Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

(повна назва)

Кафедра програмної інженерії

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський)

Генерація інформаційної системи на основі ER діаграми. Генерація клієнтської частини на основі ER-діаграми

(тема)

Виконав:

студент 4 курсу, групи ПЗПІ-21-9

Колосс М.В.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного\_ \_\_

забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник доцент. кафедри ПІ Чуприна А. С.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_ Смеляков К.С

(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_\_\_\_\_

(підпис)

« » 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Колосс Матвію Віталійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Генерація інформаційної системи на основі ER діаграми. Генерація клієнтської частини на основі ER-діаграми

Затверджена наказом по університету від 20.05. 2025р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 11.06.2025 \_

3. Вихідні дані до роботи. Програмна система Генерація інформаційної системи на основі ER діаграми.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 19.05.2025 | *виконано* |
| 2 | Створення специфікації ПЗ | 21.05.2025 | *виконано* |
| 3 | Проектування ПЗ | 23.05.2025 | *виконано* |
| 4 | Розробка ПЗ | 28.05.2025 | *виконано* |
| 5 | Тестування ПЗ | 30.05.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 05.06.2025 | *виконано* |
| 7 | Підготовка презентації та доповіді | 06.06.2025 | *виконано* |
| 8 | Попередній захист | 09.06.2025 | *виконано* |
| 9 | Нормоконтроль, рецензування | 06.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 10.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 11.06.2025 | *виконано* |

Дата видачі завдання 18 лютого 2024р.

Студент (ка) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Колосс М.В.

(підпис)

Керівник роботи доцент. кафедри ПІ Чуприна А. С.

(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 40 с., 20 рис., 9 джерел.

ГЕНЕРАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ ER ДІАГРАМИ, ГЕНЕРАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ НА ОСНОВІ ER-ДІАГРАМИ, КЛІЄНТСЬКА ЧАСТИНА, ПРОГРАМНА СИСТЕМА, С#, СЕРВЕРНА ЧАСТИНА, CSS, ENTITY FRAMEWORK, HTML, JS, MONGODB, MS SQL, REACT

Об’єкт розробки – клієнтська частина створення проектів та створення діаграми.

Мета розробки – розроблення програмної системи для генерації інформаційної системи на основі ER діаграми.

Метод рішення – мова програмування Javascript, бібліотека React.js, середовище розробки Visual Studio Code.

У результаті розроблено клієнтську частину програмної системи, призначеної для генерації інформаційної системи на основі ER-діаграми.

# ABSTRACT

GENERATION OF INFORMATION SYSTEM BASED ON ER DIAGRAM, GENERATION OF CLIENT PART BASED ON ER DIAGRAM, CLIENT PART, SOFTWARE SYSTEM, C#, SERVER PART, CSS, ENTITY FRAMEWORK, HTML, JS, MONGODB, MS SQL, REACT

The object of development is the client side of project creation and diagram creation.

Development goal - development of a software system for generating an information system based on an ER diagram.

Solution method - Javascript programming language, React.js library, Visual Studio Code development environment.

As a result, the client part of the software system designed to generate an information system based on an ER diagram was developed.

Я, Колосс Матвій Віталійович, студент гр. ПЗПІ-21-9, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Генерація інформаційної системи на основі ER діаграми. Кастомізація застосунку на основі конфігурації обраних користувачем», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ЗМІСТ

[ВСТУП 8](#_Toc198655710)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 9](#_Toc198655711)

[1.1 Аналіз предметної галузі 9](#_Toc198655712)

[1.2 Виявлення та вирішення проблем 10](#_Toc198655713)

[1.3 Постановка задачі 11](#_Toc198655714)

[2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 12](#_Toc198655715)

[3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ 15](#_Toc198655716)

[3.1 UML проектування ПЗ 15](#_Toc198655717)

[3.2 Проектування архітектури ПЗ 19](#_Toc198655718)

[3.3 Проектування структури зберігання даних 23](#_Toc198655719)

[4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ 25](#_Toc198655720)

[5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 34](#_Toc198655721)

[ВИСНОВКИ 40](#_Toc198655722)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 41](#_Toc198655723)

# ВСТУП

У сучасному цифровому світі розробка інформаційних систем є ключовою складовою автоматизації бізнес-процесів, організації обліку, обробки даних та взаємодії з клієнтами. Проте створення таких систем зазвичай вимагає значних часових та технічних ресурсів, залучення кваліфікованих спеціалістів і глибоких знань у сфері програмування. Для спрощення цього процесу виникає потреба у інструментах, які дозволяють автоматизувати частину розробки — зокрема, генерацію структури програмної системи на основі формалізованих моделей, таких як ER-діаграми.

Розроблена програма виконує саме таку функцію: вона дозволяє користувачу створити проєкт, побудувати ER-діаграму, яка описує структуру бази даних та логіку взаємозв’язків, а потім — автоматично згенерувати повноцінний вебсайт. Згенерований сайт створюється з використанням технологій C# ASP.NET, що дозволяє одразу отримати робочий прототип інформаційної системи, готовий до запуску, тестування та подальшої модифікації.

Цей підхід значно скорочує час розробки, мінімізує кількість рутинної роботи та знижує бар’єр входу для молодих спеціалістів і студентів. Він особливо корисний у навчальних цілях, стартапах, а також під час первинного проєктування складних систем, коли важливо швидко перевірити працездатність архітектури бази даних у вигляді реального додатку.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної галузі

В умовах цифровізації всієї економіки, автоматизація рутинних операцій стає визначальним фактором ефективності бізнесу. Особливо це стосується управління інформаційними ресурсами, зокрема базами даних, вебінтерфейсами для обробки записів, а також швидкого створення адміністративних панелей для різних галузей — від e-commerce до внутрішніх корпоративних систем.

