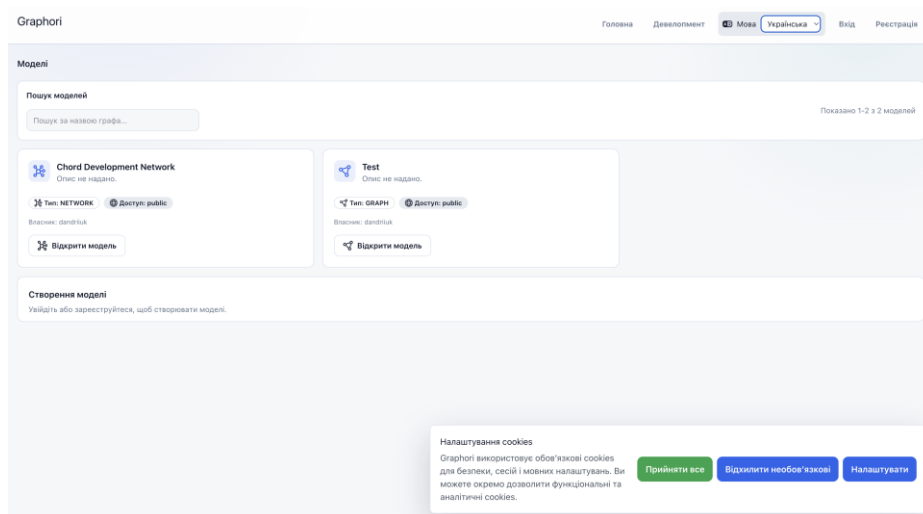


Рис. 3.19. Відображення публічної моделі для неавторизованого користувача

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

Авторизований користувач є основним актором системи. Після входу він отримує доступ до власних моделей, може створювати нові інформаційні структури, редагувати вузли та зв'язки, змінювати властивості елементів, виконувати імпорт і експорт, а також керувати доступом до моделей. Саме для цієї ролі призначений основний функціонал системи візуального моделювання.

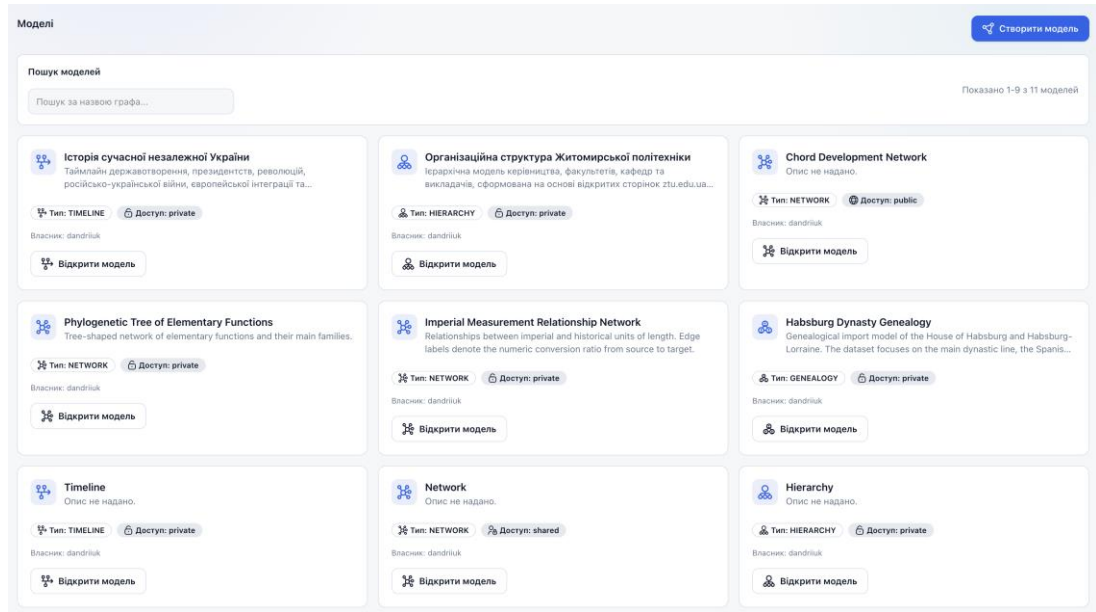


Рис. 3.20. Список моделей авторизованого користувача

Кожна модель у системі має власника. Власник має повний доступ до моделі та може змінювати її структуру, параметри, видимість і правила спільного доступу. Якщо модель є приватною, вона доступна лише власнику. Якщо модель публічна, її можуть переглядати інші користувачі після проходження модерації. Якщо модель є спільною, власник може надати доступ конкретним користувачам із визначеним рівнем дозволів.

У системі використовуються три основні типи видимості моделей: приватна, публічна та спільна (див. табл. 3.7). Приватна модель призначена для індивідуальної роботи користувача. Публічна модель може бути доступною ширшому колу користувачів, але її відображення контролюється через механізм модерації. Спільна модель використовується для надання доступу конкретним користувачам, наприклад для перегляду або редагування (див. рис. 3.21).

Таблиця 3.7.

Типи доступу до моделей

Тип доступу	Опис	Хто має доступ
Приватний	Приватна модель	Лише власник моделі
Публічний	Публічна модель	Інші користувачі після схвалення моделі
Спільний	Спільна модель	Власник і користувачі, яким надано доступ

 Спільний доступ

Змініть видимість графа або додайте користувачів для спільного доступу.

Закрити

ПОТОЧНА ВИДИМІСТЬ

 Приватний



Спільний

Запрошені користувачі можуть його відкрити.



Публічний

Видимий у каталозі після схвалення.

Скасувати

 Зберегти зміни

Рис. 3.21. Налаштування видимості моделі

Для спільних моделей передбачено два основні рівні доступу: перегляд і редагування. Доступ на перегляд дозволяє іншому користувачу відкривати модель і аналізувати її структуру, але не змінювати її вміст. Доступ на редагування дозволяє виконувати зміни у моделі: додавати вузли, редагувати властивості, створювати або змінювати зв'язки. Розмежування цих дозволів є важливим, оскільки воно дозволяє організувати спільну роботу без втрати контролю над моделлю.

Таблиця 3.8.

Рівні спільного доступу до моделі

Рівень доступу	Можливості користувача	Приклад використання
Перегляд	Перегляд структури моделі без права зміни	Надання моделі для ознайомлення або демонстрації
Редагування	Перегляд і редагування структури моделі	Спільна робота над побудовою інформаційної структури

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

Спільний доступ
Змінити видимість графа або додайте користувачів для спільного доступу.
Закрити

ПОТОЧНА ВИДИМІСТЬ
Приватний

Спільний
Запрошені користувачі можуть його відкрити.

Публічний
Видимий у каталозі після схвалення.

Ім'я користувача

Права

Перегляд
Додати користувача

admin

Редагування
Видалити

Скасувати
Зберегти зміни

Рис. 3.22. Вікно налаштування спільного доступу до моделі

Процес надання спільного доступу виконується власником моделі. Користувач відкриває параметри доступу, обирає іншого користувача або вводить його ідентифікаційні дані, визначає рівень доступу та зберігає зміни. Після цього модель стає доступною для обраного користувача відповідно до визначених прав. Якщо власник змінює або скасовує доступ, користувач втрачає відповідні можливості.

Адміністратор має розширений доступ до системи. Основними його завданнями є контроль публічного контенту та керування користувачами. Адміністратор може переглядати список користувачів, бачити інформацію про їх статус, обмежувати можливість створення публічних моделей або блокувати користувача у разі порушення правил використання системи. Такі функції важливі для підтримки безпечного та контрольованого середовища.

Користувачі				
КОРИСТУВАЧ	РОЛЬ	СТАТУС	ПУБЛІЧНІ ГРАФИ	ДІЇ
superadmin superadmin@graphoria.com	Суперадміністратор	Активний	Дозволено	Заблокувати Обмежити
admin admin@graphoria.com	Адміністратор	Активний	Дозволено	Заблокувати Обмежити
user user@graphoria.com	Користувач	Активний	Дозволено	Заблокувати Обмежити
dandriuk andriuk.denys@gmail.com	Користувач	Активний	Дозволено	Заблокувати Обмежити

Рис. 3.23. Адміністративна панель керування користувачами

Окремою адміністративною функцією є модерація публічних моделей. Якщо користувач робить модель публічною, вона може потребувати перевірки перед відображенням іншим користувачам. Адміністратор переглядає такі моделі та змінює їх статус: схвалює або відхиляє. У разі відхилення може бути вказана причина, що дозволяє користувачу зрозуміти, чому модель не була опублікована.

Таблиця 3.9.

Статуси модерації публічної моделі

Статус	Значення	Подальша дія
Not reviewed	Модель ще не перевірена	Очікує рішення адміністратора
Approved	Модель схвалена	Може бути доступна як публічна
Rejected	Модель відхилена	Не відображається як публічна, може містити причину відхилення

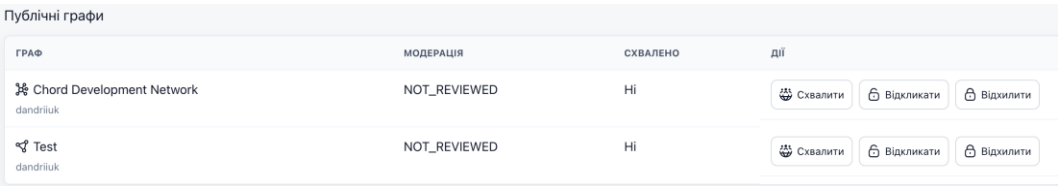


Рис. 3.24. Модерація публічних моделей в адміністративній панелі

Суперадміністратор має найвищий рівень доступу. На відміну від звичайного адміністратора, він може керувати адміністративними ролями та дозволами. Це означає, що суперадміністратор може призначати користувачам роль адміністратора, відкликати її або визначати набір конкретних адміністративних можливостей. Такий підхід дозволяє не надавати всім адміністраторам однаковий повний доступ, а розмежовувати їхні повноваження відповідно до потреб системи.

