Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ СЕРВІСУ ДЛЯ ПОБУДУВАННЯ СХЕМ І ЗВ'ЯЗКІВ БАЗИ ДАНИХ З ВИКОРИСТАННЯМ REACT, TYPESCRIPT ТА REACTFLOW

DEVELOPMENT AND RESEARCH OF A SERVICE FOR BUILDING DATABASE SCHEMAS AND RELATIONSHIPS USING REACT, TYPESCRIPT AND REACTFLOW

спеціальність 121 Інженерія програмного забезпечення
освітня програма «Інженерії програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Мельник С. В.

Керівник:
к.т.н., доцент
Повстяна Ю. С.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«__» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Мельнику Сергію Валерійовичу

1. Тема кваліфікаційної роботи: *Розробка та дослідження сервісу для побудування схем і зв'язків бази даних з використанням React, TypeScript та ReactFlow*

Керівник роботи: *Повстяна Ю.С. к.т.н., доцент*

затверджені наказом закладу вищої освіти від «29» *березня 2025 р. № 190/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«04» грудня 2025 р.*

3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *:аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи.*

5. Перелік графічного матеріалу 9 *рисунків, 4 таблиць, 9 лістингів коду.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Повстяна Ю. С.*		
Теоретичне дослідження та практична реалізація	*Повстяна Ю. С.*		
Експериментальне дослідження системи	*Повстяна Ю. С.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання 02.04.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Мельник С. В.

Керівник кваліфікаційної роботи _____ Повстяна Ю. С.

АНОТАЦІЯ

Мельник С. В. Розробка та дослідження сервісу для побудування схем і зв'язків бази даних з використанням React, TypeScript та ReactFlow. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел. У кваліфікаційній роботі магістра розглядається проблема розробки веб-сервісу для інтерактивного побудови схем і зв'язків бази даних. Робота включає теоретичне дослідження, вибір технологічного стеку, розробку архітектури та реалізацію веб-застосунку з використанням TypeScript, React та бібліотеки ReactFlow. Основна мета дослідження – створення інтерактивної візуалізації структур даних та забезпечення механізмів їх редагування в реальному часі.

У першому розділі роботи аналізуються існуючі методи та інструменти для побудови схем баз даних, вказуються їх обмеження та необхідність розробки нових, більш універсальних рішень. У другому розділі наведено розробку сервісу, обґрунтовано вибір технологій та описано процес реалізації. У третьому розділі проводиться експериментальне дослідження ефективності сервісу, аналізуються результати тестування, які підтверджують його високу продуктивність і надійність при роботі з великими обсягами даних.

Ключові слова: сервіс, схеми баз даних, візуалізація даних, React, TypeScript, ReactFlow, інтерактивні діаграми, SQL.

ABSTRACT

Melnyk S. V. Development and Research of a Service for Building Database Schemas and Relationships Using React, Typescript and Reactflow. Manuscript.

Master's Thesis of the specialty «Software Engineering». Lutsk National Technical University, Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices. This thesis addresses the problem of developing a web service for interactive construction of database schemas and relationships. The work includes theoretical research, selection of the technology stack, development of the architecture, and implementation of the web application using TypeScript, React, and the ReactFlow library. The main goal of the research is to create an interactive visualization of data structures and provide mechanisms for real-time editing.

The first chapter analyzes existing methods and tools for building database schemas, pointing out their limitations and the need for new, more versatile solutions. The second chapter presents the development of the service, justifies the choice of technologies, and describes the implementation process. In the third chapter, an experimental study of the service's efficiency is conducted, and the results of testing are analyzed, confirming its high performance and reliability when working with large amounts of data.

Keywords: service, database schemas, data visualization, React, TypeScript, ReactFlow, interactive diagrams, SQL.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна розробка програмного забезпечення нерозривно пов'язана з ефективним проєктуванням, структурним аналізом та візуалізацією баз даних. В умовах постійного зростання обсягів інформації, ускладнення міжсистемних взаємозв'язків і динамічного розвитку архітектурних підходів виникає необхідність створення нових засобів для інтерактивного моделювання, дослідження та представлення структур даних. Попри наявність різноманітних методологій та інструментів для опису моделей даних, спостерігається нестача універсальних, розширюваних і зручних сервісів, здатних забезпечити динамічну взаємодію між структурними елементами бази даних у веборієнтованому середовищі. Це обумовлює актуальність наукових досліджень, спрямованих на розробку інноваційних засобів для побудови схем і зв'язків баз даних із використанням сучасних технологій програмної інженерії.

Світові тенденції у галузі інформаційних технологій характеризуються переходом до інтерактивних моделей візуалізації даних, широким застосуванням фронтенд-фреймворків, що підтримують реактивну взаємодію користувача з об'єктами, а також інтеграцією типобезпечних мов програмування у процес розроблення програмних систем. Це забезпечує підвищення надійності, продуктивності та зручності у створенні складних інструментів для роботи з інформаційними структурами.

Актуальність теми підтверджується сучасними науковими дослідженнями, які наголошують на обмеженості традиційних підходів до візуалізації складних структур даних. Існуючі формалізми візуалізації даних обмежені вхідними даними з однієї таблиці, що робить існуючі граматики візуалізації, такі як Vega-lite або ggplot2, нудними у використанні, мають надмірно складні API та неефективними при візуалізації даних з кількох таблиць [1].

Метою кваліфікаційної роботи магістра є створення веб-сервісу для побудови та дослідження схем і зв'язків бази даних з використанням TypeScript та бібліотеки ReactFlow, який забезпечує інтерактивну візуалізацію структур даних, редагування елементів та збереження моделі в базі даних.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати сучасні інструменти, методи та бібліотеки для моделювання та візуалізації структур баз даних, визначити їхні функціональні особливості та недоліки;

– обгрунтувати вибір технологічного стеку (TypeScript, React, ReactFlow) та розробити архітектуру веб-сервісу для побудови схем і зв'язків бази даних;

– реалізувати алгоритм синтаксичного аналізу SQL-описів структур баз даних з визначенням таблиць, полів, первинних та зовнішніх ключів;

– створити інтерфейс користувача для інтерактивного відображення, редагування та управління схемами бази даних із використанням ReactFlow у веб-застосунку;

– забезпечити функціонал збереження, оновлення, імпорту та переключення між різними схемами даних з використанням системи станів;

– розробити методику та провести експериментальне дослідження сервісу, включаючи модульне тестування алгоритмів парсингу та оцінювання продуктивності при різних масштабах вихідних SQL-структур.

Об'єктом дослідження кваліфікаційної роботи магістра є процес візуального проєктування та моделювання структур баз даних у середовищі вебсервісів.

Предметом дослідження кваліфікаційної роботи магістра є методи та засоби інтерактивного побудування й відображення зв'язків бази даних із використанням TypeScript та ReactFlow у веборієнтованому середовищі.

Наукова новизна отриманих результатів. У роботі вперше запропоновано підхід до динамічної побудови та редагування схем баз даних

у браузері, який поєднує типобезпечну архітектуру на основі TypeScript з гнучкою візуальною моделлю ReactFlow.

Удосконалено підхід до візуалізації зв'язків між сутностями через розробку структури вузлів, що враховує атрибути, ключі та типи відношень. Подальшого розвитку набули методи інтерактивної взаємодії користувача зі схемою бази даних у реальному часі.

Практична цінність роботи. Розроблений вебсервіс може бути використаний у процесі навчання, проєктування та документування інформаційних систем, зокрема при підготовці ER-діаграм і технічної документації. Отримані результати можуть бути впроваджені у навчальний процес закладів вищої освіти, а також інтегровані в існуючі платформи розробки, що використовують сучасні фронтенд-технології. Розроблений прототип має високий ступінь готовності до подальшого розширення та впровадження у практичну діяльність розробників ПЗ.

Основні результати кваліфікаційної роботи магістра були представлені на X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», що відбулася 23-24 травня 2025 року в Луцьку. Матеріали конференції опубліковано в тезах доповідей [2].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Складність сучасних інформаційних систем безпосередньо корелює зі складністю їхньої підсистеми управління даними. Ефективне проєктування, документування та управління схемами реляційних баз даних (БД) є наріжним каменем успішної розробки програмного забезпечення. Предметна область цього дослідження охоплює взаємодію між теорією моделювання даних (зокрема, модель «сутність-зв'язок» (Entity-Relationship Model), введена Ченом [3]) та сучасними методами візуалізації та веб-розробки. Візуальне представлення логічної структури БД, тобто схеми, що містить таблиці (сутності) та зв'язки між ними, є необхідним інструментом для архітекторів, розробників і менеджерів [4].

Аналіз літературних джерел підтверджує важливість візуального кодування складної інформації для зниження когнітивного навантаження користувача [5]. Діаграми «вузол-ребро» (Node-Link Diagrams), які є основою для відображення схем БД, визнані найбільш ефективним способом представлення графових структур [6]. Технологічне підґрунтя роботи базується на використанні фреймворку React та мови TypeScript. Застосування TypeScriptє вимогою сучасної розробки високоякісних додатків, оскільки статична типізація забезпечує раннє виявлення помилок та підвищує надійність системи, що особливо важливо в контексті обробки структурованих метаданих БД [7]. Бібліотека ReactFlow широко використовується у сучасних node-based UI-структурах у React-екосистемі для створення інтерактивних, масштабованих та високопродуктивних діаграмних інтерфейсів, що робить її ідеальною основою для розробки сервісу [8].