Потреба в CRUD-системах (Create, Read, Update, Delete) охоплює практично всі сучасні галузі: бізнес-аналітику, медицину, освіту, логістику, державне управління. Проте, процес створення таких систем вимагає від фахівців знань у кількох сферах одночасно: від проектування баз даних і написання бекенд-логіки до формування інтерфейсу користувача та тестування. Для людей без технічної освіти це стає практично неможливим, а для розробників — потребує багато часу та ресурсів.

Наявні рішення, такі як Bubble, AppGyver, Retool, дозволяють частково вирішити ці завдання. Проте, вони або мають обмеження в функціональності, або передбачають використання власного пропрієтарного стеку, що ускладнює перенесення на інші платформи. Більше того, відсутність можливості кастомного моделювання схеми бази даних і гнучкого контролю над бізнес-логікою обмежує їх практичне використання для складніших задач.

Таким чином, актуальність створення програмного продукту, який дозволяє автоматично згенерувати вебзастосунок із CRUD-функціональністю на основі побудованої користувачем ER-діаграми, обумовлена потребою в інструменті, що поєднує простоту, швидкість, гнучкість та кастомізацію.

## Виявлення та вирішення проблем

У ході аналізу існуючих інструментів генерації вебзастосунків були виявлені наступні проблеми:

* проблема складності. У більшості наявних рішень (наприклад, Django Admin, Laravel Nova) необхідно створювати моделі вручну та писати додатковий код;
* недостатня інтерактивність. Відсутність візуального конструктора, де користувач може "намалювати" свою ER-діаграму, сповільнює процес побудови логіки бази даних;
* обмежений контроль над валідацією. Часто користувач не має змоги гнучко налаштувати обмеження для кожного поля (довжина, тип символів, формати тощо);
* відсутність preview-дизайну. Користувач не може заздалегідь побачити, як виглядатиме інтерфейс з обраним шаблоном;
* cкладність розгортання. Після генерації немає можливості легко завантажити чи розгорнути готовий сайт;

Запропоноване рішення — створити платформу, яка складається з наступних модулів:

* модуль побудови ER-діаграми;
* модуль налаштування таблиць;
* модуль валідації;
* модуль дизайну;
* модуль генерації та експорту коду.

## 1.3 Постановка задачі

Мета: розробити вебплатформу, яка дозволяє користувачам без знання програмування створювати повнофункціональні вебзастосунки з CRUD-операціями, використовуючи графічний редактор.

Основні задачі:

Створити систему реєстрації та аутентифікації користувачів.

Реалізувати функціональність створення та редагування проєктів.

Забезпечити можливість активації або деактивації модуля авторизації для кінцевого проєкту.

Створити редактор ER-діаграми:

* створення таблиць і колонок;
* встановлення типів даних;
* створення зв’язків між таблицями;
* перевірка на унікальність назв і логічну цілісність.

Реалізувати конфігуратор таблиць для фільтрації, сортування та пошуку.

Побудувати модуль валідації зі створенням правил для кожного поля.

Створити систему вибору шаблону інтерфейсу з preview.

Реалізувати механізм генерації застосунку та можливість його завантаження у вигляді архіву.

# 2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Мова програмування та технології

* серверна частина реалізується на мові програмування C# з використанням сучасного стеку .NET;
* використання ASP.NET Core Web API для створення RESTful API, що забезпечують взаємодію між сервером і клієнтом;
* вебсервер реалізується з використанням ASP.NET Core MVC, в залежності від архітектурної моделі;
* react.js використовується для побудови клієнтської частини з підтримкою компонентного підходу;
* CSS застосовується для швидкої розробки адаптивного та візуально привабливого інтерфейсу користувача.

База даних

* застосовується MS SQL Server для зберігання структурованих даних користувачів та конфігурацій проєктів;
* зберігання медіа та додаткових файлів (у випадку розширення функціоналу) за допомогою Azure Blob Storage або локального файлового сховища;
* визначення моделей бази даних за допомогою ORM Entity Framework Core з подальшою генерацією міграцій для структурування схеми;
* альтернативно підтримується використання MongoDB як документоорієнтованої бази даних для зберігання неструктурованих конфігурацій, метаданих або швидких прототипів.

Взаємодія з вебзастосунком

* реалізація REST API дозволяє здійснювати операції CRUD через HTTP-запити;
* клієнтська частина взаємодіє з сервером через запити fetch/Axios, що забезпечує асинхронну комунікацію;
* система підтримує автоматичне оновлення інтерфейсу після змін, внесених у базу даних через Web API.

Безпека

* аутентифікація користувачів реалізована за допомогою JWT (JSON Web Token) для захисту маршрутів API;
* авторизація обмежує доступ до ресурсів відповідно до ролі користувача в системі (admin/user).
* паролі користувачів зберігаються у захешованому вигляді з використанням алгоритмів SHA-256.

Обробка запитів та логіка

* контролери API реалізують логіку обробки запитів, з використанням патернів Repository та Service для розділення відповідальностей;
* усі запити до бази даних виконуються асинхронно для забезпечення продуктивності;
* валідація параметрів виконується як на клієнтському рівні (React), так і на серверному (FluentValidation або DataAnnotations).