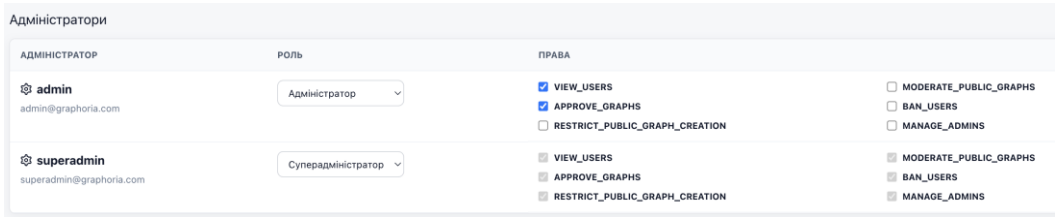


Рис. 3.25. Керування адміністраторами та їх дозволами

Для формалізації доступу до основних функцій системи доцільно використати матрицю прав. Вона показує, які дії доступні кожній ролі користувача та які операції потребують розширених прав. При цьому важливо розрізняти адміністративні повноваження та доступ до вмісту моделей. Адміністратор і суперадміністратор мають розширені можливості керування системою, однак це не означає автоматичного доступу до всіх приватних моделей користувачів. Доступ до конкретної моделі визначається її власником, типом видимості та наданими дозволами. Адміністративні ролі дають змогу виконувати модерацію публічних моделей, керувати користувачами та дозволами, але не повинні порушувати принцип приватності даних.

Таблиця 3.10.

Матриця доступу користувачів до функцій системи

Функція	Гість	Користувач	Адміністратор	Суперадміністратор
Перегляд публічних моделей	Так	Так	Так	Так
Реєстрація та вхід	Так	Так	Так	Так
Створення власної моделі	Ні	Так	Так	Так
Редагування власної моделі	Ні	Так	Так	Так
Імпорт та експорт моделей	Ні	Так	Так	Так
Надання спільного доступу	Ні	Так	Так	Так
Перегляд спільної моделі	Ні	Так, якщо доступ надано	Так, якщо доступ надано	Так, якщо доступ надано
Редагування спільної моделі	Ні	Так, якщо надано право Edit	Так, якщо надано право Edit	Так, якщо надано право Edit
Перегляд публічної моделі на модерації	Ні	Ні	Так	Так
Зміна видимості власної моделі	Ні	Так	Так	Так

Модерація публічних моделей	Ні	Ні	Так	Так
Обмеження користувачів	Ні	Ні	Так	Так
Керування адміністраторами	Ні	Ні	Ні	Так
Керування адміністративними дозволами	Ні	Ні	Ні	Так

У процесі роботи із системою користувач бачить лише ті елементи інтерфейсу, які відповідають його ролі та дозволам. Наприклад, звичайний користувач не має доступу до адміністративної панелі, а гість не бачить функцій створення або редагування моделей. Якщо користувач має право лише на перегляд спільної моделі, інтерфейс не повинен надавати йому можливість змінювати вузли або зв'язки. Це зменшує ризик помилкових дій і робить роботу з системою більш зрозумілою.

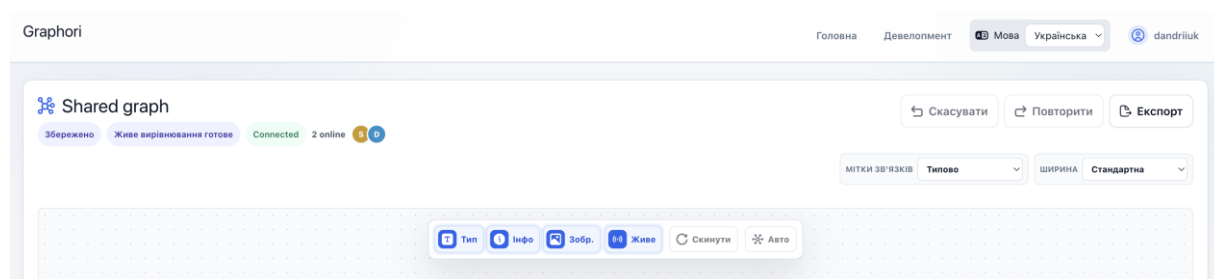


Рис. 3.26. Приклад обмеження функцій редагування для користувача з правом View

Керування доступом також пов'язане з профілем користувача. На сторінці профілю користувач може переглядати власні дані, змінювати персональну інформацію, мову інтерфейсу та інші налаштування. Хоча ці дії не належать до адміністративного керування, вони є частиною персонального доступу до системи, оскільки дозволяють підтримувати актуальність облікового запису.

Graphori

Головна Девелопмент Мова Українська dandriuk

Профіль користувача

Керуйте персональними даними, налаштуваннями акаунта, фото та вподобаннями.

dandriuk
@dandriuk

Особиста інформація

Ці дані використовуються лише для відображення профілю.

Ім'я

Прізвище

Телефон

Зберегти

Фото профілю

JPG, PNG або WebP до 2 MB.

Завантажити фото

Видалити фото

Рис. 3.27 Сторінка профілю користувача (початок)

Налаштування акаунту

Безпека та приватність інформації акаунта.

Роль

Статус

Google акаунт

Вхід через Google не налаштований.

Username

Змінити username

Email

Змінити email

Поточний пароль

Новий пароль

Змінити пароль

Персоналізація

Мова зберігається в акаунті та застосовується одразу.

Бажана мова

Приватність і cookies

Кошик графів

Видалені графи зберігаються 30 днів, після чого остаточно видаляються з бази даних.

Рис. 3.28 Сторінка профілю користувача (продовження)

Таким чином, система Graphori підтримує комплексну модель керування доступом, яка поєднує ролі користувачів, типи видимості моделей, рівні спільного доступу та адміністративні дозволи. Такий підхід дозволяє використовувати систему як для індивідуальної роботи, так і для спільного редагування моделей, публікації відкритого контенту та адміністративного контролю. Розмежування прав доступу підвищує безпеку даних, зменшує ризик несанкціонованих змін і забезпечує контрольовану роботу користувачів із різними функціями системи.

3.5. Тестування роботи системи.

Тестування системи Graphori виконувалося з метою підтвердження її працездатності, перевірки основних користувацьких сценаріїв, коректності взаємодії frontend і backend, правильності збереження даних у PostgreSQL,

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

стабільності API та надійності механізмів керування доступом. Оскільки система є повноцінним клієнт-серверним програмним продуктом, перевірка проводилася не лише через інтерфейс користувача, а й на рівні REST API, бази даних, сценаріїв авторизації та критичних ситуацій, які можуть призвести до некоректної роботи або порушення безпеки.

Тестування доцільно поділити на декілька напрямів: функціональне тестування інтерфейсу, тестування API, перевірку збереження даних, тестування ролей і прав доступу, перевірку критичних сценаріїв та тестування розгортання системи. Такий підхід дозволяє оцінити систему не лише з позиції кінцевого користувача, а й з позиції стабільності серверної логіки та безпеки обробки даних.

Таблиця 3.11.

Основні напрями тестування системи.

Напрямок тестування	Мета перевірки	Спосіб підтвердження
Тестування інтерфейсу	Перевірка основних дій користувача у вебзастосунку	Скріншоти інтерфейсу, таблиці результатів
Тестування API	Перевірка REST-запитів, статус-кодів і відповідей backend	Postman/Swagger/браузерні DevTools
Тестування бази даних	Перевірка збереження користувачів, моделей, вузлів і зв'язків	Перегляд записів у PostgreSQL або Prisma Studio
Тестування доступу	Перевірка ролей, private/public/shared, view/edit	Сценарії з різними користувачами
Тестування критичних сценаріїв	Перевірка стійкості до помилок, некоректних дій і заборонених операцій	API-відповіді, повідомлення про помилки, відсутність аварійного завершення
Тестування розгортання	Перевірка запуску PostgreSQL, backend, frontend і desktop	Docker logs, список контейнерів, відкритий застосунок

Першим етапом було виконано функціональне тестування через інтерфейс користувача. Цей тип тестування дозволяє перевірити, чи може користувач виконати основний сценарій роботи з системою: зареєструватися або увійти,

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

створити модель, додати вузли, створити зв'язки, відредагувати властивості, зберегти результат і виконати експорт або надати доступ до моделі.

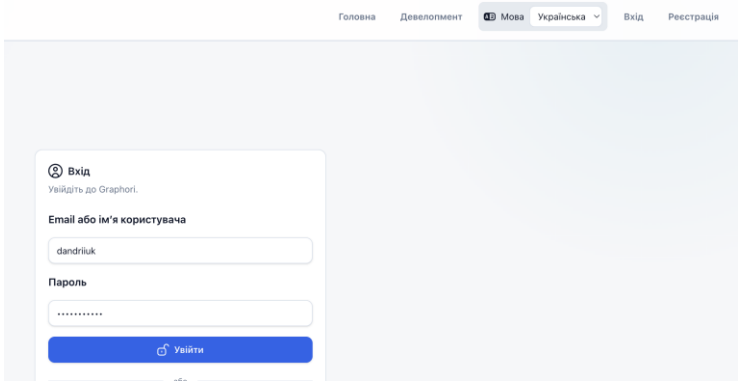


Рис. 3.29. Форма входу користувача в систему

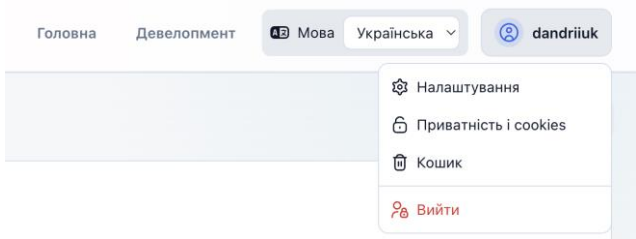


Рис. 3.30. Успішний вхід користувача до системи

Після входу до системи було перевірено створення нової моделі. Користувач заповнює форму створення, вказує назву, опис і тип моделі. Очікуваним результатом є поява нової моделі у списку доступних моделей і можливість відкрити її в редакторі.

Таблиця 3.12.

Перевірка створення та відкриття моделі

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Підтвердження
1	Створення нової моделі	Модел ь створюється та з'являється у списку	Модел ь відображається на головній сторінці	Додаток А, рис. 4
2	Відкриття створеної моделі	Відкривається редактор моделі	Редактор відкривається коректно	Додаток А, рис. 5
3	Вибір типу моделі	Редактор адаптується до типу моделі	Відображається відповідний режим редактора	Додаток А, рис. 6

Наступним напрямом було тестування REST API. Перевірка API є необхідною, оскільки саме серверна частина забезпечує створення, збереження, оновлення, отримання та захист даних. Тестування виконувалося шляхом надсилання HTTP-запитів до основних endpoint-ів системи за допомогою Postman [32]. Для кожного запиту перевірялися HTTP-метод, очікуваний і фактичний статус відповіді, час виконання запиту, структура відповіді та коректність обробки помилкових або неавторизованих дій.