Існуючі інструменти для моделювання БД можна умовно поділити на три основні категорії, кожна з яких має значні обмеження, що створюють основу для даного дослідження:

– спеціалізовані СУБД-орієнтовані інструменти (наприклад, MySQL Workbench, pgAdmin ERD Tool [9, 10]). Їхня головна перевага – глибока інтеграція з конкретними базами даних та можливість зворотного інжинірингу (генерації схеми з існуючої БД). Однак, вони часто є громіздкими, мають застарілий користувацький інтерфейс, обмежені у гнучкості візуалізації (стилі, кастомізація) і майже завжди прив'язані до конкретної СУБД, ускладнюючи роботу з гетерогенними середовищами. Крім того, їхня архітектура не завжди придатна для легкого вбудовування в сучасні веб-додатки як окремого сервісу;

– загальні інструменти для діаграм (draw.io, Lucidchart [11]). Вони пропонують високу гнучкість у малюванні, веб-доступність та зручні функції співпраці. Проте вони не мають вбудованої інтелектуальної логіки БД. Побудова схеми є ручною роботою, без автоматичної валідації зв'язків (наприклад, перевірки цілісності зовнішніх ключів) та без механізмів імпорту/експорту реальних метаданих, що перетворює їх на інструменти для «малювання», а не для «моделювання» [4].

Відкриті компоненти на основі React/ReactFlow. Існує низка прикладів та демонстрацій використання ReactFlow для ER-діаграм. Однак ці рішення, як правило, не виходять за рамки мінімально функціонального прототипу. Вони демонструють інтерактивність, але не містять важливого бізнес-шару: механізму підключення до БД, інтелектуального алгоритму автоматичного компонування (auto-layout) для складних схем (критично для схем із понад 30 таблицями) та стандартизованого механізму імпорту/експорту метаданих;

– спеціалізовані веб-сервіси (DB Schema Visualizer), які активно конкурують у сфері User Experience (UX) та швидкості роботи, використовуючи сучасні фронтенд-технології. До цієї категорії належать такі інструменти, як dbdiagram.io, drawsql.app та chartdb.io.

DBDiagram.io [12] представляє філософію «код-перший» (code-first), використовуючи власну декларативну мову DBML (Database Markup Language). Цей підхід є надзвичайно ефективним для швидкого визначення схеми та її документування, оскільки DBML є більш читабельним і незалежним від конкретної СУБД, ніж SQL DDL. Однак, такий метод має обмеження для користувачів, які віддають перевагу прямому візуальному маніпулюванню (drag-and-drop) та моделюванню без написання коду.

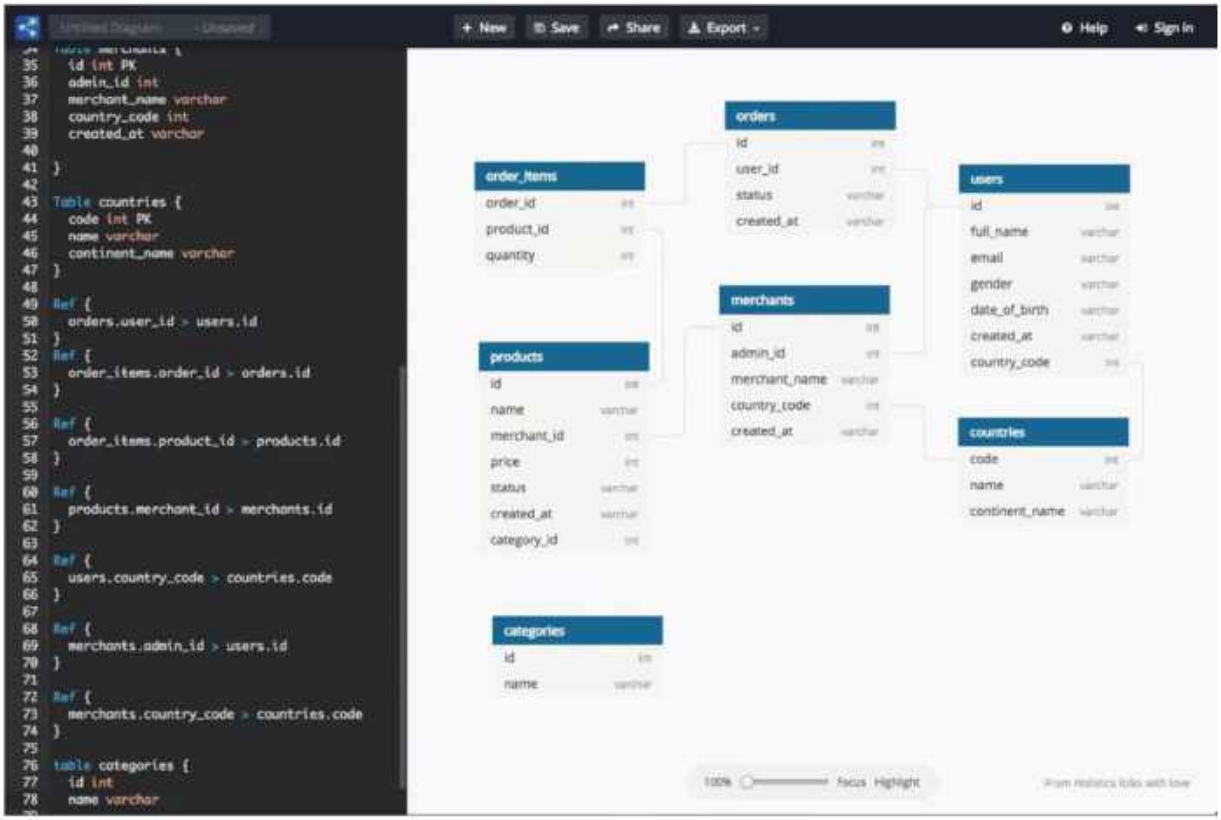

Рисунок 1.1 – Онлайн-інструмент DBDiagram.io [12]

DrawSQL.app [13] фокусується на візуальній привабливості («visually stunning») та командній співпраці (real-time collaboration). Інструмент пропонує шаблони, імпорт SQL-скриптів та, що критично важливо, функціонал автоматичного компонування (Auto Layout). Це вирішує проблему неефективного розміщення елементів на полотні для складних схем.

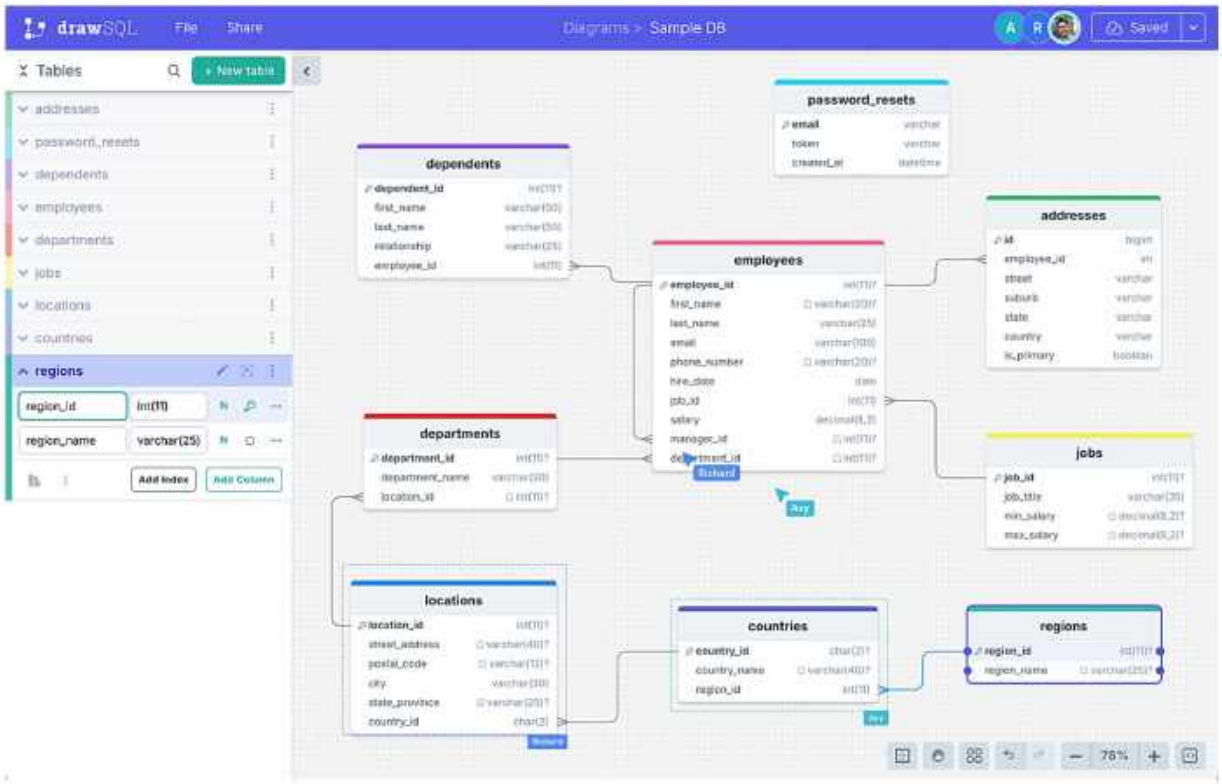

Рисунок 1.2 – Веб-сервіс DrawSQL.app [13]

ChartDB.io [14] позиціонує себе як Open-Source рішення, яке забезпечує миттєвий імпорт схеми через «Smart Query» (SQL або JSON) та пропонує інтерактивне редагування. Його переваги включають прозорість (відкритий код) та фокус на швидкій візуалізації, що є прямим конкурентом ідеї, закладеної в даній кваліфікаційній роботі.