Масштабованість та продуктивність

* система розробляється з урахуванням можливості горизонтального масштабування на хмарних сервісах (Azure, AWS);
* оптимізація SQL-запитів за допомогою індексів, нормалізації та кешування дозволяє забезпечити стабільну продуктивність при великій кількості користувачів;
* передбачена можливість кешування відповідей API для зменшення навантаження на базу даних.

Документація та тестування

* автоматична генерація API-документації за допомогою Swagger/OpenAPI забезпечує зручну взаємодію для сторонніх клієнтів;
* UI-тестування клієнтської.

Таким чином, запропонована система на основі сучасного стеку технологій дозволяє створити повнофункціональний генератор CRUD-застосунків з високою швидкістю, надійністю, захищеністю та гнучкістю.

# 3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

## 3.1 UML проектування ПЗ

Представлена діаграма (див. рис. 3.1) випадків використання демонструє повний цикл дій користувача (User) в системі генерації CRUD-застосунку. Кожен етап проєктування і створення програми винесено як окремий функціональний блок, що ілюструє взаємодію користувача з ключовими підсистемами.

Изображение выглядит как диаграмма, круг, рисунок, шаблон

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.1 - Use-Case діаграма

Діаграма побудована за принципами UML і складається з наступних основних вузлів:

«Створити UML-діаграму» - цей блок охоплює первинне моделювання структури даних у вигляді ER-діаграми:

* створення нових таблиць — ініціація створення логічних сутностей;
* створення колонок — додавання атрибутів до таблиць;
* визначення типів колонок — задавання типів даних (рядок, число, дата тощо);
* визначення зв’язків між таблицями — логічне структурування через зовнішні ключі;
* визначення первинних і вторинних ключів — критично важливе для генерації EF Core моделей.

«Налаштування валідації» - після створення таблиць і колонок користувач має можливість налаштувати обмеження на введення даних:

* зробити поле обов'язковим — поля, які не можуть бути порожніми;
* задати мінімальну та максимальну довжину — для рядкових значень;
* задати числовий діапазон — обмеження для числових типів;
* вказати формат email або телефону — шаблони для коректної валідації.

«Налаштувати зовнішній вигляд застосунку» - цей блок визначає дизайн інтерфейсу:

* конфігурація кольорової палітри — вибір теми оформлення;
* конфігурація таблиць — включає налаштування структури UI-компонентів;
* налаштування фільтрів для таблиць — визначає, які поля доступні для пошуку;
* налаштування колонок таблиць — вказує відображувані поля, їх назви, сортування, фільтрацію.

«Згенерувати застосунок» - ключовий функціонал, який після збереження всіх параметрів генерує готовий сайт:

* створити структуру з підтримкою EF Core — моделі з анотаціями, контексти, міграції;
* згенерувати API-контролери з CRUD-операціями — backend-логіка доступу до даних;
* згенерувати атрибути валідації в моделях та DTO — забезпечує перевірку даних на рівні API.

Ця діаграма ілюструє, як користувач у рамках єдиного інтерфейсу може пройти повний цикл: від моделювання до отримання повноцінного вебзастосунку. Всі дії взаємопов’язані й логічно структуровані, що значно спрощує процес генерації проєкту без глибоких знань програмування.

Діаграма послідовностей (див. рис. 3.2) демонструє покрокову взаємодію між основними компонентами системи: клієнтом (вебінтерфейс), сервером (API), та базою даних (SQL або MongoDB). Вона відображає логіку створення та генерації проєкту з моменту ініціації користувачем до моменту отримання готового вебзастосунку.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2 - Діаграма послідовності

Деталізація кожного етапу дозволяє зрозуміти, як побудована взаємодія між частинами системи на практичному рівні:

* ініціація створення проєкту. Користувач, який увійшов у систему, натискає кнопку "Створити проєкт". У цей момент на клієнті формується HTTP POST-запит до маршруту /api/projects, у якому передається назва проєкту, стан увімкнення авторизації, дата створення та інші метадані;
* обробка запиту сервером. Сервер обробляє запит і ініціалізує створення нового запису проєкту в базі даних. На цьому етапі сервер перевіряє токен автентифікації, звіряє формат вхідних даних, формує унікальний ідентифікатор проєкту (UUID) та зберігає проєкт в колекції, у випадку MongoDB;
* повернення відповіді клієнту. Сервер надсилає клієнту HTTP-відповідь з ідентифікатором проєкту, який буде використовуватись надалі для асоціації даних;
* робота з редактором ER-діаграми. Після успішного створення проєкту користувач автоматично перенаправляється до розділу моделювання. У візуальному редакторі користувач створює структуру таблиць, задає імена, типи даних, зв’язки між таблицями. Кожна дія (додавання таблиці або колонки) надсилає асинхронний запит на сервер (наприклад, POST /api/tables, POST /api/columns);
* збереження структури в базі. Сервер приймає кожен запит, валідує вхідні дані, створює записи у таблицях Tables, Columns, Relations та забезпечує підтримку зв’язності. У MongoDB відповідна структура зберігається у вигляді вкладених документів проєкту;
* завершення моделювання. Після створення повної структури проєкту, користувач переходить до етапу генерації. Натискається кнопка "Згенерувати сайт", що тригерить запит POST /api/generate;
* генерація CRUD-застосунку. Сервер збирає всі пов’язані дані по проєкту (таблиці, поля, валідації, шаблон дизайну). Застосовується логіка шаблонізації: генеруються моделі, контролери, маршрути, форми. Для фронтенду створюються компоненти cshtml, таблиці, фільтри та форми редагування. Усі файли збираються у проєктну структуру;
* формування архіву. Після успішної генерації файлів сервер запаковує всю структуру у .zip-архів за допомогою спеціальної бібліотеки і віддає клієнту;
* отримання архіву користувачем. Клієнт завантажує архів безпосередньо у браузері. Далі користувач має можливість розгорнути згенерований сайт на власному комп’ютері, сервері або хостингу.