Для перевірки REST API було використано Postman Collection Runner. У межах одного запуску виконано 33 запити та 38 тестових перевірок. Загальна тривалість виконання становила 942 мс, середній час відповіді REST API становить 10 мс. Помилкок виконання тестів виявлено не було, що підтверджує коректну роботу основних endpoint-ів і механізмів обробки помилкових запитів (див. додат. Д рис. 1-4).

Результати тестування REST API показали, що серверна частина коректно обробляє як успішні, так і заборонені запити. Для приватних маршрутів без JWT-токена система повертає статус 401 Unauthorized, а для дій, які вимагають адміністративних прав, 403 Forbidden. Це підтверджує, що механізми автентифікації та авторизації працюють відповідно до очікуваної логіки.

За результатами запуску колекції більшість основних CRUD-операцій виконувалася зі статусами 200 або 201: створення моделі виконано за 54 мс, отримання списку моделей за 10 мс, отримання моделі за ідентифікатором за 8 мс, оновлення метаданих моделі за 10 мс, створення вузлів за 13 мс і 11 мс, створення зв'язку за 12 мс, а оновлення зв'язку за 11 мс. Окремо перевірялися негативні сценарії: 401 Unauthorized для запитів без токена або з некоректним JWT, 403 Forbidden для звернення до адміністративних endpoint-ів без відповідних прав, 400 Bad Request для створення моделі з порожньою назвою та 404 Not Found для звернення до неіснуючих моделей, вузлів або зв'язків. Отримані результати свідчать, що API коректно обробляє як успішні операції, так і помилкові або заборонені запити.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

Перевірка критичних сценаріїв і потенційних ризиків

Критичний сценарій	Потенційна небезпека	Очікувана реакція системи	Фактичний результат	Підтвердження
Спроба відкрити приватну модель іншого користувача	Несанкціонований доступ до даних	Система повертає 403 Forbidden або повідомлення про відсутність доступу	Доступ заборонено	Рис. 3.31
Спроба редагування спільної моделі з правом View	Несанкціонована зміна даних	Запит на зміну даних відхиляється, елементи редагування недоступні	Зміни не застосовано	Рис. 3.26
Створення зв'язку з неіснуючим вузлом	Порушення цілісності структури моделі	Система повертає 400 Bad Request або 404 Not Found	Зв'язок не створено	додаток Д, рис. 4
Передавання некоректних значень атрибутів	Потрапляння невалідних даних у базу	Система виконує валідацію та повертає 400 Bad Request	Некоректні дані не прийнято	додаток Д, рис. 4
Відкриття моделі після soft delete	Доступ до видалених даних	Видалена модель не відображається як активна	Модель прихована зі списку	додаток А, рис. 5
Відновлення несумісної версії моделі	Порушення структури даних	Система відхиляє відновлення через несумісність схеми або типу	Відновлення не виконано	додаток Д, рис. 3
Робота з великою моделлю	Зниження продуктивності редактора	Інтерфейс залишається придатним до роботи, критичних збоїв немає	Критичних збоїв не виявлено	додаток А, рис. 6

На відміну від попередньої таблиці, де перевірялися окремі REST API-запити, таблиця 3.14 відображає критичні сценарії, які можуть вплинути на безпеку, цілісність даних або стабільність роботи системи. Такий поділ дозволяє уникнути дублювання результатів тестування та окремо показати, як система реагує на потенційно небезпечні ситуації.

Окремо було виконано перевірку швидкодії вебінтерфейсу та візуального редактора моделей за допомогою інструментів розробника Google Chrome DevTools [33]. Для цього використовувалися вкладки Network і Performance, які дозволяють оцінити час завантаження сторінки, тривалість HTTP-запитів до backend, обсяг переданих даних, час рендерингу інтерфейсу та поведінку застосунку під час взаємодії з графом (див. додат. А рис. А.7). Під час перевірки аналізувалося відкриття головної сторінки, завантаження моделі в редакторі, масштабування полотна, переміщення вузлів і відображення зв'язків між елементами. Така перевірка є важливою, оскільки для системи візуального моделювання недостатньо лише коректного збереження даних: інтерфейс повинен залишатися придатним до інтерактивної роботи навіть під час відображення складних моделей із великою кількістю вузлів і зв'язків. Отримані результати використовувалися для оцінювання практичної швидкодії редактора та визначення потенційних напрямів подальшої оптимізації, зокрема зменшення кількості повторних рендерингів, оптимізації обробників подій і покращення роботи з великими графовими структурами.

Для підтвердження коректності керування доступом було протестовано декілька користувацьких ролей. Звичайний користувач не має доступу до адміністративної панелі, адміністратор може виконувати модерацію публічних моделей, а суперадміністратор може керувати адміністративними ролями й дозволами.

Також перевірено, що користувач із правом View не може редагувати спільну модель.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

You do not have access to this model

Рис. 3.31. Приклад повідомлення про помилку під час некоректного запиту або дії

Таблиця 3.15.

Тестування ролей і прав доступу

№	Роль користувача	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Гість	Перегляд публічної моделі	Модель відкривається лише в режимі перегляду	Перегляд доступний
2	Гість	Створення моделі	Дія недоступна	Створення заблоковано
3	Користувач	Редагування власної моделі	Редагування доступне	Зміни зберігаються
4	Користувач із View	Редагування спільної моделі	Редагування недоступне	Зміни заблоковано
5	Користувач із Edit	Редагування спільної моделі	Редагування доступне	Зміни зберігаються
6	Адміністратор	Модерація публічної моделі	Модель схвалюється або відхиляється	Статус змінюється

Також було протестовано імпорт і експорт моделей. Для експорту перевірялося формування файлу з актуальною структурою моделі. Для імпорту перевірялося, що система може завантажити коректний файл і відхиляє пошкоджені або неправильні дані.

Результати тестування показали, що система виконує основні функціональні сценарії, визначені на етапі проєктування. Було підтверджено

працездатність авторизації, створення та редагування моделей, роботи з вузлами й зв'язками, різних типів візуалізації, імпорту й експорту, керування доступом, адміністративної панелі, API та контейнеризованого розгортання.

Водночас тестування дозволило визначити потенційно критичні напрями, які потребують особливої уваги під час подальшого розвитку системи. До них належать перевірка доступу до чужих моделей, захист адміністративних маршрутів, коректна обробка пошкоджених імпортованих файлів, перевірка цілісності зв'язків між вузлами, обробка великих моделей і конфліктів синхронізації у десктопній версії. У межах проведеної перевірки критичних збоїв, які унеможливають використання системи, не виявлено.

Таким чином, тестування підтвердило працездатність системи Graphori як інтерактивного кросплатформеного засобу для візуального моделювання інформації. Поєднання перевірки інтерфейсу, API, бази даних, ролей доступу, критичних сценаріїв і розгортання дозволяє зробити висновок, що система виконує основні функціональні вимоги та може бути використана для побудови, редагування, збереження й поширення різних типів інформаційних моделей.

Висновки до третього розділу.

У третьому розділі було розглянуто практичну реалізацію інтерактивної кросплатформеної системи для візуального моделювання інформації. Описано реалізацію серверної частини на основі NestJS, клієнтської частини на React, підсистеми візуального моделювання, механізмів зберігання даних, безпеки та кросплатформеності. Реалізована система підтримує роботу з різними типами моделей, зокрема мережами, генеалогічними структурами, організаційними ієрархіями та часовими лініями.

У результаті розробки було створено програмний продукт, який поєднує модульну backend-архітектуру, інтерактивний frontend, реляційну базу даних PostgreSQL, JWT-автентифікацію, рольову модель доступу, Electron-десктопну версію та Docker-контейнеризацію. Запропонована реалізація відповідає сформованим функціональним і нефункціональним вимогам та створює основу для подальшого розвитку системи.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		101

ВИСНОВКИ

У ході виконання випускної кваліфікаційної роботи бакалавра було розроблено інтерактивну кросплатформену систему Graphori, призначену для створення, редагування, збереження та візуалізації структурованої інформації. Розроблений програмний продукт забезпечує роботу з різними типами інформаційних моделей, зокрема мережевими графами, ієрархічними структурами, таймлайнами, генеалогічними деревами, організаційними ієрархіями та моделями залежностей. Поставлену мету роботи було досягнуто, оскільки система реалізує повний цикл взаємодії користувача з інформаційною моделлю: від її створення до редагування, збереження, експорту, надання доступу та подальшого використання.

Розроблена система має клієнт-серверну архітектуру та складається з вебінтерфейсу, серверної частини, бази даних, десктопного застосунку й інфраструктурних засобів розгортання. Такий підхід дозволив розділити відповідальність між компонентами, забезпечити масштабованість, підтримку різних середовищ запуску та можливість подальшого розвитку продукту. Серверна частина відповідає за обробку запитів, автентифікацію, авторизацію, керування моделями, вузлами, зв'язками, атрибутами, версіями та правами доступу. Клієнтська частина забезпечує інтерактивну роботу користувача з моделями, а десктопна версія розширює можливості системи за рахунок локального збереження й синхронізації з сервером.

У межах роботи було реалізовано візуальний редактор моделей, який дозволяє користувачу працювати з інформаційними структурами у наочному графічному форматі. Користувач може створювати вузли, встановлювати зв'язки між ними, редагувати властивості елементів, змінювати їх положення, працювати з додатковими атрибутами та використовувати спеціалізовані режими для різних типів моделей. Для мережових графів передбачено довільне формування зв'язків між об'єктами, для ієрархічних структур – побудову батьківських і дочірніх елементів, для таймлайнів – відображення подій і часових

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		102

проміжків на часовій осі, для генеалогічних дерев – роботу з особами та родинними зв'язками, для організаційних ієрархій – відображення підпорядкування, посад і структурних одиниць, а для моделей залежностей – представлення напрямлених зв'язків між взаємозалежними елементами.