Ці сучасні інструменти (DrawSQL, ChartDB, dbdiagram) успішно вирішили проблему UX та візуальної естетики, яка була слабкою стороною застарілих інструментів. Вони також реалізували механізми імпорту зі SQL, що частково долає проблему ручного внесення даних. Проте, їхня інтеграція в інші веб-додатки як вбудований компонент або бібліотека залишається обмеженою, часто через комерційну модель (freemium, як-от DrawSQL, dbdiagram Pro). Більшість функціоналу, такого як версіонування, розширене компонування та приватні діаграми, доступні лише у платних планах.

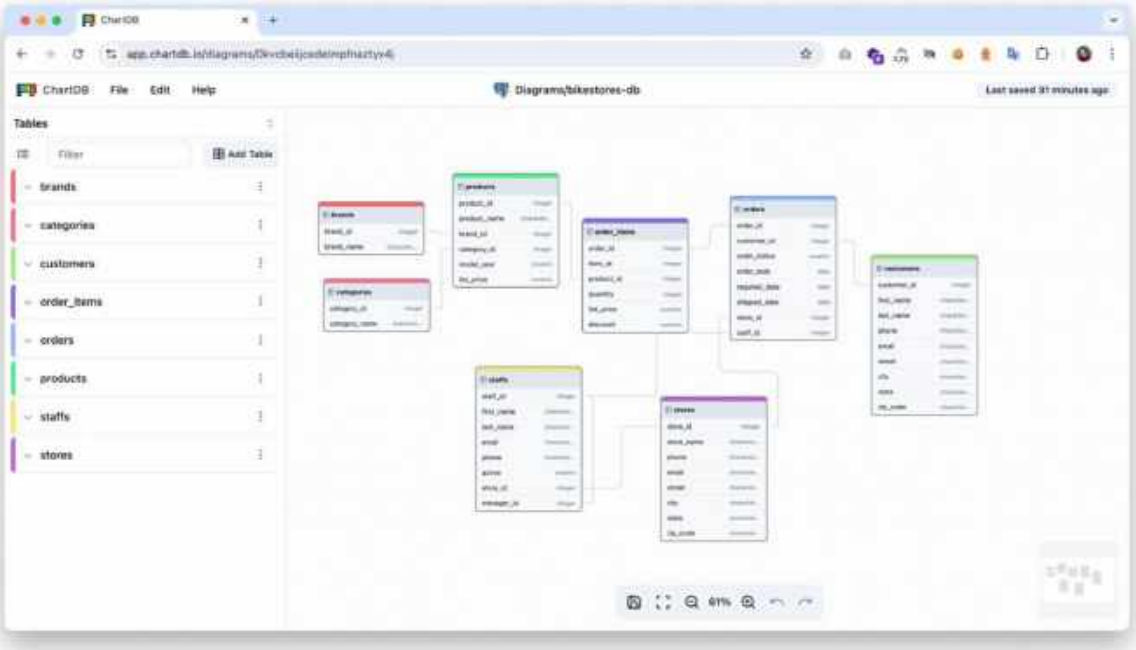

Рисунок 1.3 – Платформа ChartDB.io [14]

Відеоматеріали, як-от YouTube-демонстрації з розробки інструментів візуалізації [15], підтверджують зростаючий інтерес спільноти до створення власних, кастомізованих рішень на базі сучасних фреймворків. Це свідчить про практичну доцільність дослідження, спрямованого не просто на використання готового SaaS-продукту, а на розробку та дослідження самого сервісу зсередини, зокрема вивчення та оптимізацію його ключових компонентів – ReactFlow, TypeScript та алгоритмів компонування.

Проведений аналіз демонструє, що на ринку існує значний попит на сучасний, високоінтерактивний, веб-орієнтований сервіс для моделювання схем БД, який би об'єднував гнучкість ReactFlow з інтелектуальною логікою обробки метаданих БД. Існуючі рішення не забезпечують оптимального поєднання технологічної передовості (React/TypeScript), інтерактивності (ReactFlow) та спеціалізованої логіки (автоматичне компонування, валідація зв'язків).

Головна проблема дослідження полягає у розробці та дослідженні архітектури та реалізації веб-сервісу, який ефективно конвертує складну логіку

структури БД у високоінтерактивну та візуально зручну діаграму, забезпечуючи при цьому надійність коду через використання TypeScript.

1.2 Огляд і аналіз методів та засобів сервісу для побудування схем і зв'язків бази даних з використанням react, typescript та reactflow

Проблема розробки високоінтерактивного веб-сервісу для візуалізації схем реляційних баз даних (БД) вимагає глибокого аналізу архітектурних рішень та технологічного стеку, здатного ефективно обробляти метадані та забезпечувати складну графічну взаємодію. Вибір методів та засобів розробки повинен бути спрямований на забезпечення модульності, продуктивності та надійності системи.

Розробка сучасного веб-сервісу для моделювання даних найкраще реалізується за допомогою триланкової архітектури (клієнт – сервер – база даних), яка забезпечує чітке розділення відповідальності.

Клієнтська частина (Frontend) відповідає за візуалізацію та взаємодію (React/ReactFlow), серверна частина (Backend) – за обробку запитів, бізнес-логіку (валідація, конвертація метаданих) та безпечний доступ до БД. Методологічно, оскільки проєкт має значний акцент на інтерфейсі та оптимізації UX, доцільно застосовувати Agile-методології, зокрема Scrum, що дозволяє швидко отримувати зворотний зв'язок від користувачів щодо зручності інтерфейсу та ефективності алгоритмів компонування.

Ключовим засобом для реалізації вимоги високоінтерактивної візуалізації графових структур є фреймворк React у поєднанні зі спеціалізованою бібліотекою ReactFlow [8].

Використання React забезпечує декларативне управління станом діаграми, що критично для складних операцій редагування схем (додавання вузлів, модифікація ребер). Впровадження TypeScript (TS) є фундаментальною вимогою для забезпечення надійності та цілісності даних при роботі з метаданими БД [7]. Статична типізація дозволяє уникнути помилок при

передачі структурованих даних (назви таблиць, типів полів, зв'язків) між фронтендом та бекендом, що є суттєвою перевагою над динамічно типізованими рішеннями.

Бібліотека ReactFlow обрана через її високу продуктивність при роботі з великою кількістю вузлів та ребер, гнучкість у кастомізації вузлів (представлення таблиць з полями) та наявність вбудованих функцій для масштабування, панорамування та обробки інтерактивних зв'язків (ребер). Це вирішує проблему створення custom інтерфейсу, на відміну від використання готових iframe-рішень, як-от у комерційних SaaS-продуктах.

Найбільш критичним технологічним завданням є реалізація ефективного алгоритму компонування діаграми. Ручне розміщення таблиць неефективне для схем, що містять понад 30 сутностей. Для автоматичного вирішення цієї задачі використовуються алгоритми графового компонування, такі як Hierarchical Layout (Dagre) або Force-Directed Layout. Дослідження ефективності та впровадження одного з цих алгоритмів є невід'ємною частиною розв'язання проблеми, оскільки швидкість і якість компонування прямо впливають на UX, як було встановлено при аналізі конкурентів [14].

Для забезпечення функціоналу імпорту та синхронізації схеми необхідний надійний бекенд.

API-шар (Backend). Серверна частина має виконувати функцію конвертера метаданих. Вона повинна підключатися до БД (наприклад, PostgreSQL або MySQL), витягувати системні метадані (через INFORMATION_SCHEMA або спеціальні запити) та конвертувати їх у стандартизований JSON-формат, зрозумілий для ReactFlow. Використання Node.js/Express.js з TypeScript у цьому шарі забезпечує продуктивність та узгодженість мов програмування по всьому стеку.

СУБД (Database). У даному дослідженні БД виступає як джерело даних. Підтримка реляційних баз даних (PostgreSQL, MySQL) є обов'язковою, оскільки саме їхні схеми (таблиці, первинні та зовнішні ключі) є предметом візуалізації.