Ця модель забезпечує послідовну, ізольовану і безпечну обробку кожного етапу, зберігаючи логіку та інтеграцію всіх частин системи в єдине ціле.

## 3.2 Проектування архітектури ПЗ

Сторінка перегляду проєктів користувача - частина системи є відправною точкою для всіх дій користувача після входу в систему. Вона забезпечує доступ до списку існуючих проєктів, а також надає функціональність створення, перегляду та видалення проєктів.

Основні функції сторінки:

* вивід таблиці або плитки з наявними проєктами користувача;
* кнопка "Створити проєкт", яка переводить до першого етапу генератора;
* можливість видалення проєкту (з підтвердженням);
* можливість перегляду дати створення, кількості таблиць у проєкті тощо;
* перехід до редагування/моделювання обраного проєкту.

Інтерфейс реалізовано у вигляді адаптивного компонента React, який підключений до API /api/projects (GET/POST/DELETE).

Редактор ER-діаграми є ключовим компонентом системи. Він реалізований у вигляді drag-and-drop інтерфейсу, що дозволяє користувачеві візуально проєктувати структуру бази даних.

Можливості редактора:

* створення таблиці — визначення назви, ID, можливість видалити або перейменувати;
* додавання колонок — кожна колонка має назву, тип (рядок, число, дата, булеве значення тощо);
* редагування колонок — зміна типу даних, перейменування, видалення;
* встановлення зв’язків між таблицями — підтримка зв’язків один-до-багатьох і багато-до-одного через вказання зовнішніх ключів;
* валідація унікальності назв — система блокує повторення назв колонок та таблиць;
* збереження стану діаграми — автоматично при кожній зміні надсилаються запити POST/PUT до відповідних маршрутів API.

Усі дії користувача інтерактивні: при натисканні на таблицю чи колонку відкривається форма редагування, а створення зв’язків реалізується шляхом вказування ключів між таблицями. Це дозволяє повністю згенерувати структуру бази даних без ручного написання SQL-коду.

Система має чітко розділену архітектуру:

* frontend: реалізований на React + TypeScript, знаходиться у директорії client/src;
* backend: реалізований на ASP.NET Core, знаходиться у директорії server/SharpCraftStudio;
* обмін даними реалізується через REST API
* клієнтська частина застосовує модулі з http/ і Services/:
* umlApi.ts, projectApi.ts, projectConfigurationApi.ts, templateApi.ts, тощо — відповідають за CRUD-запити;
* DiagramService.ts, ProjectService.ts — управляють логікою збереження діаграми, колонок, фільтрації, кольорів тощо.

Основні маршрути, що викликаються з React:

* POST /api/projects — створення нового проєкту;
* GET /api/projects — отримання списку проєктів користувача;
* DELETE /api/projects/{id} — видалення проєкту;
* POST /api/uml — збереження структури ER-діаграми;
* POST /api/projectConfiguration — налаштування колонок, сортування/фільтрації;
* POST /api/projectConfiguration/validation — встановлення правил валідації;
* POST /api/template — вибір кольорової схеми;
* POST /api/projects/generate — ініціація генерації архіву.

Технічна реалізація:

* кожен запит на сервер містить токен автентифікації, отриманий під час входу;
* дані з React формуються у вигляді DTO (data transfer object) і надсилаються в API;
* backend через Entity Framework Core зберігає отриману інформацію в SQL-базі;
* після завершення всіх етапів, сервер формує ASP.NET MVC проєкт, пакує його та надсилає клієнту.

Ця архітектура забезпечує гнучкість, масштабованість і чіткий розподіл відповідальностей між частинами системи.

Діаграма розгортання демонструє фізичну архітектуру програмної системи, показуючи, які апаратні та програмні вузли взаємодіють між собою у процесі виконання. Дану діаграму показано на рисунку 3.3.

Изображение выглядит как текст, диаграмма, линия, План

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.3 – Діаграма розгортання

Основні вузли системи:

* користувач (Client Device) — це будь-який комп’ютер або мобільний пристрій із браузером, через який користувач взаємодіє із системою. Вебінтерфейс написаний на React і завантажується клієнтом з вебсервера;
* вебсервер (ASP.NET Core API Server) — виконує логіку бекенду. Приймає запити від фронтенду, обробляє дані, здійснює доступ до бази даних та виконує генерацію проєкту. Розгорнутий на окремому хостингу або у хмарі;
* база даних (Database Server) — сервер з реляційною або NoSQL СУБД (MS SQL Server або MongoDB). Містить інформацію про користувачів, таблиці, конфігурації, зв’язки та налаштування інтерфейсу;
* файлове сховище (File Storage) — використовується для тимчасового зберігання архівів .zip, які створює сервер після генерації застосунку.