Одним із важливих результатів роботи є побудова гнучкої структури даних, яка підтримує різні типи моделей без необхідності створення окремої незалежної схеми для кожного сценарію. Для цього було використано універсальні сутності моделей, вузлів і зв'язків, а також додаткові таблиці для спеціалізованих даних, атрибутів, налаштувань відображення, версій і правил доступу. Такий підхід забезпечує розширюваність системи та дозволяє в майбутньому додавати нові типи моделей без повної зміни архітектури бази даних.

У системі реалізовано механізми безпеки та керування доступом. Підтримуються автентифікація користувачів, JWT-токени, вхід через Google, рольова модель доступу, адміністративні дозволи, приватні, публічні та спільні моделі. Користувач може керувати доступністю власних моделей, надавати іншим користувачам права перегляду або редагування, а адміністратори можуть виконувати модерацію публічного контенту та керувати обліковими записами. Це дозволяє використовувати систему як для індивідуальної роботи, так і для спільного опрацювання інформаційних структур.

Практична реалізація системи також передбачає підтримку імпорту, експорту, версіонування та відновлення моделей. Завдяки цьому користувач може переносити дані між середовищами, зберігати результати роботи у зовнішніх форматах, фіксувати важливі стани моделі та повертатися до попередніх версій у разі потреби. Наявність таких механізмів підвищує надійність роботи із складними інформаційними структурами та зменшує ризик втрати даних.

Окрему увагу приділено кросплатформеності системи. Вебверсія дозволяє працювати із системою через браузер, а десктопна версія надає можливість локальної роботи з моделями. Використання локального сховища у десктопному

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		103

застосунку дозволяє створювати та редагувати моделі на пристрої користувача, а механізм синхронізації забезпечує можливість обміну даними із серверною частиною. Це робить систему більш гнучкою та придатною для різних сценаріїв використання.

Працездатність системи було перевірено шляхом тестування основних користувацьких сценаріїв, REST API, механізмів авторизації, роботи з базою даних, керування доступом, адміністративних функцій, імпорту, експорту, розгортання та десктопного застосунку. Результати тестування підтвердили, що система коректно виконує основні функції, забезпечує збереження даних, обмежує доступ відповідно до ролей і дозволів, підтримує різні типи моделей і може бути розгорнута в контейнеризованому середовищі.

Практичне значення роботи полягає у створенні програмного продукту, який може застосовуватися для моделювання складних інформаційних структур у різних предметних областях. Система може бути корисною в освіті, управлінні, аналітиці, генеалогії, плануванні, дослідженні взаємозв'язків між об'єктами, побудові організаційних структур, представленні історичних або часових послідовностей і моделюванні залежностей між елементами. Завдяки універсальному підходу до моделей GraphoGі не обмежується однією вузькою предметною областю, а може бути адаптована до різних прикладних задач.

Отриманий результат відповідає сучасним вимогам до інтерактивних інформаційних систем: система має модульну архітектуру, підтримує клієнт-серверну взаємодію, централізоване зберігання даних, інтерактивний графічний інтерфейс, рольове розмежування доступу, контейнеризоване розгортання та кросплатформену роботу.

Подальший розвиток системи може передбачати розширення набору типів моделей, покращення алгоритмів автоматичного компонування графів, оптимізацію роботи з великими структурами, розширення можливостей спільного редагування, додавання аналітичних інструментів, інтеграцію зі сторонніми сервісами. Це підтверджує, що розроблена система має не лише завершене практичне значення, а й потенціал для подальшого розвитку.

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		104

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Munzner T. Visualization Analysis and Design. Boca Raton: CRC Press, 2014. 428 p. URL: <https://www.routledge.com/Visualization-Analysis-and-Design/Munzner/p/book/9781466508910> (дата звернення: 12.03.2026).
2. Card S. K., Mackinlay J. D., Shneiderman B. Readings in Information Visualization: Using Vision to Think. San Francisco: Morgan Kaufmann, 1999. 686 p. URL: <https://dl.acm.org/doi/10.5555/300679> (дата звернення: 13.03.2026).
3. Ware C. Information Visualization: Perception for Design. 4th ed. Cambridge: Morgan Kaufmann, 2020. 560 p. URL: <https://shop.elsevier.com/books/information-visualization/ware/978-0-12-812875-6> (дата звернення: 14.03.2026).
4. Di Battista G., Eades P., Tamassia R., Tollis I. G. Graph Drawing: Algorithms for the Visualization of Graphs. Upper Saddle River: Prentice Hall, 1999. 397 p. URL: <https://dl.acm.org/doi/10.5555/551884> (дата звернення: 18.03.2026).
5. Tamassia R. Handbook of Graph Drawing and Visualization. Boca Raton: CRC Press, 2013. 862 p. URL: <https://www.routledge.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/p/book/9781138034242> (дата звернення: 19.03.2026).
6. Sugiyama K., Tagawa S., Toda M. Methods for Visual Understanding of Hierarchical System Structures. IEEE Transactions on Systems, Man, and Cybernetics. 1981. Vol. 11, No. 2. P. 109–125. URL: <https://ieeexplore.ieee.org/document/4308636> (дата звернення: 21.03.2026).
7. 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. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102> (дата звернення: 20.03.2026).
8. Shneiderman B., Plaisant C., Cohen M., Jacobs S., Elmqvist N., Diakopoulos N. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th ed. Boston: Pearson, 2016. URL: <https://dl.acm.org/doi/10.5555/3033040> (дата звернення: 16.03.2026).

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		105

9. Norman D. The Design of Everyday Things: Revised and Expanded Edition. New York: Basic Books, 2013. 368 p. URL: https://books.google.com.ua/books/about/The_Design_of_Everyday_Things.html?id=qBfRDQAAQBAJ (дата звернення: 17.03.2026).
10. ISO/IEC 25010:2011. Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. URL: <https://www.iso.org/standard/35733.html> (дата звернення: 24.03.2026).
11. Family Tree Builder. MyHeritage. URL: <https://www.myheritage.com/FP/ftb-welcome.php> (дата звернення: 26.03.2026).
12. Time.Graphics. Free Online Timeline Maker. URL: <https://time.graphics/> (дата звернення: 27.03.2026).
13. Miro. Online Whiteboard for Realtime Collaboration. URL: <https://miro.com/online-whiteboard/> (дата звернення: 28.03.2026).
14. 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. URL: <https://dl.acm.org/doi/10.1109/TVCG.2011.185> (дата звернення: 30.03.2026).
15. OWASP Foundation. OWASP Top 10:2021. URL: <https://owasp.org/Top10/> (дата звернення: 06.04.2026).
16. JSON Web Token Introduction. Auth0. URL: <https://jwt.io/introduction> (дата звернення: 09.05.2026).
17. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (дата звернення: 02.04.2026).
18. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. Sebastopol: O'Reilly Media, 2013. 406 p. URL: <https://www.oreilly.com/library/view/restful-web-apis/9781449359713/> (дата звернення: 04.04.2026).

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

19. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p. URL: https://books.google.com/books/about/Clean_Architecture.html?id=fwvetAEACAAJ (дата звернення: 08.04.2026).
20. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p. URL: <https://martinfowler.com/books/ea.html> (дата звернення: 10.04.2026).
21. React Documentation. React: The Library for Web and Native User Interfaces. URL: <https://react.dev/> (дата звернення: 14.04.2026).
22. TypeScript Documentation. TypeScript: JavaScript with Syntax for Types. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 15.04.2026).
23. React Flow Documentation. React Flow: Node-Based UIs in React. URL: <https://reactflow.dev/> (дата звернення: 17.04.2026).
24. PostgreSQL Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/> (дата звернення: 23.04.2026).
25. Prisma ORM Documentation. Type-Safe ORM for Node.js and TypeScript. URL: <https://www.prisma.io/docs> (дата звернення: 25.04.2026).
26. NestJS Documentation. A Progressive Node.js Framework. URL: <https://docs.nestjs.com/> (дата звернення: 20.04.2026).
27. MDN Web Docs. WebSocket API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 28.04.2026).
28. Electron Documentation. Build Cross-Platform Desktop Apps with JavaScript, HTML, and CSS. URL: <https://www.electronjs.org/docs/latest/> (дата звернення: 03.05.2026).
29. SQLite Documentation. SQLite: SQL Database Engine. URL: <https://www.sqlite.org/docs.html> (дата звернення: 05.05.2026).
30. Docker Documentation. Docker Compose. URL: <https://docs.docker.com/compose/> (дата звернення: 07.05.2026).

31. Fett D., Küsters R., Schmitz G. A Comprehensive Formal Security Analysis of OAuth 2.0. arXiv preprint arXiv:1601.01229. 2016. URL: <https://arxiv.org/abs/1601.01229> (дата звернення: 10.05.2026).

32. Postman Learning Center. Test your API using the Collection Runner. URL: <https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs> (дата звернення: 14.05.2026).

33. Chrome for Developers. Inspect network activity. URL: <https://developer.chrome.com/docs/devtools/network> (дата звернення: 15.05.2026).