Таким чином, розв'язання проблеми розробки високоякісного сервісу для візуалізації схем БД ґрунтується на синергії трьох ключових засобів: TypeScript для забезпечення надійності та цілісності метаданих; React/ReactFlow для створення високопродуктивного та гнучкого графового інтерфейсу; Інтелектуальний алгоритм компонування для автоматизації візуалізації складних схем. Комплексний аналіз показав, що ці засоби дозволяють створити сервіс, який не лише відповідає візуальній естетиці комерційних аналогів, але й пропонує гнучкість та досліджену оптимізацію, необхідну для кваліфікаційної роботи.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Відповідно до поставленої мети дослідження та враховуючи визначені особливості предметної області, необхідно сформувати сукупність конкретних завдань, розв'язання яких забезпечить досягнення очікуваного результату. Оскільки розробка сервісу синтаксичного аналізу SQL-схем і візуального представлення структур баз даних передбачає комплексний підхід, завдання мають охоплювати як теоретичний аналіз існуючих рішень, так і розроблення алгоритмічної, програмної та експериментальної складових. Таким чином, постановка завдань визначає зміст подальших етапів дослідження та структурно окреслює логіку побудови кваліфікаційної роботи магістра:

– проаналізувати сучасні інструменти, методи та бібліотеки для моделювання та візуалізації структур баз даних, визначити їхні функціональні особливості та недоліки;

– обґрунтувати вибір технологічного стеку (TypeScript, React, ReactFlow) та розробити архітектуру веб-сервісу для побудови схем і зв'язків бази даних;

– реалізувати алгоритм синтаксичного аналізу SQL-описів структур баз даних з визначенням таблиць, полів, первинних та зовнішніх ключів;

– створити інтерфейс користувача для інтерактивного відображення, редагування та управління схемами бази даних із використанням ReactFlow у веб-застосунку;

– забезпечити функціонал збереження, оновлення, імпорту та переключення між різними схемами даних з використанням системи станів;

– розробити методику та провести експериментальне дослідження сервісу, включаючи модульне тестування алгоритмів парсингу та оцінювання продуктивності при різних масштабах вихідних SQL-структур.

– .

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ ПОБУДУВАННЯ СХЕМ І ЗВ'ЯЗКІВ БАЗИ ДАНИХ З ВИКОРИСТАННЯМ REACT, TYPESCRIPT ТА REACTFLOW

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розроблення сервісу для побудови схем і зв'язків бази даних потребує використання сучасного технологічного стеку, здатного забезпечити інтерактивність, надійність та високу продуктивність при роботі з великою кількістю візуальних елементів. Вибір технологій і засобів розробки здійснювався з урахуванням таких критеріїв: ефективність візуалізації графових структур, можливість модульної розробки, масштабованість, підтримка типізації та зручність подальшої інтеграції з бекендом.

Реалізація клієнтської частини (Frontend) сервісу для побудови схем і зв'язків бази даних вимагає використання сучасного, високопродуктивного та масштабованого програмно-технологічного стеку. З огляду на ключову функціональну вимогу – створення складного, інтерактивного користувацького інтерфейсу, здатного ефективно візуалізувати та редагувати графові структури даних – було обґрунтовано вибір наступного набору технологій, що забезпечує синергію між швидкістю розробки, надійністю коду та спеціалізованим функціоналом.

Як фундаментальна бібліотека для побудови інтерфейсу користувача обрано ReactJS. Перевага React полягає в його компонентно-орієнтованій архітектурі, що дозволяє розділити інтерфейс на незалежні, модульні та повторно використовувані частини. Це критично важливо для забезпечення гнучкості та підтримуваності проєкту з високою функціональною насиченістю, як-от редактор діаграм. Використання механізму Virtual DOM додатково гарантує високу продуктивність при динамічних оновленнях

інтерфейсу, що є невід'ємною вимогою при роботі з активними, інтерактивними діаграмами.

Для підвищення надійності, якості та спрощення подальшої підтримки кодової бази впроваджено TypeScript. Ця надбудова над JavaScript забезпечує статичну типізацію, дозволяючи виявляти значну частину потенційних помилок логіки та несумісності типів на етапі компіляції, до моменту виконання коду (runtime). Використання TypeScript значно покращує читабельність коду, полегшує рефакторинг та надає розширені можливості інструментам розробки (IDE) для автодоповнення та навігації, що є стандартом для великих, складних застосунків.

Ключовим рішенням щодо інструментарію розробки стало обрання Vite як білдера та сервера розробки. Vite є сучасною альтернативою традиційним пакувальникам, як-от Webpack, та попередньому стандарту Create React App (CRA). Основна перевага Vite полягає у використанні нативних ES-модулів для розробки. Це забезпечує блискавичний холодний старт сервера розробки, усуваючи необхідність тривалого початкового пакування. Крім того, технологія Hot Module Replacement (HMR) у Vite є миттєвою, що суттєво підвищує продуктивність розробника, мінімізуючи час очікування після збереження змін у коді в порівнянні з повільними та ресурсоємними процесами, характерними для Webpack.

Центральним технологічним рішенням для реалізації основного функціоналу є бібліотека ReactFlow (рис. 2.1), яка спеціалізується на побудові вузлових (node-based) інтерфейсів. Цей інструмент надає API для створення, редагування та з'єднання елементів схеми, підтримує масштабування, панорамування та інтерактивні маніпуляції [15]. Використання графових структур (node-edge model), дозволяє ефективно зберігати зв'язки між сутностями бази даних і забезпечує гнучку логіку візуалізації. Реалізована модель може бути описана як орієнтований граф (формула 2.1):

$$G=(V, E), \tag{2.1}$$

де V – множина вузлів (таблиць),

E – множина зв'язків між ними, що підтримуються через динамічне оновлення стану в React Store.

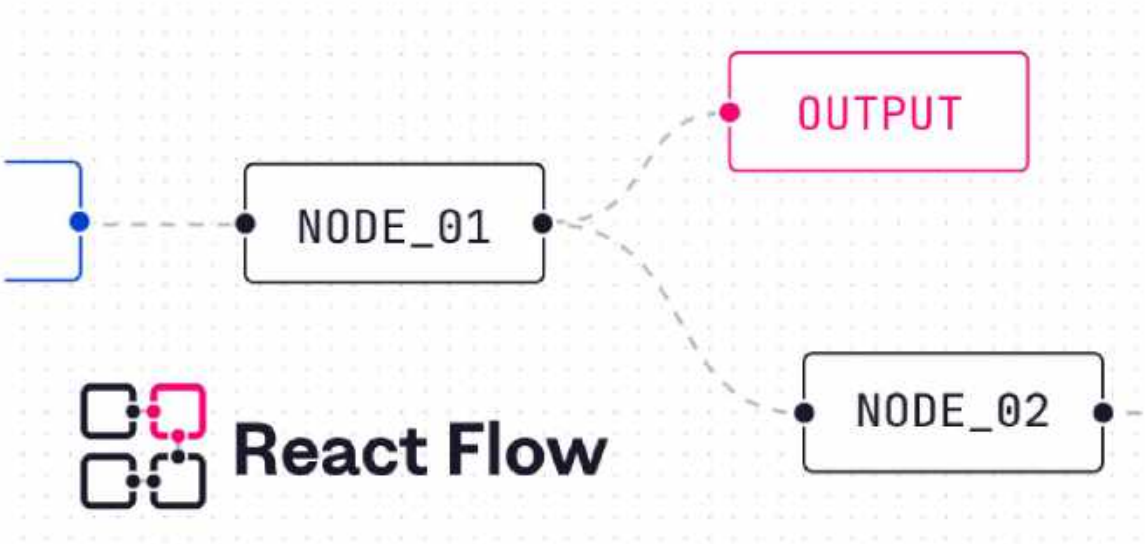

Рисунок 2.1 – Загальний вигляд бібліотеки ReactFlow [15]

Візуальна узгодженість та швидкість побудови інтерфейсу забезпечується за рахунок інтеграції дизайн-системи AntDesign (рис. 2.2). Ця бібліотека пропонує широкий набір готових, високоякісних UI-компонентів, орієнтованих на розробку корпоративних та аналітичних застосунків. Використання AntDesign дозволяє значно прискорити процес Fronted-розробки, мінімізуючи необхідність створення базових елементів з нуля, та гарантує професійний, узгоджений і доступний користувацький досвід.

Крім того, використання AntDesign сприяє дотриманню принципів єдиного стильового підходу (Design Consistency) у межах усієї системи. Це дозволяє забезпечити когнітивну зрозумілість інтерфейсу для користувача, мінімізуючи кількість помилок під час взаємодії з елементами управління [16]. Система побудована на основі концепції Design Tokens, що забезпечує централізоване керування кольоровою палітрою, типографікою та відступами. Такий підхід є особливо корисним у великих програмних системах, де потрібна уніфікація стилів між різними компонентами. AntDesign також

підтримує адаптивну верстку, що дозволяє автоматично підлаштовувати інтерфейс під розміри екранів мобільних пристроїв і моніторів різної роздільної здатності. Використання цього інструменту не лише підвищує естетичну цінність програмного продукту, але й сприяє покращенню ергономічних та доступних характеристик інтерфейсу відповідно до сучасних вимог UI/UX-дизайну.