Потік даних:

* користувач через веббраузер взаємодіє з React-застосунком;
* фронтенд відправляє HTTP-запити (REST API) до серверної частини (ASP.NET Core);
* серверна частина обробляє запити, зберігає або зчитує дані через ORM Entity Framework Core;
* після генерації структури, бекенд створює директорію проєкту, формує файли, архівує і зберігає в локальному або хмарному файловому сховищі;
* користувач завантажує архів з проектом через браузер.

Ця модель забезпечує масштабованість, логічне розділення відповідальностей та зручність обслуговування. Візуальне представлення діаграми наочно демонструє взаємозв'язки між усіма ключовими компонентами розгортання системи.

## 3.3 Проектування структури зберігання даних

Проєктування структури зберігання даних у межах даного CRUD-генератора побудоване на основі внутрішньої ER-моделі, яка зображена у вигляді UML-діаграми класів. Основу зберігання складають об'єкти, пов'язані між собою через колекції (списки) та вказівники на зовнішні ключі. ER-діаграму показано на риснку 3.4.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.4 – ER-діаграма

Основні сутності:

* UMLDiagram — головний контейнер діаграми. Містить колекції Tables (список об'єктів UMLTable) і Connections (список об'єктів UMLTableConnection);
* UMLTable — логічна таблиця бази даних. Має поля TableId, Name, Position, LabelColumnTableId та колекцію Columns. Поле Position вказує на координати розташування таблиці в редакторі;
* UMLTableColumn — колонка таблиці. Містить TableColumnId, Name, прапорці IsPrimaryKey, IsForeignKey, поле Operations, а також DataType, що вказує на тип даних (з enum UMLColumnDataType);
* UMLTableConnection — модель зв’язку між таблицями. Визначається LeftTableId, RightTableId, ForeignKeyColumnId і ConnectionType, що вказує на тип зв’язку (один-до-багатьох чи багато-до-одного);
* UMLColumnDataType — перелік типів даних: String, Int, DateTime, Double, Boolean;
* UMLConnectionType — типи зв’язків між таблицями (наприклад, OneToMany, ManyToOne);
* Position — координати таблиці на полотні (X, Y), зберігаються як об’єкт.

Схема зберігання у базі - у випадку MongoDB, кожна діаграма (UMLDiagram) зберігається як один документ з вкладеними масивами Tables і Connections. Таблиці містять вкладені масиви Columns, що полегшує зчитування при відтворенні схеми.

Проєктування даної структури дозволяє забезпечити збереження всіх параметрів, які створює користувач у редакторі, включаючи візуальну позицію таблиць, типи колонок і типи зв’язків. Це є основою для подальшої генерації вебзастосунку, де вся логіка будується на базі цієї моделі.

Основні сутності у структурі даних:

* user: містить дані користувача — Id, Email, PasswordHash, CreatedAt;
* project: проєкт користувача — Id, UserId, Name, CreatedAt, HasAuth, Status;
* tableSchema: логічна таблиця в межах проєкту — Id, ProjectId, Name;
* columnSchema: стовпці таблиці — Id, TableSchemaId, Name, Type, IsRequired, IsFilterable, IsSortable, Label, ValidationRules;
* relation: зв’язки між таблицями — Id, SourceTableId, TargetTableId, SourceColumn, TargetColumn, RelationType;
* template: шаблон інтерфейсу — Id, Name, CssPath, PreviewImageUrl;
* generationTask: дані для генерації застосунку — Id, ProjectId, CreatedAt, Status, ArchivePath.

Entity Framework Core (або Mongoose) використовується як ORM/ODM для управління моделями, автоматичних міграцій та створення схеми БД.

Проєктування структури зберігання забезпечує надійне збереження логіки проєкту, конфігурацій відображення, правил валідації та шаблонів, що критично важливо для точного відтворення та генерації готового вебзастосунку.

# 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

У цьому розділі подано детальний опис реалізованих функціональних компонентів, що були розроблені в межах проєкту. Зокрема, розглянуто програмну логіку, вибір технологій, структуру інтерфейсу та особливості взаємодії користувача з основними модулями системи. Особливу увагу приділено реалізації частини функціоналу, за яку відповідав автор: сторінка перегляду власних проєктів (із можливістю створення та видалення) та модуль створення UML-діаграм. Описано, як ці компоненти інтегруються в загальну архітектуру системи, які бібліотеки використовуються, а також наведено логіку основних дій користувача.

У межах розробки серверної частини було реалізовано контролер ProjectConfigurationController.cs, який відповідає за управління проєктами користувача. Контролер надає REST API для створення, перегляду списку та отримання окремого проєкту.

Основні методи контролера:

* create([FromBody] ProjectCreationDto projectCreationDto) - метод використовується для створення нового проєкту. Приймає дані у форматі JSON, які десеріалізуються у об'єкт ProjectCreationDto. Після перевірки коректності користувача та унікальності ID проєкту, викликається відповідний метод сервісу для створення нового запису в базі даних;
* get() - повертає список усіх проєктів, що належать поточному користувачу. Авторизаційний механізм дозволяє отримувати тільки ті проєкти, де OwnerUserName == userName;
* get(Guid id) - повертає дані конкретного проєкту за ідентифікатором, якщо він належить поточному користувачу. Якщо проєкт не знайдено або він не належить користувачу — повертається null.

Приклад методу для створення проєкту показано на рисунку 4.1.