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		108

ДОДАТКИ

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		109

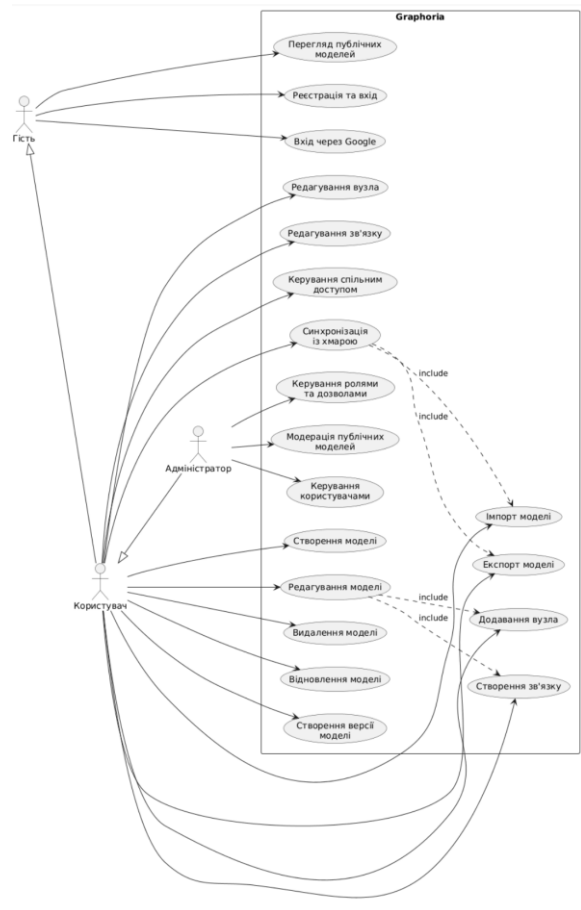


Рис. А.1. Діаграма варіантів використання системи

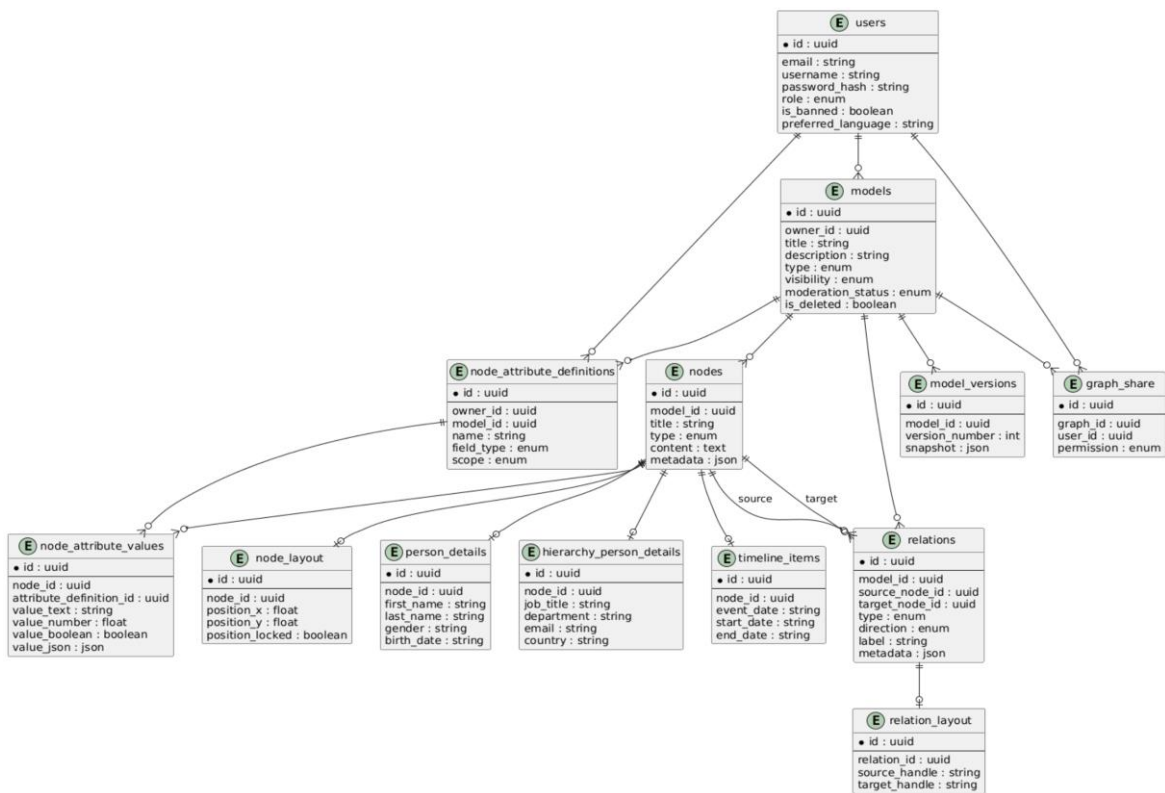


Рис. А.2. ER-діаграма структури даних системи

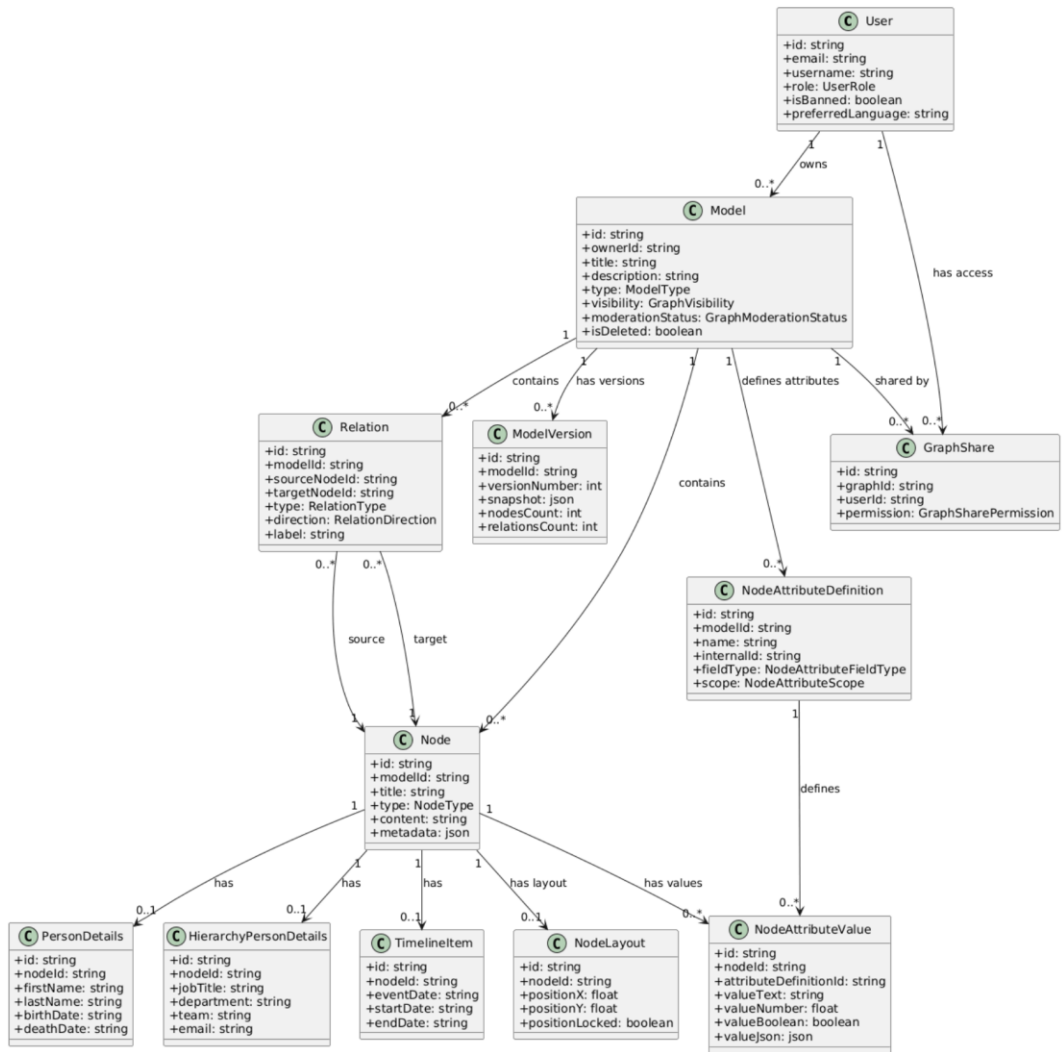


Рис. А.3. Діаграма класів основних сутностей системи

Гнучкий граф

Вільні об'єкти та зв'язки для будь-якої теми.

Сім'я та стосунки

Люди, родинні зв'язки та соціальні стосунки у вигляді мережі.

Оргструктура

Організаційні схеми лише з людьми та зв'язками підпорядкування.

Мережа зв'язків

Люди, об'єкти або групи з багатьма взаємними зв'язками.

Хронологія

Події та допоміжні об'єкти, впорядковані в часі.

Мапа залежностей

Компоненти та документи, пов'язані залежностями.

Організаційні схеми лише з людьми та зв'язками підпорядкування.

Доступ

Приватний

Лише ви можете відкрити цей граф.

Спільний

Запрошені користувачі можуть його відкрити.

Публічний

Видимий у каталозі після схвалення.

Назва

Організаційна структура Житомирської політехніки

Опис

Ієрархічна модель керівництва, факультетів, кафедр та викладачів, сформована на основі відкритих сторінок ztu.edu.ua. Для ФІКТ додано повні списки викладачів із доступних сторінок кафедр; для інших факультетів додано рівень керівництва факультетів і кафедр.

Скасувати

Створити модель

Рис. А.4. Створення нової моделі через форму інтерфейсу

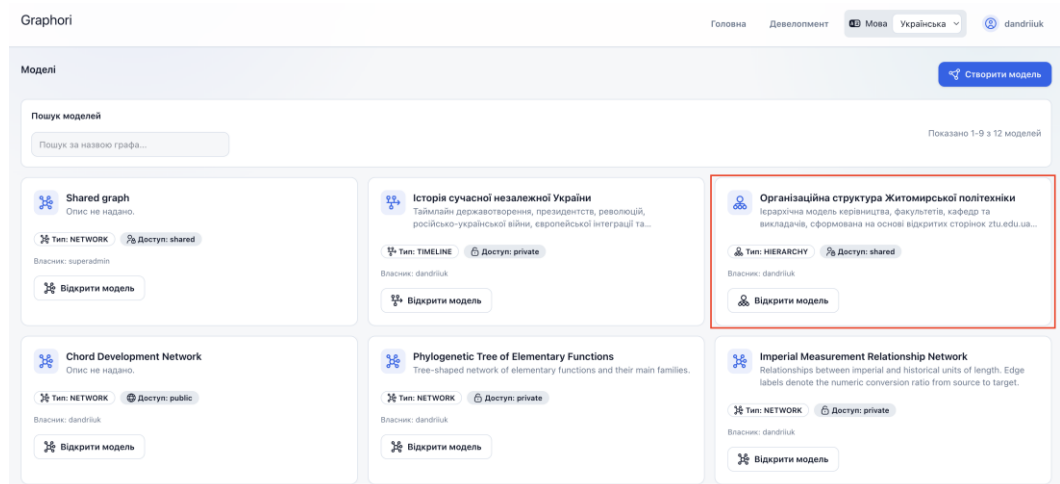


Рис. А.5. Створена модель у списку усіх моделей

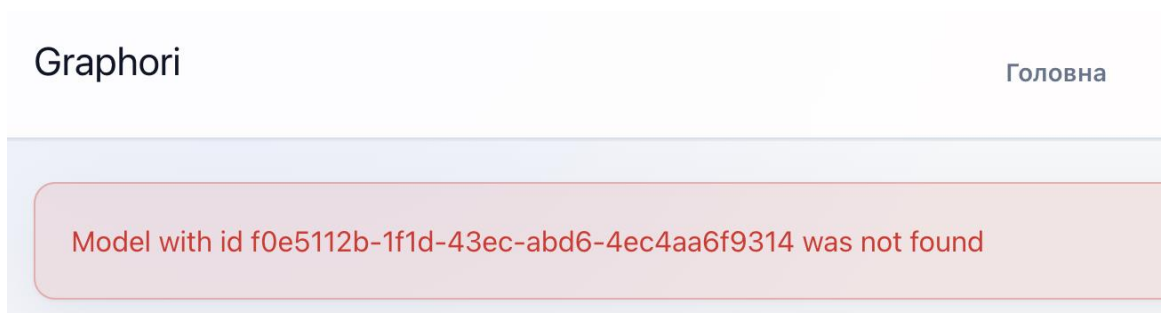


Рис. А.6. Результат відкриття моделі після «soft видалення»