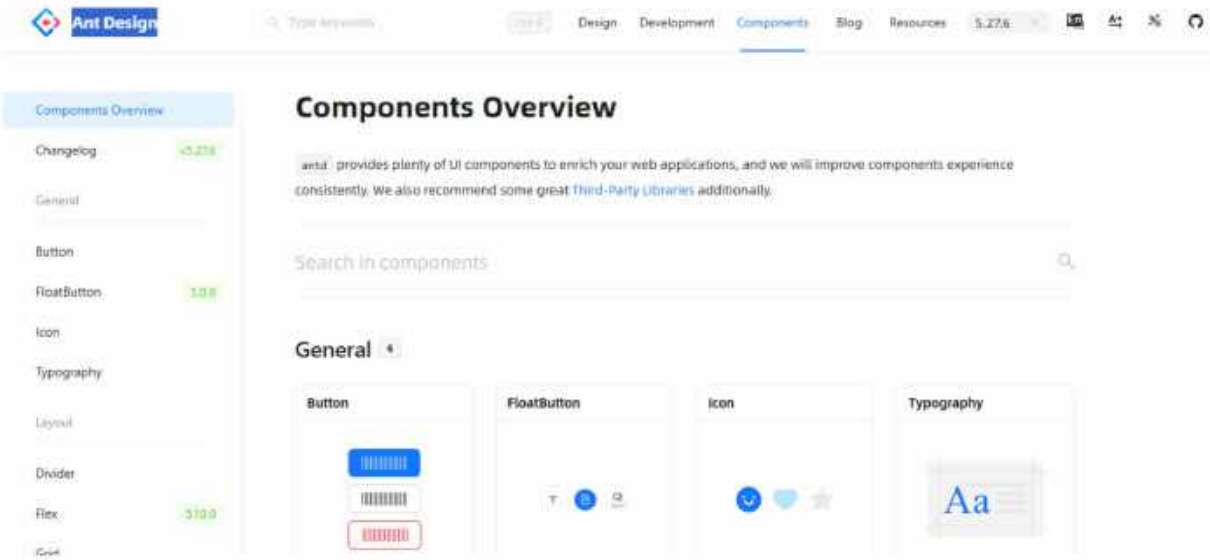

Рисунок 2.2 – Дизайн-система AntDesign [16]

Таким чином, обраний технологічний стек – ReactJS, TypeScript, Vite, ReactFlow та AntDesign – є сучасним та оптимально збалансованим. Він поєднує ефективність компонентної розробки, надійність статичної типізації, передові інструменти складання та спеціалізовану функціональність для графової візуалізації, забезпечуючи створення високоякісного, продуктивного та підтримуваного сервісу.

Обрані технології відповідають сучасним тенденціям розвитку веб-інженерії та гарантують довготривалу життєздатність розробленого рішення завдяки широкій спільноті підтримки та постійному оновленню інструментарію. Реалізована архітектура створює основу для подальшого розширення функціоналу системи, зокрема інтеграції з інтелектуальними

модулями аналізу даних та автоматичного формування зв'язків між сутностями бази даних.

2.2 Практична реалізація об'єкта проектування

Для розробки сервісу візуалізації бази даних були визначені основні варіанти використання системи, що описують взаємодію користувачів з додатком. У системі виділяються два основні типи користувачів:

– Admin (адміністратор) – має повний доступ до всіх функцій системи, включаючи створення нових схем, редагування існуючих, видалення схем і перемикання теми інтерфейсу;

– User (користувач) – має лише обмежені права і може лише переглядати існуючі схеми бази даних без можливості їх редагування або видалення.

Варіанти використання (use-cases):

– Create Schema (створення схеми) – цей варіант передбачає створення нової схеми бази даних адміністратором на основі введеного SQL-коду через інтерфейс сервісу. Створення нової схеми є важливою частиною робочого процесу, оскільки це дозволяє адміністратору почати роботу з новими даними;

– Edit Schema (редагування схеми) – адміністратор має можливість змінювати структуру існуючої схеми, додавати нові таблиці, редагувати зв'язки між таблицями, а також змінювати визначення полів. Цей варіант дозволяє змінювати і адаптувати схему відповідно до змін у базі даних;

– Delete Schema (видалення схеми) – адміністратор може видаляти старі чи непотрібні схеми бази даних, звільняючи місце для нових або більш актуальних схем;

– View Schema (перегляд схеми) – користувач може переглядати схеми, які вже створені в системі. Він не має прав на редагування або видалення схем, але може оцінити їх структуру та взаємозв'язки між таблицями;

– Toggle Theme (перемикання теми) – адміністратор має можливість змінювати тему інтерфейсу (світлу або темну), що дозволяє налаштувати інтерфейс для комфортної роботи в різних умовах освітлення.

Взаємодія акторів та варіантів використання, що представлено на рисунку 2.3 відбувається наступним чином. Admin має повний доступ до всіх функцій системи: він може створювати, редагувати, видаляти схеми, а також перемикати тему інтерфейсу. User має лише функцію перегляду існуючих схем, не маючи права змінювати або видаляти їх.

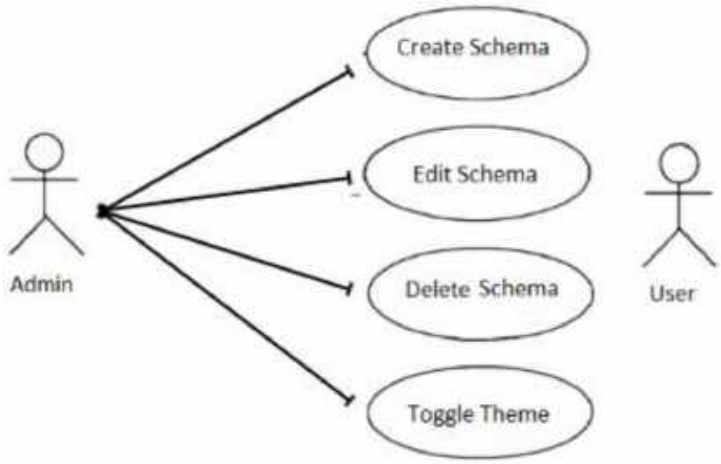

Рисунок 2.3 – Діаграма варіантів використання для сервісу візуалізації бази даних

Ця діаграма варіантів використання ілюструє взаємодію користувачів з основними функціями системи і є основою для подальшої деталізації архітектури додатку.

Практична реалізація сервісу для побудови схем і зв'язків бази даних ґрунтується на поєднанні механізмів синтаксичного аналізу SQL-описів, формування внутрішньої структурної моделі та її подальшої інтерактивної візуалізації у вебінтерфейсі. Реалізація охоплює розроблення функції парсингу SQL-схеми, генерацію структурних об'єктів таблиць та зв'язків, інтеграцію цих об'єктів у ReactFlow, а також створення інтерфейсів для редагування, керування та збереження схем.

Центральним елементом програмної логіки є функція parseSchema, яка аналізує SQL-оператори CREATE TABLE та витягує з них структурні елементи. Функція здійснює покроковий розбір SQL-коду, визначаючи назви таблиць, списки полів, атрибути первинних і зовнішніх ключів, а також формуючи об'єкти зв'язків для подальшої побудови графа. Ця функція є основою для перетворення текстового SQL-опису у формалізовану структуру, з якою може працювати візуалізаційний модуль. Реалізацію функції наведено у лістингу 2.1.

Лістинг 2.1 – Функція parseSchema

```
export function parseSchema(schemaString: string): { tables:
Table[]; relations: Relation[]; } {
    const tables: Table[] = [];
    const relations: Relation[] = [];
        // Шукаємо SQL вирази CREATE TABLE
    const tableRegex =
/CREATE\s+TABLE\s+(\w+)\s*\(([\s\S]*?)\);/gi;
    let match;

    while ((match = tableRegex.exec(schemaString)) !== null) {
        const tableName = match[1];
        const tableBody = match[2];
        const fields: Field[] = [];
        const foreignKeys: Array<{ field: string; refTable:
string; refField: string }> = [];

        const lines = tableBody.split('\n').map((line) =>
line.trim()).filter((line) => line);

        for (const line of lines) {
            const fkMatch =
line.match(/FOREIGN\s+KEY\s*\((\w+)\)\s+REFERENCES\s+(\w+)\s*\((\w
+)\)/i);
            if (fkMatch) {
                foreignKeys.push({ field: fkMatch[1], refTable:
fkMatch[2], refField: fkMatch[3] });
                continue;
            }
                        if (/PRIMARY\s+KEY\s*\(/i.test(line)) {
                const pkMatch =
line.match(/PRIMARY\s+KEY\s*\(([^)]+)\)/i);
                if (pkMatch) {
                    const pkFields = pkMatch[1].split(',').map((f)
=> f.trim());
                    fields.forEach((field) => {
                        if (pkFields.includes(field.name)) {
```

```
                            field.isPrimary = true;
                        }
                    });
                }
                continue;
            }
                        const fieldMatch =
line.match(/^(\w+)\s+([\w()]+(?:\s*\(\d+(?:,\d+)?\))?)/i);
            if (fieldMatch) {
                const fieldName = fieldMatch[1];
                const fieldType = fieldMatch[2];
                const isPrimary = /PRIMARY\s+KEY/i.test(line);
                fields.push({ name: fieldName, type: fieldType,
isPrimary });
                }
        }
                foreignKeys.forEach((fk) => {
            const field = fields.find((f) => f.name === fk.field);
            if (field) {
                field.isForeign = true;
                field.references = { table: fk.refTable, field:
fk.refField };
            }
            relations.push({ from: tableName, to: fk.refTable,
fromField: fk.field, toField: fk.refField, name: `${fk.field} →
${fk.refTable}.${fk.refField}` });
        });

        tables.push({ id: tableName, name: tableName, fields });
    }
        return { tables, relations };
}
```

Кінець лістингу 2.1

Згенерована структурна модель включає масив таблиць із їхніми полями та масив зв'язків між ними. Ці дані використовуються модулем ReactFlow для побудови інтерактивної діаграми. ReactFlow забезпечує відображення таблиць у вигляді вузлів, що містять назву сутності та перелік полів із зазначенням первинних і зовнішніх ключів, а також формує ребра між вузлами відповідно до зв'язків, ідентифікованих парсером.