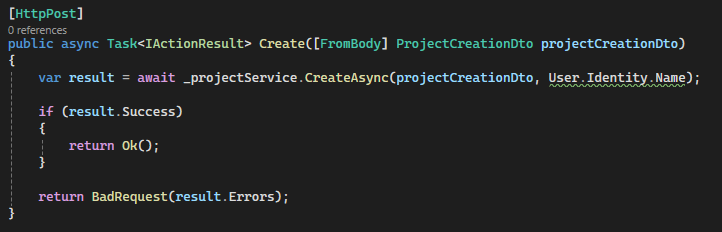


Рисунок 4.1 – Метод для створення проєкту

Приклад коду для отримання проєктів для поточного користувача показано на рисунку 4.2.

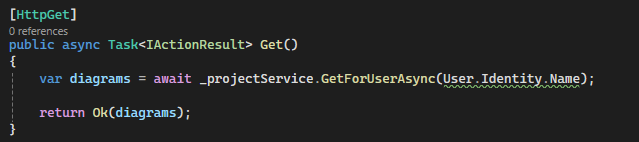


Рисунок 4.2 – Метод для отримання проекту поточного користувача

Приклад коду для отримання проєкту за його ідентифікатором показано на рисунку 4.3.

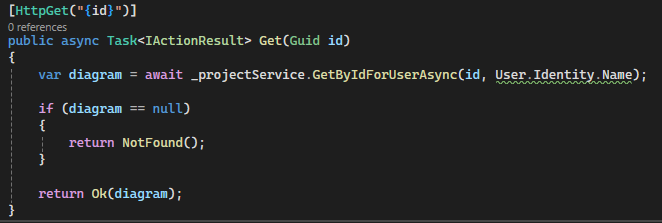


Рисунок 4.3 – Метод для отримання проєкту за його ідентифікатором

Клас ProjectCreationDto є об’єктом передавання даних (Data Transfer Object), який використовується при створенні нового проєкту. Його структура показана на рисунку 4.4.

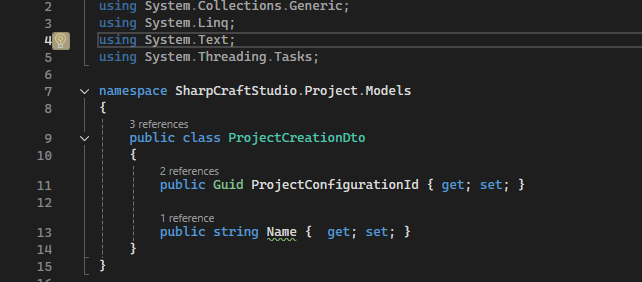


Рисунок 4.4 – Клас ProjectCreationDto

Призначення:

* projectConfigurationId: унікальний ідентифікатор, який дозволяє однозначно розпізнати проєкт;
* name: назва проєкту, що відображається в інтерфейсі користувача.

Цей DTO використовується як в контролері, так і в сервісному шарі для передачі необхідної інформації в репозиторій.

Метод CreateAsync у сервісному шарі (див.рис. 4.5) виконує основну бізнес-логіку створення нового проєкту.

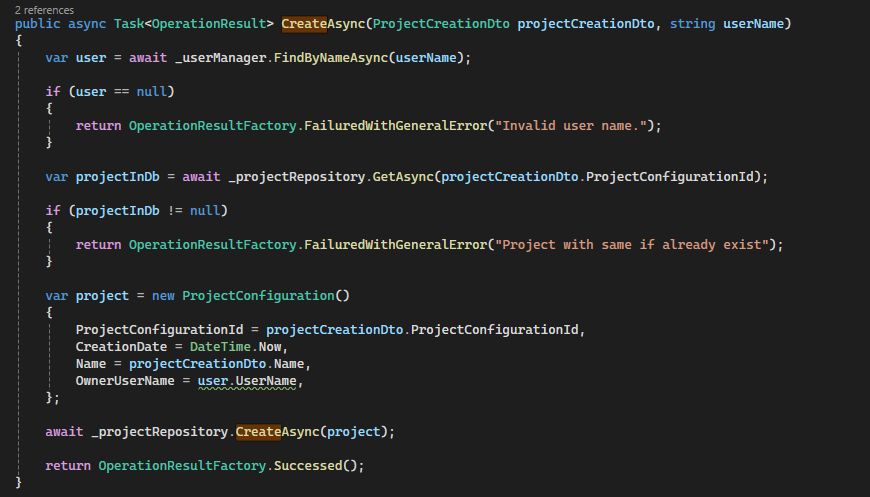


Рисунок 4.5 – Метод CreateAsync для створення проекту у сервісному шарі

Алгоритм роботи - знаходить користувача за іменем (\_userManager.FindByNameAsync(userName)), якщо користувача не знайдено — повертає помилку, перевіряє, чи існує проєкт із таким же ProjectConfigurationId. Якщо проєкт уже є — повертає повідомлення про помилку, інакше створює новий об'єкт ProjectConfiguration із полями: ProjectConfigurationId, Name, CreationDate (поточна дата), OwnerUserName. Далі викликає метод репозиторію CreateAsync, щоб зберегти об’єкт у БД та повертає OperationResult.Successed() як ознаку успішної операції.

GetByIdForUserAsync - цей метод повертає дані одного проєкту за ID, якщо проєкт належить конкретному користувачую. Приклад коду показано на рисунку 4.6.