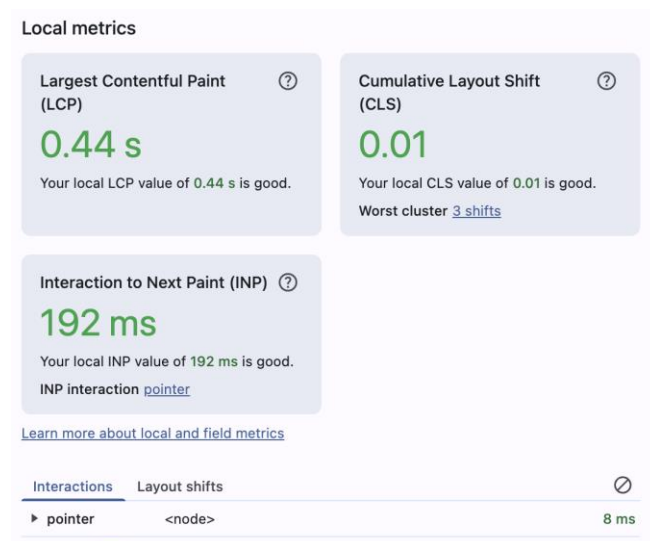


Рис. А.7. Метрики завантаження сторінки графу з великою кількістю елементів

Основні варіанти використання системи

Варіант використання	Основний актор	Результат виконання
Реєстрація користувача	Гість	Створено новий обліковий запис
Вхід до системи	Гість	Користувач отримує доступ до власних моделей
Вхід через Google	Гість	Користувач авторизується через зовнішній обліковий запис
Перегляд публічних моделей	Гість, користувач	Відображається модель із публічним доступом
Створення моделі	Авторизований користувач	У системі створюється нова інформаційна модель
Вибір типу моделі	Авторизований користувач	Визначається спосіб подальшої візуалізації та редагування
Редагування моделі	Авторизований користувач	Оновлюється структура моделі
Додавання вузла	Авторизований користувач	У моделі створюється новий елемент
Редагування вузла	Авторизований користувач	Змінюються властивості елемента моделі
Створення зв'язку	Авторизований користувач	Між вузлами формується відношення
Редагування зв'язку	Авторизований користувач	Змінюються параметри відношення між вузлами
Імпорт моделі	Авторизований користувач	Дані моделі завантажуються у систему
Експорт моделі	Авторизований користувач	Модель зберігається у файл
Створення версії моделі	Авторизований користувач	Фіксується поточний стан моделі
Відновлення моделі	Авторизований користувач	Повертається попередньо видалена або збережена версія
Керування спільним доступом	Власник моделі	Іншим користувачам надається право перегляду або редагування
Локальне збереження моделі	Користувач десктопної версії	Модель зберігається на пристрої користувача

продовж. табл. Б.1

Синхронізація моделі	Користувач десктопної версії	Локальні дані узгоджуються із сервером
Модерація публічних моделей	Адміністратор	Публічна модель підтверджується або відхиляється
Керування користувачами	Адміністратор	Облікові записи можуть бути обмежені або відновлені
Керування адміністраторами	Суперадміністратор	Призначаються ролі та адміністративні дозволи

Таблиця Б.2.

Основні атрибути сутностей

Сутність	Основні атрибути	Призначення
User	id, email, username, passwordHash, role, isBanned, preferredLanguage, createdAt	Зберігання облікових записів користувачів, ролей і персональних налаштувань
UserOAuthAccount	id, userId, provider, providerAccountId, email, displayName, avatarUrl	Зберігання прив'язаних OAuth-акаунтів користувача
Model	id, ownerId, title, description, type, visibility, moderationStatus, isDeleted, createdAt	Представлення окремої інформаційної моделі
ModelVersion	id, modelId, versionNumber, snapshot, nodesCount, relationsCount, createdById	Зберігання версій і знімків стану моделі
ModelDisplaySettings	id, modelId, showType, showDetails, showImages, autoLayout, nodeWidthMode	Зберігання налаштувань відображення моделі
TimelineSettings	id, modelId, timelineType, startValue, endValue, defaultZoomLevel	Зберігання глобальних параметрів часової лінії
Node	id, modelId, title, type, content, metadata, positionX, positionY	Представлення окремого елемента моделі

продовж. табл. Б.2

NodeLayout	id, nodeId, positionX, positionY, positionLocked	Зберігання координат і параметрів розміщення вузла
PersonDetails	id, nodeId, firstName, lastName, gender, birthDate, deathDate, biography	Зберігання персональних даних для генеалогічних моделей
HierarchyPersonDetails	id, nodeId, jobTitle, department, team, email, country, city	Зберігання даних працівників для організаційної ієрархії
TimelineItem	id, nodeId, eventDate, startDate, endDate, rangeType, periodId, color	Зберігання часових параметрів подій і періодів
Relation	id, modelId, sourceNodeId, targetNodeId, type, direction, label, metadata	Представлення зв'язку між двома вузлами
RelationLayout	id, relationId, sourceHandle, targetHandle	Зберігання параметрів візуального підключення зв'язку
GraphShare	id, graphId, userId, permission, createdAt	Зберігання правил спільного доступу до моделей
AdminPermission	id, userId, permission, createdAt	Зберігання адміністративних дозволів користувачів
NodeAttributeDefinition	id, ownerId, modelId, name, internalId, fieldType, scope, isRequired	Опис системних і користувацьких атрибутів вузлів
NodeAttributeValue	id, nodeId, attributeDefinitionId, valueText, valueNumber, valueBoolean, valueDate, valueJson	Зберігання значень атрибутів для конкретних вузлів

Деталізація ключових сценаріїв використання системи

Сценарій	Передумова	Основна послідовність дій	Результат
Реєстрація користувача	Користувач не має облікового запису	Користувач вводить реєстраційні дані, система перевіряє їх коректність і створює обліковий запис	Створено нового користувача
Вхід до системи	Користувач має обліковий запис	Користувач вводить email або username і пароль, система перевіряє дані та видає токен доступу	Користувач авторизований
Створення моделі	Користувач авторизований	Користувач відкриває форму створення, вводить назву, обирає тип моделі та підтверджує створення	У системі створено нову модель
Додавання вузла	Відкрита модель із правом редагування	Користувач додає вузол, заповнює властивості, система зберігає його в базі даних	Вузол з'являється в моделі
Створення зв'язку	У моделі є щонайменше два вузли	Користувач обирає початковий і кінцевий вузол, задає тип зв'язку, система перевіряє допустимість дії	Зв'язок створено та відображено
Редагування властивостей	Вибрано вузол або зв'язок	Користувач змінює атрибути елемента, система оновлює дані	Властивості елемента змінено
Імпорт моделі	Користувач має файл моделі	Користувач завантажує файл, система перевіряє структуру та створює модель	Модель імпортовано
Експорт моделі	Модель існує в системі	Користувач обирає формат експорту, система формує файл	Модель збережено у файл
Надання спільного доступу	Користувач є власником моделі	Користувач обирає іншого користувача та рівень доступу	Іншому користувачу надано доступ
Модерація моделі	Користувач має адміністративні права	Адміністратор переглядає публічну модель і змінює її статус	Модель схвалено або відхилено

продовж. табл. Б.3.

Локальне збереження	Користувач працює в desktop-версії	Користувач зберігає модель на пристрої	Модель доступна локально
Синхронізація моделі	Є локальна або хмарна версія моделі	Користувач запускає синхронізацію, система порівнює та передає дані	Дані узгоджено між desktop і backend

Таблиця Б.4.

Відповідність сценаріїв використання компонентам системи

Сценарій використання	Основні дії користувача	Компоненти системи, що беруть участь
Реєстрація та вхід до системи	Створення облікового запису, введення облікових даних	Клієнтський інтерфейс, модуль автентифікації, REST API, база даних
Перегляд доступних моделей	Відкриття головної сторінки, перегляд власних, спільних або публічних моделей	Клієнтський інтерфейс, підсистема керування моделями, REST API, PostgreSQL
Робота з різними типами моделей	Побудова мережі, генеалогії, таймлайну, ієрархії або структури залежностей	Візуальний редактор, layout-модулі, спеціалізовані параметри вузлів і зв'язків
Збереження та версіонування	Збереження поточного стану моделі, створення версії, відновлення попереднього стану	Підсистема керування моделями, ModelVersion, Prisma ORM, PostgreSQL
Імпорт та експорт моделей	Завантаження моделі з файлу, збереження структури у зовнішньому форматі	Клієнтський інтерфейс, REST API, сервіси імпорту й експорту, база даних
Керування спільним доступом	Надання прав перегляду або редагування іншим користувачам	Підсистема спільного доступу, GraphShare, REST API, PostgreSQL
Модерація публічних моделей	Перегляд моделей, схвалення або відхилення публікації	Адміністративна панель, модуль адміністрування, модуль дозволів, PostgreSQL

продовж. табл. Б.4

Локальна робота в десктопній версії	Створення, редагування та збереження моделей на пристрої користувача	Electron-застосунок, SQLite, preload API, IPC-механізм
Синхронізація десктопної версії з сервером	Передавання локальної моделі на сервер або оновлення локальної копії	Sync Service, REST API, SQLite, backend, PostgreSQL