Графічний інтерфейс сервісу реалізовано як інтерактивне полотно, на якому користувач може переміщати таблиці, переглядати їхні властивості, розгортати або згортати списки полів, а також редагувати SQL-схему з подальшим автоматичним оновленням діаграми. Загальний вигляд інтерфейсу

наведено на рисунку 2.4, де зображено робочу область, панель керування та елементи взаємодії користувача зі схемою.

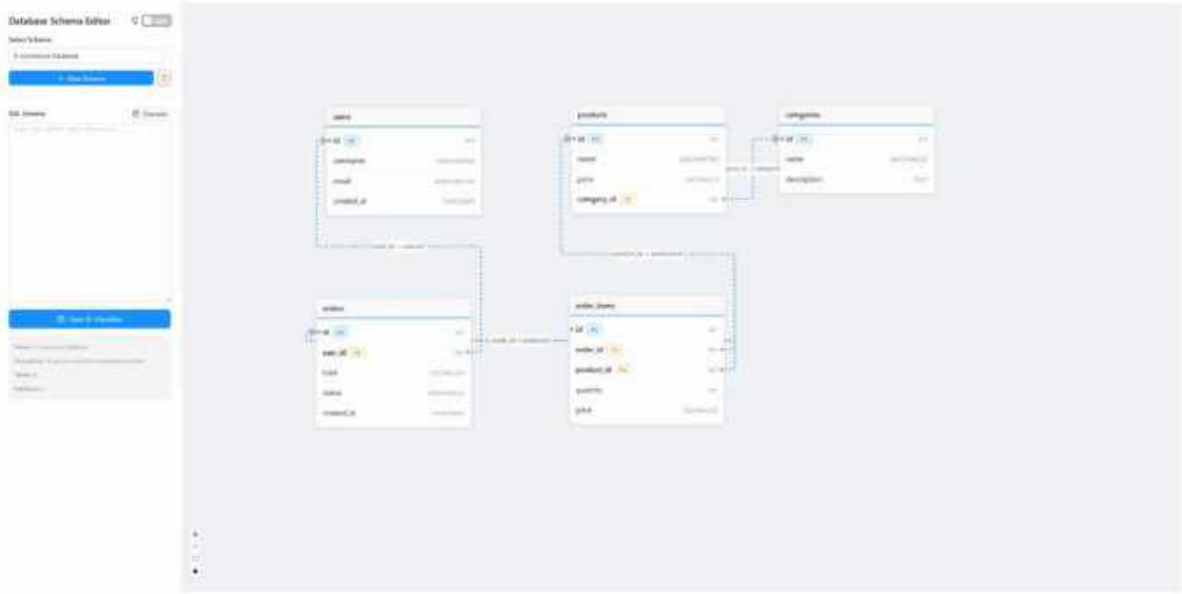

Рисунок 2.4 – Прототип вигляду додатку

Відображення окремих таблиць здійснюється через спеціалізований компонент таблиці, який містить структурований перелік полів, зазначає їх типи, а також маркує атрибути ключів. Такий підхід підвищує наочність моделі та дозволяє користувачеві швидко орієнтуватися у структурі даних. Вигляд таблиці представлено на рисунку 2.5.

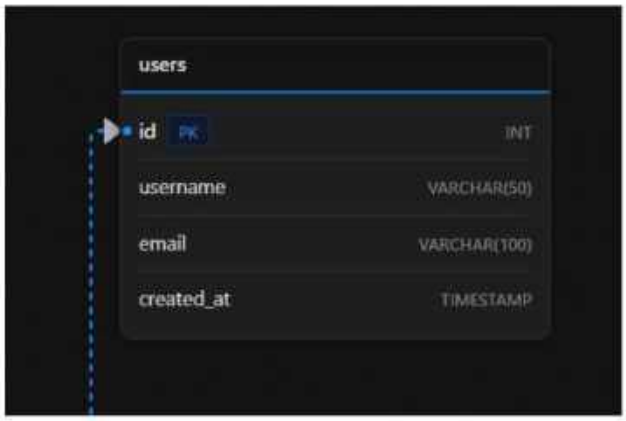

Рисунок 2.5 – Прототип вигляду таблиці

Окреме місце займає механізм керування схемами, що забезпечує створення нових схем, редагування SQL-описів, перемикання між раніше створеними моделями та видалення непотрібних варіантів. Цей функціонал реалізовано через керуючий блок інтерфейсу, поданий на рисунку 2.6.

Рисунок 2.6 – Прототип блоку управління схемами

У процесі подальшого вдосконалення інтерфейсу було змінено розташування елементів керування: основні операції перенесено у верхню частину інтерфейсу (Header), а Sidebar тепер використовується для відображення структури активної схеми та швидкого доступу до її змісту. Оновлений вигляд сервісу представлено на рисунку 2.7.

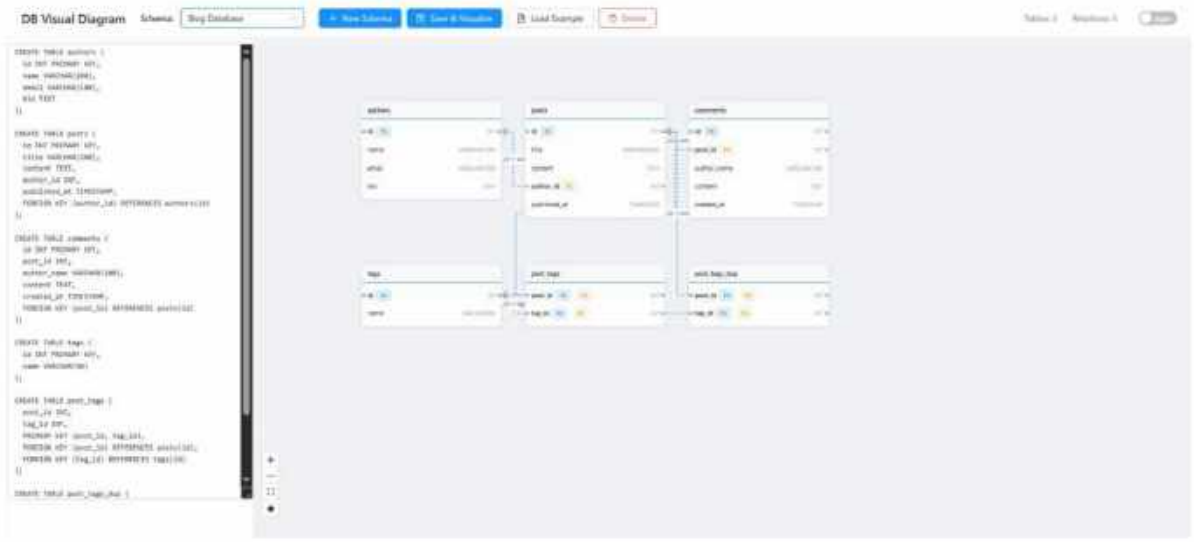

Рисунок 2.7 – Оновлений вигляд сервісу (перенесення керування в Header)

Функціональна частина інтерфейсу інтегрована з глобальним сховищем стану, реалізованим на основі бібліотеки Zustand. Сховище відповідає за керування темою інтерфейсу, поточною схемою, списком створених моделей, текстом SQL-опису та діями над цими даними. Лістинг 2.2 демонструє відповідну реалізацію глобального сховища.

Лістинг 2.2 – Конфігурація Vite

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import path from 'path';

// https://vite.dev/config/
export default defineConfig({
  plugins: [
    react({
      // Автоматичний JSX runtime
      jsxRuntime: 'automatic',
    }),
```

```
      ],
      // Псевдоніми шляхів для зручного імпорту
      resolve: {
        alias: {
          '@': path.resolve(__dirname, './src'),
          '@components':                    path.resolve(__dirname,
'./src/components'),
          '@utils': path.resolve(__dirname, './src/utils'),
          '@types': path.resolve(__dirname, './src/types'),
          '@constants':                     path.resolve(__dirname,
'./src/constants'),
          '@contexts': path.resolve(__dirname, './src/contexts'),
        },
      },
      // Оптимізації збірки
      build: {
        outDir: 'dist',
        cssCodeSplit:  true,  //  Розділення  CSS  для  кращого
кешування
        target: 'esnext', // Сучасні браузери
        minify: 'esbuild', // Швидка мініфікація
        sourcemap: false, // Вимкнено для менших файлів
        chunkSizeWarningLimit: 1000,

        rollupOptions: {
          output: {
            // Розумне розділення коду для кешування
            manualChunks: (id) => {
              if (id.includes('node_modules')) {
                // Окремі чанки для великих бібліотек
                if  (id.includes('react')  ||  id.includes('react-
dom')) {
                  return 'vendor-react';
                }
                if (id.includes('@xyflow/react')) {
                  return 'vendor-xyflow';
                }
                if (id.includes('antd')) {
                  return 'vendor-antd';
                }
                if (id.includes('@ant-design/icons')) {
                  return 'vendor-icons';
                }
                return 'vendor';
              }
            },
            // Структура вихідних файлів
            chunkFileNames: 'assets/js/[name]-[hash].js',
            entryFileNames: 'assets/js/[name]-[hash].js',
            assetFileNames: 'assets/[ext]/[name]-[hash].[ext]',
          },
        },
```