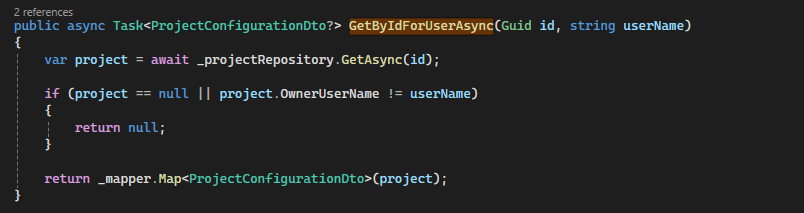


Рисунок 4.6 – Метод GetByIdForUserAsync

Якщо проєкт не існує або належить іншому користувачу — повертає null. В іншому випадку — повертає об'єкт ProjectConfigurationDto, який формується через AutoMapper.

GetForUserAsync - метод повертає всі проєкти, які належать конкретному користувачу. Приклад коду показано на рисунку 4.7.

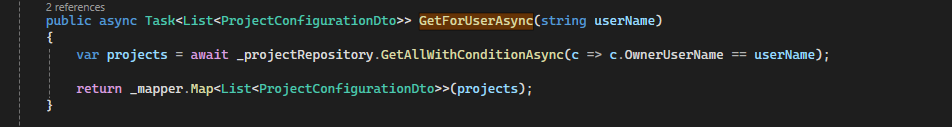


Рисунок 4.7 – Метод GetForUserAsync

Фільтрація виконується через умову c => c.OwnerUserName == userName. Після цього всі проєкти мапляться в DTO-об'єкти для подальшої передачі на фронтенд.

Ці рішення дозволяють забезпечити захищений доступ до користувацьких даних, із чітким розділенням прав, а також забезпечити логіку роботи з проєктами на основі сучасних підходів REST API та DTO-моделей.

Для реалізації збереження користувацьких UML-діаграм у системі було створено окремий функціональний модуль, який дозволяє користувачу створювати, редагувати та зберігати структуру проєкту у вигляді візуальної діаграми. Центральною частиною цього модуля є API-контролер UserUMLController та пов'язаний із ним сервіс UMLDiagramService.

Контролер UserUMLController надає єдиний, але ключовий метод для роботи з UML-діаграмами, а саме метод Save.

Призначення:

* метод приймає на вхід DTO-об’єкт UMLDiagramSaveDto, що містить збережену структуру UML-діаграми;
* запит надсилається з клієнтської частини, коли користувач натискає кнопку «Зберегти» під час роботи з діаграмою;
* контролер викликає відповідний сервісний метод SaveAsync, передаючи також ім’я поточного користувача.

Об’єкт UMLDiagramSaveDto інкапсулює всі необхідні дані для збереження UML-діаграми в базі даних. Типове наповнення цього DTO показано на рисунку 4.7.

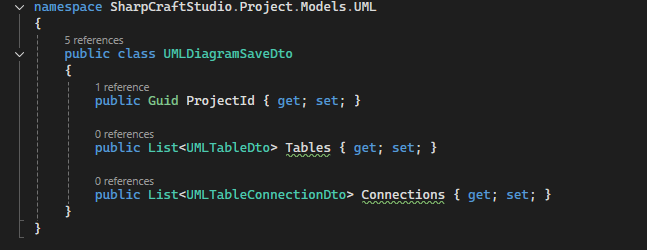


Рисунок 4.7 – UMLDiagramSaveDto клас

Опис полів:

* projectConfigurationId: ідентифікатор проєкту, до якого належить діаграма;
* diagramJson: JSON-рядок, який описує структуру UML-діаграми у форматі, зручному для подальшої обробки або рендерингу на клієнті.

Логіка збереження діаграми в UMLDiagramService. Основна бізнес-логіка, реалізована у методі показана на рисунку 4.8.

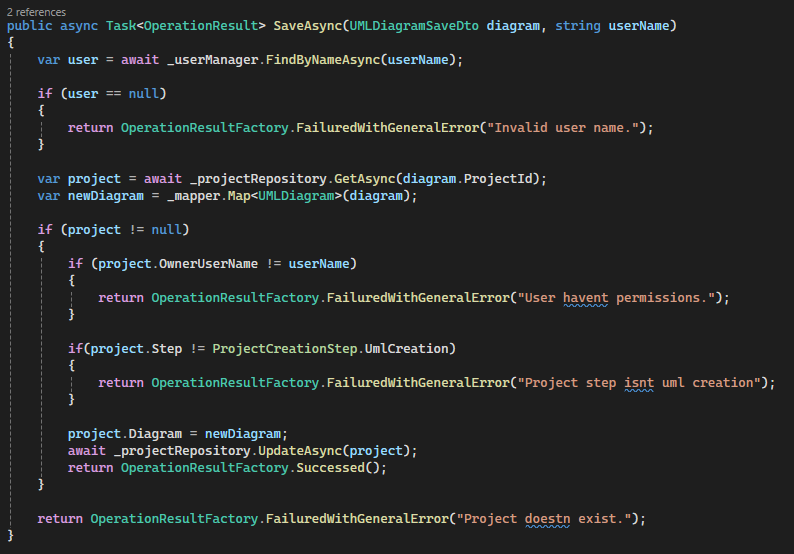


Рисунок 4.8 – Метод SaveAsync

Алгоритм роботи - отримання проєкту з бази даних по diagram.ProjectConfigurationId, перевірка, чи проєкт належить поточному користувачу (захист від несанкціонованого доступу). Якщо проєкт не знайдено або не належить користувачу — метод повертає без змін (може бути розширено до обробки помилок). Якщо перевірка успішна, то властивість UmlDiagramJson у відповідному об’єкті ProjectConfiguration оновлюється новими даними з diagram.DiagramJson. Зміни зберігаються в базі даних через репозиторій.