методи класу GraphLayoutService:

```

solve(input: SolveGraphLayoutDto): GraphLayoutResponseDto {
    const normalized = this.normalizeInput(input);
    const components = this.buildComponents(normalized.nodes, normalized.edges);
    const options = normalized.options;
    const initialPositions = this.buildInitialPositions(
        normalized.nodes,
        normalized.edges,
        options,
        components,
    );
    let bestPositions = initialPositions;
    let bestMetrics = this.measureLayout(
        normalized.nodes,
        normalized.edges,
        bestPositions,
        components.length,
        options.iterations,
    );

    for (
        let restartIndex = 0;
        restartIndex < options.restarts;
        restartIndex += 1
    ) {
        const variantPositions = this.createVariantPositions(
            normalized.nodes,
            initialPositions,
            restartIndex,
            options,
        );
        const relaxed = this.relaxLayout(
            normalized.nodes,
            normalized.edges,
            variantPositions,
            components,
            options,
        );
        const refined = this.optimizeLayoutLocally(
            normalized.nodes,
            normalized.edges,
            relaxed,
            components.length,
            options,
        );
        const packed = this.packComponents(
            refined,
            normalized.nodes,
            components,
            options,
        );
        const metrics = this.measureLayout(
            normalized.nodes,
            normalized.edges,
            packed,
            components.length,
            options.iterations,
        );

        if (metrics.score < bestMetrics.score) {

```

		Андріюк Д. Р.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		119

```

        bestPositions = packed;
        bestMetrics = metrics;
    }
}

return new GraphLayoutResponseDto(
    normalized.nodes.map(
        (node) =>
            new GraphLayoutNodePositionDto(
                node.id,
                Math.round((bestPositions.get(node.id)?.x ?? 0) * 100) / 100,
                Math.round((bestPositions.get(node.id)?.y ?? 0) * 100) / 100,
            ),
    ),
    new GraphLayoutMetricsDto(
        bestMetrics.edgeCrossings,
        bestMetrics.nodeOverlaps,
        Math.round(bestMetrics.totalEdgeLength * 100) / 100,
        Math.round(bestMetrics.density * 1000) / 1000,
        bestMetrics.componentCount,
        bestMetrics.iterations,
    ),
);
}

private normalizeInput(input: SolveGraphLayoutDto): {
    nodes: LayoutNode[];
    edges: LayoutEdge[];
    options: SolveOptions;
}{
    const options = this.normalizeOptions(input.options);
    const nodes = input.nodes.map((node) => ({
        id: node.id,
        x: this.toFiniteNumber(node.x),
        y: this.toFiniteNumber(node.y),
        width: this.clamp(
            this.toFiniteNumber(node.width) ?? DEFAULT_NODE_WIDTH,
            60,
            480,
        ),
        height: this.clamp(
            this.toFiniteNumber(node.height) ?? DEFAULT_NODE_HEIGHT,
            48,
            240,
        ),
        locked: Boolean(node.locked),
        degree: 0,
    }));
    const nodeIds = new Set(nodes.map((node) => node.id));
    const edges = input.edges
        .filter(
            (edge) =>
                nodeIds.has(edge.sourceId) &&
                nodeIds.has(edge.targetId) &&
                edge.sourceId !== edge.targetId,
        )
        .map((edge, index) => ({
            id: edge.id ?? `edge-${index}`,
            sourceId: edge.sourceId,
            targetId: edge.targetId,
            weight: this.clamp(this.toFiniteNumber(edge.weight) ?? 1, 0.2, 5),
            directed: Boolean(edge.directed),
        }));
}

```

		Андріюк Д. Р.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		120

```

const degreeByNode = new Map<string, number>();

edges.forEach((edge) => {
  degreeByNode.set(
    edge.sourceId,
    (degreeByNode.get(edge.sourceId) ?? 0) + 1,
  );
  degreeByNode.set(
    edge.targetId,
    (degreeByNode.get(edge.targetId) ?? 0) + 1,
  );
});
nodes.forEach((node) => {
  node.degree = degreeByNode.get(node.id) ?? 0;
});

return { nodes, edges, options };
}

private measureLayout(
  nodes: LayoutNode[],
  edges: LayoutEdge[],
  positions: Map<string, Position>,
  componentCount: number,
  iterations: number,
): LayoutMetrics {
  const edgeCrossings = this.countEdgeCrossings(edges, positions);
  const nodeOverlaps = this.countNodeOverlaps(nodes, positions);
  const totalEdgeLength = edges.reduce((total, edge) => {
    const source = positions.get(edge.sourceId);
    const target = positions.get(edge.targetId);

    if (!source || !target) {
      return total;
    }

    return total + Math.hypot(target.x - source.x, target.y - source.y);
  }, 0);
  const bounds = this.getOverallBounds(nodes, positions);
  const density =
    nodes.length === 0
      ? 0
      : (nodes.length * DEFAULT_NODE_WIDTH * DEFAULT_NODE_HEIGHT) /
        Math.max(1, bounds.width * bounds.height);
  const score =
    edgeCrossings * 18000 +
    nodeOverlaps * 36000 +
    totalEdgeLength * 0.1 +
    density * 5200;

  return {
    edgeCrossings,
    nodeOverlaps,
    totalEdgeLength,
    density,
    componentCount,
    iterations,
    score,
  };
}

```

```

private createVariantPositions(
  nodes: LayoutNode[],
  initial: Map<string, Position>,
  variant: number,
  options: SolveOptions,
): Map<string, MutablePosition> {
  const positions = new Map<string, MutablePosition>();
  const center = this.getPositionMapCenter(initial);
  const rotation = variant === 0 ? 0 : ((variant % 4) * Math.PI) / 2;
  const mirrorX = variant > 0 && variant % 2 === 0 ? -1 : 1;
  const mirrorY = variant > 0 && variant % 3 === 0 ? -1 : 1;

  nodes.forEach((node, index) => {
    const base = initial.get(node.id) ?? { x: index * 24, y: index * 12 };
    const jitter =
      variant === 0
        ? { x: 0, y: 0 }
        : this.variantNoise(node.id, variant, index);
    const amplitude =
      variant === 0 ? 0 : Math.max(42, options.edgeLength * 0.34);
    const relativeX = (base.x - center.x) * mirrorX;
    const relativeY = (base.y - center.y) * mirrorY;
    const rotatedBase =
      variant === 0
        ? base
        : {
            x:
              center.x +
              relativeX * Math.cos(rotation) -
              relativeY * Math.sin(rotation),
            y:
              center.y +
              relativeX * Math.sin(rotation) +
              relativeY * Math.cos(rotation),
          };

    positions.set(node.id, {
      x: rotatedBase.x + jitter.x * amplitude,
      y: rotatedBase.y + jitter.y * amplitude,
      vx: 0,
      vy: 0,
    });
  });
}

private measureLayout(
  nodes: LayoutNode[],
  edges: LayoutEdge[],
  positions: Map<string, Position>,
  componentCount: number,
  iterations: number,
): LayoutMetrics {
  const edgeCrossings = this.countEdgeCrossings(edges, positions);
  const nodeOverlaps = this.countNodeOverlaps(nodes, positions);
  const totalEdgeLength = edges.reduce((total, edge) => {
    const source = positions.get(edge.sourceId);
    const target = positions.get(edge.targetId);

    if (!source || !target) {
      return total;
    }

    return total + Math.hypot(target.x - source.x, target.y - source.y);
  }, 0);
}

```

```

const bounds = this.getOverallBounds(nodes, positions);
const density =
  nodes.length === 0
    ? 0
    : (nodes.length * DEFAULT_NODE_WIDTH * DEFAULT_NODE_HEIGHT) /
      Math.max(1, bounds.width * bounds.height);
const score =
  edgeCrossings * 18000 +
  nodeOverlaps * 36000 +
  totalEdgeLength * 0.1 +
  density * 5200;

return {
  edgeCrossings,
  nodeOverlaps,
  totalEdgeLength,
  density,
  componentCount,
  iterations,
  score,
};
}

```

Додаток Г

метод normalizeAttributeValue класу NodeAttributesService:

```

private normalizeAttributeValue(
  fieldType: NodeAttributeFieldType,
  value: unknown,
  options: string[] | null,
): NormalizedAttributeValue | null {
  if (value === null || value === undefined || value === "") {
    return null;
  }

  if (
    fieldType === NodeAttributeFieldType.TEXT ||
    fieldType === NodeAttributeFieldType.TEXTAREA
  ) {
    if (typeof value !== 'string') {
      throw new BadRequestException('Attribute value must be a string');
    }

    return { valueText: value.trim() };
  }

  if (fieldType === NodeAttributeFieldType.PHONE) {
    if (typeof value !== 'string' || !this.phonePattern.test(value.trim())) {
      throw new BadRequestException(
        'Phone attribute must be a valid phone-like string',
      );
    }

    return { valueText: value.trim() };
  }

  if (fieldType === NodeAttributeFieldType.EMAIL) {
    if (typeof value !== 'string' || !this.emailPattern.test(value.trim())) {
      throw new BadRequestException(
        'Email attribute must be a valid email address',
      );
    }
  }
}

```

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		123

```

    }

    return { valueText: value.trim() };
}

if (fieldType === NodeAttributeFieldType.URL) {
    if (typeof value !== 'string') {
        throw new BadRequestException('URL attribute must be a string');
    }

    try {
        const nextUrl = new URL(value.trim());
        return { valueText: nextUrl.toString() };
    } catch {
        throw new BadRequestException('URL attribute must be a valid URL');
    }
}

if (fieldType === NodeAttributeFieldType.NUMBER) {
    const nextValue = typeof value === 'number' ? value : Number(value);

    if (!Number.isFinite(nextValue)) {
        throw new BadRequestException(
            'Number attribute must be a valid number',
        );
    }

    return { valueNumber: nextValue };
}

if (fieldType === NodeAttributeFieldType.DATE) {
    if (
        typeof value !== 'string' &&
        typeof value !== 'number' &&
        !(value instanceof Date)
    ) {
        throw new BadRequestException('Date attribute must be a valid date');
    }

    const nextDate = value instanceof Date ? value : new Date(value);

    if (Number.isNaN(nextDate.getTime())) {
        throw new BadRequestException('Date attribute must be a valid date');
    }

    return { valueDate: nextDate };
}

if (fieldType === NodeAttributeFieldType.BOOLEAN) {
    if (typeof value !== 'boolean') {
        throw new BadRequestException(
            'Boolean attribute must be true or false',
        );
    }

    return { valueBoolean: value };
}

if (fieldType === NodeAttributeFieldType.SELECT) {
    if (typeof value !== 'string') {
        throw new BadRequestException('Select attribute must be a string');
    }
}