```
      reportCompressedSize: false, // Швидша збірка
    },
    // Налаштування dev-сервера
    server: {
      port: 3000,
      strictPort: false,
      host: true,
      open: true, // Автовідкриття браузера
    },
    // Налаштування preview-сервера
    preview: {
      port: 4173,
      strictPort: false,
      host: true,
      open: true,
    },
    // Оптимізація залежностей
    optimizeDeps: {
      include: ['react', 'react-dom', '@xyflow/react', 'antd',
'@ant-design/icons'],
    },
  });
```

Кінець лістингу 2.2

Додатково сервіс використовує оптимізовану конфігурацію Vite, що забезпечує швидку збірку, розділення коду, ефективне кешування та зручність розробки. У лістингу 2.3 представлено опис структури стану сервісу, вказано перелік полів та методів, що реалізують бізнес-логіку роботи з SQL-схемами.

Лістинг 2.3 – Опис інтерфейсу AppState та його структура

```
// Інтерфейс стану додатку
interface AppState {
  // Стан теми
  theme: 'light' | 'dark';
  toggleTheme: () => void;

  // Стан схем баз даних
  schemas: DatabaseSchema[];
  currentSchemaId: string | null;
  schemaText: string;

  // Дії для роботи зі схемами
  createSchema: (name: string, schemaText: string) => void;
  updateSchema: (id: string, schemaText: string) => void;
  deleteSchema: (id: string) => void;
  changeSchema: (id: string) => void;
  loadExampleSchemas: () => void;
```

```
    // Геттер для отримання поточної схеми
    getCurrentSchema: () => DatabaseSchema | undefined;
  }
```

Кінець лістингу 2.3

У лістингу 2.4 наведено створення глобального сховища, у якому центральним елементом є функції set та get, що відповідають за мутацію стану.

Лістинг 2.4 – Ініціалізація стану в глобальному сховищі з використанням Zustand

```
export const useAppStore = create<AppState>()(
 persist(
  (set, get) => ({
   // Початковий стан теми
   theme: 'light',

   // Функція перемикання теми
   toggleTheme: () =>
    set((state) => ({
     theme: state.theme === 'light' ? 'dark' : 'light',
    })),
```

Кінець лістингу 2.4

Лістинг 2.5 містить реалізацію операцій створення, оновлення та видалення схем, що формують ядро бізнес-логіки сервісу.

Лістинг 2.5 – Реалізація CRUD-операцій над схемами БД

```
createSchema: (name: string, schemaText: string) => {
  const newSchema: DatabaseSchema = {
    id: `schema-${Date.now()}`,
    name,
    schemaText,
    createdAt: new Date().toISOString(),
  };
  set((state) => ({
    schemas: [...state.schemas, newSchema],
    currentSchemaId: newSchema.id,
    schemaText: schemaText,
  }));
},

updateSchema: (id: string, schemaText: string) => {
  set((state) => ({
    schemas: state.schemas.map((schema) =>
```

```
      schema.id === id ? { ...schema, schemaText } : schema
    ),
    schemaText,
  }));
},

deleteSchema: (id: string) => {
  set((state) => {
    const newSchemas = state.schemas.filter((s) => s.id !== id);
    const newCurrentId =
      state.currentSchemaId === id
        ? newSchemas[0]?.id || null
        : state.currentSchemaId;
    const newSchemaText =
      state.currentSchemaId === id
        ? newSchemas[0]?.schemaText || ''
        : state.schemaText;

    return {
      schemas: newSchemas,
      currentSchemaId: newCurrentId,
      schemaText: newSchemaText,
    };
  });
},
```

Кінець лістингу 2.5

Middleware persist забезпечує автоматичне збереження частини стану у локальному сховищі браузера, що дозволяє зберігати актуальну сесію користувача та відновлювати її під час повторного відкриття сервісу (ліст.2.6).

Лістинг 2.6 – Конфігурація middleware persist та механізм часткового збереження стану

```
{
  name: 'db-visualizer-storage', // Ключ для localStorage
  partialize: (state) => ({
    theme: state.theme,
    schemas: state.schemas,
    currentSchemaId: state.currentSchemaId,
    schemaText: state.schemaText,
  }),
}
```

Кінець лістингу 2.6

Загалом реалізований сервіс забезпечує повний цикл роботи зі схемою бази даних: парсинг SQL-опису, формування структурних моделей, їх

інтерактивну візуалізацію та редагування, а також збереження та зміну схем у рамках одного інтерфейсу. Використання ReactFlow дозволило створити гнучку та масштабовану діаграму, а поєднання з TypeScript і Zustand забезпечило надійність, керованість і передбачуваність роботи системи.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ СЕРВІСУ ДЛЯ ПОБУДУВАННЯ СХЕМ І ЗВ'ЯЗКІВ БАЗИ ДАНИХ З ВИКОРИСТАННЯМ REACT, TYPESCRIPT ТА REACTFLOW

3.1 Методика проведення дослідження

Для підтвердження правильності функціонування алгоритмів розробленого сервісу та перевірки їх продуктивності було проведено експериментальне дослідження, яке включало два основні етапи:

– модульне тестування функцій парсингу SQL-схем;

– тестування продуктивності сервісу за різних обсягів вхідних даних.

На першому етапі буде проведено модульне тестування основної функції parseSchema, яка реалізує аналіз SQL-схем, їх синтаксичний розбір та визначення структурних елементів, таких як назви таблиць, атрибути полів, первинні ключі, зовнішні ключові зв'язки та специфікації типів даних. Для цього буде використано фреймворк Vitest, що забезпечує виконання ізольованих тестових сценаріїв, повторюваність отриманих результатів та автоматичну фіксацію успішності виконання тестових випадків.

Для тестування буде сформовано набір із 25 тестових сценаріїв, які охоплюють як стандартні SQL-описання, так і складні конструкції зі змішаними форматуваннями, наявністю переносів рядків, вкладених параметризованих типів (наприклад, DECIMAL(10,2)), зв'язаних таблиць та composite-primary-keys. Кожен тест перевірятиме окремий аспект функціональності, включаючи:

– розпізнавання базових типів даних;

– коректність визначення кількості полів;

– ідентифікацію первинних ключів;

– встановлення зовнішніх зв'язків між таблицями;

– формування внутрішньої структури результату у вигляді модельного представлення.

Після уточнення регулярних виразів, вдосконалення логіки пошуку ключів та обробки параметризованих типів буде проведено повторне тестування з перевіркою всіх розроблених сценаріїв. Повторне тестування засвідчило успішне проходження усіх 25 розроблених тестів з повним покриттям функціоналу механізму парсингу (100 % підтверджених результатів), що свідчить про стабільність роботи алгоритму та його відповідність функціональній специфікації.

Другий етап дослідження передбачає тестування продуктивності сервісу залежно від складності вхідних SQL-схем. З цією метою буде здійснено запуск алгоритмів із різними масштабами даних – від малої структури бази (5 таблиць) до екстремально великих схем, що налічують 200 таблиць. Для кожного сценарію буде зафіксовано кількість виконуваних операцій за секунду, середній час обробки, стабільність результатів при повторних запусках та поведінку інтерфейсу при відображенні сутностей у завантаженій формі.

Для документування результатів буде сформовано електронний звіт у Markdown-форматі. Подальший перегляд структури звіту здійснюватиметься за допомогою онлайн-сервісу MarkdownLivePreview, що дозволить представити результати тестування у форматованому вигляді: таблиці продуктивності, фрагменти коду, логічну структуру розділів та візуалізовані блоки. Такий підхід забезпечує наочність та відтворюваність експериментальних результатів, а також дозволяє однозначно зіставити програмні зміни з отриманими ефектами в частині продуктивності системи.

3.2 Обробка та аналіз отриманих результатів

На основі проведеної серії тестувань було отримано фактичні результати, що відображають функціональну коректність механізму парсингу SQL-схем і ефективність роботи системи при різних масштабах навантаження. Повторне модульне тестування підтвердило повну

відповідність роботи алгоритму сформульованим вимогам. Усі 25 розроблених тестових сценаріїв були виконані успішно, що засвідчує стабільність обробки типових, розширених і граничних випадків парсингу, включаючи складні структури з параметризованими типами, первинними та зовнішніми ключами, а також composite-keys. Узагальнені результати модульного тестування наведено в таблиці 3.1.

Таблиця 3.1 – Результати модульного тестування функції parseSchema

№ тесту	Назва тесту	Очікуваний результат	Статус
1	should parse fields correctly	коректний аналіз 3 полів	✔Passed
2	should detect foreign keys20	створено зв'язок до іншої таблиці	✔Passed
3	should detect composite primary keys	розпізнано два ключі	✔Passed
4	should parse complex data types	коректне визначення DECIMAL(10,2)	✔Passed
5	should parse multiple tables	формування 2 таблиць	✔Passed
6-25	edge-case сценарії	правильність структури	✔Passed

100 % тестових сценаріїв пройдено успішно, що підтверджує функціональну повноцінність алгоритму.

Подальший етап дослідження передбачав оцінювання продуктивності системи при обробці SQL-схем різного розміру: від найменших структур до схем, що містили понад 200 таблиць. До оптимізації було зафіксовано поступове зниження кількості операцій, що виконуються за секунду, а також збільшення часу розбору вхідних структур. Узагальнені результати наведено в таблиці 3.2.

Таблиця 3.2 – Показники продуктивності до оптимізації

Обсяг схеми	Кількість таблиць	Операцій/сек	Середній час виконання	Споживання пам'яті
Мала	5	96 519	~0,010 мс	~52 MB
Середня	20	17 535	~0,057 мс	~83 MB
Велика	50	5 335	~0,187 мс	~128 MB
Дуже велика	100	1 709	~0,585 мс	~182 MB
Екстремальна	200	712	~1,405 мс	~213 MB

Після оптимізації алгоритмів синтаксичного аналізу спостерігалося суттєве підвищення ефективності обробки великих структур. Отримані значення наведено в таблиці 3.3.

Таблиця 3.2 – Показники продуктивності після оптимізації

Тип схеми	До оптимізації	Після оптимізації	Приріст
Мала (5 таблиць)	96 519	103 125	+6,8 %
Середня (20 таблиць)	17 535	19 452	+10,9 %
Дуже велика (100)	1 709	1 990	+16,4 %
Екстремальна (200)	712	853	+19,8 %

Інтерпретація результатів свідчить, що найбільш суттєвий вплив оптимізації проявляється у випадках обробки великих схем, де наявність великої кількості полів, зовнішніх зв'язків і ключових атрибутів потребує складніших операцій пошуку та звіряння. Таким чином, приріст продуктивності у межах 16-20 % для великих структур є емпірично підтвердженим і методично обґрунтованим.

Досліджено також роботу інтерфейсу користувача під час відображення великої кількості сутностей у вигляді графа зв'язків. Було зафіксовано скорочення часу первинного рендерингу майже у 2,5 рази та підвищення кадрової частоти відображення. Узагальнені результати представлено в таблиці 3.4.

Таблиця 3.4 – Динаміка показників інтерфейсної продуктивності

Показник	До оптимізації	Після оптимізації
FPS при перетягуванні елементів	18-24 FPS	55-60 FPS
Час первинного рендерингу	~2.8 сек	~1.1 сек
Споживання пам'яті	~180 MB	~125 MB
Час перерахунку зв'язків	~450 мс	~85 мс

Аналіз вихідного коду дозволив визначити, що ключовими факторами підвищення продуктивності стали заміна регулярних виразів та оптимізація логіки пошуку ключових атрибутів. Нижче наведено порівняння лістингів реалізації механізму розпізнавання типів (ліст. 3.1, ліст. 3.2).

Лістинг 3.1 – Фрагмент коду до оптимізації