```

```

if (options && !options.includes(value)) {
  throw new BadRequestException(
    'Select attribute value is not in the allowed options',
  );
}

return { valueText: value };
}

if (fieldType === NodeAttributeFieldType.MULTI_SELECT) {
  if (
    !Array.isArray(value) ||
    value.some((item) => typeof item !== 'string')
  ) {
    throw new BadRequestException(
      'Multi-select attribute must be an array of strings',
    );
  }

  const nextValues = value
    .map((item: string) => item.trim())
    .filter(Boolean);

  if (options && nextValues.some((item) => !options.includes(item))) {
    throw new BadRequestException(
      'Multi-select attribute value is not in the allowed options',
    );
  }

  return { valueJson: nextValues };
}
throw new BadRequestException('Unsupported attribute field type');
}

```

Конфігурація файлу docker-compose.yml:

```

services:
  postgres:
    image: postgres:15
    container_name: Graphori-postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5432:5432"
    volumes:
      - Graphori_postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 5s
      timeout: 5s
      retries: 10

  api:
    build:
      context: ./apps/api
      dockerfile: Dockerfile
      target: development
    container_name: Graphori-api
    restart: unless-stopped
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      DATABASE_URL: ${DATABASE_URL}
      JWT_SECRET: ${JWT_SECRET}
      PORT: 3000
      CHOKIDAR_USEPOLLING: true
    ports:
      - "3000:3000"
    command: sh -c "npx prisma migrate deploy && npm run start:dev"
    develop:
      watch:
        - action: sync
          path: ./apps/api/src
          target: /app/src
          initial_sync: true

        - action: sync
          path: ./apps/api/prisma
          target: /app/prisma
          initial_sync: true

        - action: sync
          path: ./apps/api/prisma.config.ts
          target: /app/prisma.config.ts
          initial_sync: true

        - action: rebuild
          path: ./apps/api/package.json

        - action: rebuild
          path: ./apps/api/package-lock.json

```

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Левківський В.Л.				126
Змн.	Арк.	№ докум.	Підпис	Дата		

```

- action: rebuild
  path: ./apps/api/Dockerfile

web:
  build:
    context: ./apps/web
    dockerfile: Dockerfile
    target: development
  container_name: Graphori-web
  restart: unless-stopped
  depends_on:
    - api
  environment:
    VITE_API_BASE_URL: ${VITE_API_BASE_URL}
    CHOKIDAR_USEPOLLING: true
  ports:
    - "5173:5173"
  command: npm run dev -- --host 0.0.0.0
  develop:
    watch:
      - action: sync
        path: ./apps/web/src
        target: /app/src
        initial_sync: true

      - action: sync
        path: ./apps/web/public
        target: /app/public
        initial_sync: true

      - action: sync
        path: ./apps/web/index.html
        target: /app/index.html
        initial_sync: true

      - action: rebuild
        path: ./apps/web/package.json

      - action: rebuild
        path: ./apps/web/package-lock.json

      - action: rebuild
        path: ./apps/web/vite.config.ts

      - action: rebuild
        path: ./apps/web/Dockerfile

volumes:
  Graphori_postgres_data:

```

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		127

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	Graphoria Local Environment	1	942ms	38	0	10 ms

All 38Passed 38Failed 0Skipped 0Errors 0Console log

ListGrid

Iteration 1

GETHealth / Smoke > API smoke check

http://192.168.10.123:3000/api/v1/auth/me

200 • 43 ms • 678 B 1

PASSAPI responds with either user data or auth error

POSTModels > Create network model

http://192.168.10.123:3000/api/v1/models

201 • 34 ms • 780 B 2

PASSStatus code is 200 or 201

PASSResponse contains model id

GETModels > List my models

http://192.168.10.123:3000/api/v1/models

200 • 10 ms • 9.539 KB 2

PASSStatus code is successful

PASSResponse is JSON

GETModels > Get model by id

http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098

200 • 8 ms • 775 B 1

PASSStatus code is successful

PATCHModels > Update model metadata

http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098

200 • 10 ms • 769 B 1

PASSStatus code is successful

POSTModels > Create model version

http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098/versions

201 • 33 ms • 616 B 1

PASSVersion endpoint returns success or is not implemented yet

GETModels > Export model as JSON

http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098/export?format=json

404 • 2 ms • 407 B 1

PASSExport endpoint returns success or is not implemented yet

POSTNodes > Create first node

http://192.168.10.123:3000/api/v1/nodes

201 • 13 ms • 595 B 2

PASSStatus code is 200 or 201

PASSResponse contains node id

Рис. 1. Результати тестів API системи у Postman

POST	Nodes > Create second node	http://192.168.10.123:3000/api/v1/nodes	201	11 ms	588 B	2
PASS	Status code is 200 or 201					
PASS	Response contains node id					
GET	Nodes > Get first node	http://192.168.10.123:3000/api/v1/nodes/ec05b1c4-2c3b-4b9c-9955-ac1b5e6122c2	200	6 ms	590 B	1
PASS	Node endpoint returns success or is not implemented yet					
PATCH	Nodes > Update first node	http://192.168.10.123:3000/api/v1/nodes/ec05b1c4-2c3b-4b9c-9955-ac1b5e6122c2	200	12 ms	598 B	1
PASS	Status code is successful					
POST	Relations > Create relation	http://192.168.10.123:3000/api/v1/relations	201	12 ms	594 B	2
PASS	Status code is 200 or 201					
PASS	Response contains relation id					
PATCH	Relations > Update relation	http://192.168.10.123:3000/api/v1/relations/d6951462-3257-4e77-93f7-41c3b56d090c	200	11 ms	603 B	1
PASS	Status code is successful					
GET	Relations > Get model after relation changes	http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098	200	5 ms	769 B	1
PASS	Status code is successful					
PATCH	Sharing and Visibility > Make model public / pending moderation	http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098	200	11 ms	768 B	1
PASS	Status code is successful					
GET	Sharing and Visibility > List public models	http://192.168.10.123:3000/api/v1/models	200	5 ms	1.961 KB	1
PASS	Public models endpoint responds					
POST	Sharing and Visibility > Share model with user	http://192.168.10.123:3000/api/v1/sharing	404	2 ms	352 B	1
PASS	Sharing endpoint returns expected result or needs path adjustment					

Рис. 2. Результати тестів API системи у Postman

		Андріюк Д. Р.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Левківський В.Л.				
Змн.	Арк.	№ докум.	Підпис	Дата		129

POST	Sharing and Visibility > Share model with edit permission	404	2 ms	352 B	1
	http://192.168.10.123:3000/api/v1/sharing				
PASS	Sharing endpoint returns expected result or needs path adjustment				
GET	Admin > Admin - list users	403	4 ms	343 B	1
	http://192.168.10.123:3000/api/v1/admin/users				
PASS	Admin route is either accessible for admin or forbidden for normal user				
GET	Admin > Admin - list public graphs for moderation	403	3 ms	343 B	1
	http://192.168.10.123:3000/api/v1/admin/graphs/public				
PASS	Admin moderation route is either accessible for admin or forbidden for normal user				
PATCH	Admin > Admin - approve model	403	4 ms	343 B	1
	http://192.168.10.123:3000/api/v1/admin/graphs/7b278e0d-e11a-4dc1-9de9-c746d409b098/moderation				
PASS	Admin moderation update is either successful or forbidden for normal user				
GET	Admin > Super admin - list admins	403	4 ms	343 B	1
	http://192.168.10.123:3000/api/v1/admin/admins				
PASS	Super admin route is either accessible or forbidden				
GET	Negative and Critical Scenarios > No token - get current user	401	2 ms	320 B	1
	http://192.168.10.123:3000/api/v1/auth/me				
PASS	Request is rejected				
GET	Negative and Critical Scenarios > Invalid token - list models	401	2 ms	359 B	1
	http://192.168.10.123:3000/api/v1/models				
PASS	Request is rejected				
GET	Negative and Critical Scenarios > Forbidden - get unknown/private model	404	5 ms	390 B	1
	http://192.168.10.123:3000/api/v1/models/00000000-0000-0000-0000-000000000000				
PASS	Request is rejected as invalid				
POST	Negative and Critical Scenarios > Invalid create model - empty title	400	5 ms	356 B	1
	http://192.168.10.123:3000/api/v1/models				
PASS	Request is rejected as invalid				

Рис. 3. Результати тестів API системи у Postman

POST	Negative and Critical Scenarios > Invalid create node - fake model id	http://192.168.10.123:3000/api/v1/nodes	404	4 ms	391 B	1
PASS	Request is rejected as invalid					
POST	Negative and Critical Scenarios > Invalid create relation - fake node ids	http://192.168.10.123:3000/api/v1/relations	404	7 ms	355 B	1
PASS	Request is rejected as invalid					
GET	Negative and Critical Scenarios > Forbidden admin API as regular user	http://192.168.10.123:3000/api/v1/admin/users	403	3 ms	343 B	1
PASS	Regular user must not access admin API					
DELETE	Cleanup > Delete relation	http://192.168.10.123:3000/api/v1/relations/d6951462-3257-4e77-93f7-41c3b56d090c	200	11 ms	359 B	1
PASS	Delete relation is successful or already deleted/not implemented					
DELETE	Cleanup > Delete first node	http://192.168.10.123:3000/api/v1/nodes/ec05b1c4-2c3b-4b9c-9955-ac1b5e6122c2	200	17 ms	355 B	1
PASS	Delete node is successful or already deleted/not implemented					
DELETE	Cleanup > Delete second node	http://192.168.10.123:3000/api/v1/nodes/cc4b0c0b-aaeb-4fe1-94d5-f6653fa8c7a3	200	15 ms	355 B	1
PASS	Delete node is successful or already deleted/not implemented					
DELETE	Cleanup > Delete model	http://192.168.10.123:3000/api/v1/models/7b278e0d-e11a-4dc1-9de9-c746d409b098	200	9 ms	350 B	1
PASS	Delete model is successful or soft-deleted/not implemented					

Рис. 4. Результати тестів API системи у Postman