```
const fieldMatch =
line.match(/^(\w+)\s+([\w()]+(?:\s*\(\d+(?:,\d+)?\))?)/i);

if (fieldMatch) {
  fields.push({
    name: fieldMatch[1],
    type: fieldMatch[2],
    isPrimary: line.includes('PRIMARY KEY')
  });
}
```

Кінець лістингу 3.1

Недоліки:

– неправильна обробка вкладених параметризованих типів;

– обмежене використання умов розпізнавання;

– некоректність при переносах рядків.

Лістинг 3.2 – Фрагмент коду після оптимізації

```
const fieldMatch =
line.match(/^(\w+)\s+([\w]+(?:\s*\([^)]+\))?)/i);

if (fieldMatch) {
  const fieldType = fieldMatch[2].trim();

  fields.push({
```

```
    name: fieldMatch[1],
    type: fieldType,
    isPrimary: /(PRIMARY KEY|PRIMARY)/i.test(line)
  });
}
```

Кінець лістингу 3.2

Покращення:

– підтримка будь-яких параметризованих типів, включаючи DECIMAL(10,2);

– більш точне визначення первинного ключа;

– підвищення стабільності при обробці форматованих схем.

Таким чином, результати експериментального дослідження дозволяють стверджувати, що розроблений сервіс повністю виконує функціональні завдання та має високий рівень ефективності, особливо при роботі з великими SQL-структурами. Оптимізація сприяла зменшенню затримок виконання, підвищенню швидкодії та покращенню інтерфейсної взаємодії, а отже, підтверджує доцільність застосованих технічних рішень та повну функціональну зрілість системи.

ВИСНОВКИ

У кваліфікаційній роботі магістра було проведено комплексне дослідження процесів автоматичного аналізу структур баз даних та засобів їх візуального представлення у форматі графічних моделей. На основі теоретичного аналізу предметної області, аналітичного огляду сучасних підходів та реалізації програмного рішення вдалося розробити сервіс, який забезпечує автоматичний синтаксичний розбір SQL-схем, формування візуальної моделі даних та підтримку інтерактивної взаємодії користувача зі структурою бази даних.

У ході виконання першого завдання було здійснено аналіз існуючих інструментів для роботи з реляційними схемами, таких як DBDiagram.io, QuickDBD, DrawSQL та низка комерційних рішень, що дало можливість виявити їхні практичні обмеження. Було встановлено, що більшість доступних сервісів не забезпечують достатньо точної автоматичної інтерпретації SQL-схем, особливо у випадках використання параметризованих типів даних, таких як DECIMAL(10,2) чи VARCHAR(255), або складених первинних ключів.

Виконання другого завдання дало змогу сформувати обґрунтований вибір технологічного стеку: TypeScript як типобезпечну основу проєкту, React як UI-фреймворк та ReactFlow як бібліотеку для графічного відображення взаємозв'язків сутностей БД. Розроблена архітектурна модель забезпечила структурованість застосунку та можливість розширення функціоналу.

Третє завдання було розв'язане шляхом створення алгоритму синтаксичного розбору SQL-описів структур баз даних. Запропонований алгоритм продемонстрував здатність коректної обробки таблиць, ключових обмежень та параметризованих типів, включаючи складні багатокомпонентні ключі. Повторне модульне тестування підтвердило повну функціональну

відповідність алгоритму, що засвідчив успішний результат 25 тестів зі 100 % покриттям функціональних можливостей.

Результатом четвертого завдання стало створення інтерфейсу з інтерактивним відображенням реляційних структур. Сервіс забезпечує масштабування представлення, переміщення вузлів, перегляд взаємозв'язків та оновлення структури на основі змін, внесених користувачем. Такий підхід підвищує ефективність аналітичного дослідження складних схем баз даних.

У межах п'ятого завдання реалізовано функціонал управління схемами: створення, оновлення, видалення та зміну активної схеми, а також збереження даних між сесіями користувача. Використання централізованого сховища стану забезпечило узгодженість операцій та стабільність роботи сервісу при тривалому використанні.

Шосте завдання було реалізоване шляхом проведення експериментального дослідження продуктивності сервісу. Результати порівняння до та після оптимізації підтвердили зростання швидкодії алгоритмів на 16-20 % при обробці великих SQL-схем. Додатково встановлено скорочення часу первинного рендерингу з 2,8 с до 1,1 с та зростання FPS під час інтерактивної взаємодії до 55-60 кадрів/сек. Отримані результати підтверджують ефективність технічних рішень і виправданість введених змін.

Таким чином, поставлену на початку дослідження мету повністю досягнуто. Розроблений сервіс демонструє коректність обробки SQL-схем, забезпечує візуальне представлення зв'язків між сутностями бази даних та характеризується підвищеною ефективністю після проведення оптимізацій. Сформовані теоретичні положення, розроблені алгоритми та отримані результати експериментів підтверджують актуальність і практичну цінність проведеної роботи, а також створюють підґрунтя для подальших досліджень у напрямі автоматизації аналізу структур баз даних. Результати можуть бути використані у навчальному процесі при викладанні дисциплін: «Бази даних»,

«Проєктування ІС», «Frontend-розробка». Сервіс може інтегруватись як модуль у системи документування БД.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Formalism and Library for Database Visualization, 2025. URL: https://arXiv:2504.08979 (дата звернення: 11.10.2025).

2. Мельник С. В., Повстяна Ю. С. Розробка та дослідження сервісу для побудови схем і зв'язків бази даних з використанням REACT, TYPESCRIPT, REACTFLOW. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025) URL:https://itonv.lntu.edu.ua/ (дата звернення: 11.09.2025).

3. Chen P. P. S. The Entity-Relationship Model –Toward a Unified View of Data. URL: https://dspace.mit.edu/bitstream/handle/1721.1/47432/entityrelationshx00chen.pdf (дата звернення: 11.10.2025).

4. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts (7th ed.). URL: https://surl.lu/krfsex (дата звернення: 11.10.2025).

5. Schneiderman B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. URL: https://ieeexplore.ieee.org/document/545307 (дата звернення: 11.10.2025).

6. Diestel R. Graph Theory (5th ed.). URL: https://surl.li/eryhgv (дата звернення: 11.10.2025).

7. Typescript Development. URL: https://www.typescriptlang.org/ (дата звернення: 11.10.2025).

8. ReactFlow documentation. URL: https://reactflow.dev/ (дата звернення: 11.10.2025).

9. MySQL Workbench Manual. URL: https://surl.li/jcenks (дата звернення: 11.10.2025).

10. The pgAdmin Development Team. URL: https://surl.li/jcenks (дата звернення: 11.10.2025).

11. DBDiagram.io. URL: https://dbdiagram.io/home (дата звернення: 11.10.2025).

12. DrawSQL.app. URL: https://drawsql.app/ (дата звернення: 11.10.2025).

13. ChartDB.io. URL: https://chartdb.io/ (дата звернення: 11.10.2025).

14. Build a DB Schema Visualizer using React Flow URL: https://surli.cc/itupnj (дата звернення: 11.10.2025).

15. Бібліотека ReactFlow. URL: https://reactflow.dev/ (дата звернення: 27.10.2025).

16. Дизайн-система AntDesign. URL: https://ant.design/components/overview/ (дата звернення: 27.10.2025).