Ключові можливості та призначення модуля:

* збереження діаграм: структура діаграми зберігається у форматі JSON, що дозволяє легко її візуалізувати, редагувати або використовувати для генерації коду в майбутньому;
* прив’язка до проєкту: кожна діаграма жорстко пов’язана з певним проєктом, що забезпечує логічну цілісність даних;
* захист даних: реалізовано перевірку власника проєкту перед записом змін, що унеможливлює доступ до чужих діаграм;
* масштабованість: структура рішення дозволяє в майбутньому реалізувати історію змін, автозбереження або версіювання діаграм.

Цей модуль є критично важливим для зручної та інтуїтивної роботи користувача з візуальним поданням логіки свого проєкту. Він інтегрується в загальну систему, дозволяючи пов’язати створену ER/UML-діаграму з іншими частинами вебзастосунку.

Фронтенд-частина редактора UML-діаграм реалізована в рамках папки UmlCreator. Вона є ядром взаємодії користувача з візуальним інтерфейсом побудови таблиць, колонок і зв’язків між ними. Уся логіка написана з використанням бібліотеки React, React Flow, TypeScript та CSS-модулів.

Папка містить такі основні частини:

* components/DiagramConnections — модуль для управління зв’язками між таблицями (типи зв’язків: один-до-одного, один-до-багатьох);
* components/ReactFlowTables — інтеграція з бібліотекою React Flow для відображення діаграми та рендеру вузлів (таблиць), збереження та завантаження;
* components/TableList, TableListItem, TableColumn, TableItemInfo — компоненти для створення, редагування, дублювання та видалення таблиць і колонок;
* components/UI/DiagramLine — стилізація ліній зв’язку між таблицями;
* assets/ — набір графічних іконок (наприклад, зображення ключів для позначення ID).

На основі коду й інтерфейсу, реалізовано наступні функції.

Створення таблиць:

* користувач може додати нову таблицю натисканням на кнопку "New Table";
* таблиця автоматично отримує назву (NewTable, NewTable1, тощо);
* внутрішній стан зберігає список усіх таблиць і їхні поля.

Додавання колонок

* доступна функція Add column — для кожної таблиці можна додати поле з вибором типу: Int, String, DateTime, Double, Boolean;
* є можливість змінювати тип даних і назву колонки через drop-down меню;
* foreign key поля маркуються відповідною іконкою.

Створення зв’язків між таблицями:

* модуль DiagramConnections.tsx відкриває вікно, де користувач задає тип зв’язку (наприклад, A has many B);
* дані зберігаються в окремій структурі стану для візуалізації стрілок і відображення типів залежностей.

Візуальне редагування (React Flow):

* у компоненті ReactFlowTables.tsx реалізовано drag-n-drop інтерфейс переміщення таблиць;
* рендер вузлів (nodes) і зв’язків (edges) забезпечує інтерактивність: таблиці можна з’єднувати, видаляти, переміщувати;
* позиція таблиці також зберігається в даних, щоб зберігати її положення після оновлення.

Збереження діаграми

Кнопка «Save diagram» викликає функцію saveUmlDiagram, яка розміщена в файлі umlApi.ts. Код файлу umlApi.ts показано на рисунку 4.9.

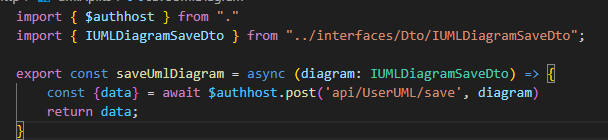


Рисунок 4.9 – Код файлу umlApi.ts

Далі відправляється об’єкт IUMLDiagramSaveDto, що містить:

* projectConfigurationId — ID проєкту;
* diagramJson — JSON-подання діаграми, яке включає: всі таблиці, їхні поля, позиції на полотні, зв’язки між ними.

Функція «Download UML» дозволяє завантажити діаграму у вигляді зображення. Користувач може виділити область та зберегти її як PNG/SVG.

Операції з таблицями - через контекстне меню (права кнопка миші по таблиці) доступні дії: сopy table, duplicate table, cut table, delete table.

Усі компоненти написані з підтримкою TypeScript для типобезпеки. Для візуалізації використовується бібліотека React Flow, яка дозволяє створювати кастомні вузли (таблиці) та кастомні зв’язки (CustomEdge.tsx). Усі стилі оформлені через \*.module.css, що забезпечує ізоляцію CSS.

Редактор UML-діаграм на клієнті реалізований як окремий модуль з чіткою структурою, підтримкою типів, інтерактивністю та можливістю збереження у backend. Вся логіка взаємодії користувача відбувається у реальному часі, а фінальний результат передається на сервер для збереження в базі даних, забезпечуючи цілісність структури даних проєкту.

# 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування - це невід'ємна частина процесу розробки програмного забезпечення, що забезпечує якість, стабільність та відповідність функціоналу заявленим вимогам. У цьому розділі описано підходи, які були використані для перевірки працездатності вебзастосунку, його окремих модулів та інтеграції між ними.

Метою тестування було виявлення та усунення можливих помилок, а також перевірка правильності роботи основних функцій, зокрема: створення та управління проєктами, побудови UML-діаграм, збереження даних, відображення зв’язків між таблицями та інтерактивної взаємодії користувача з інтерфейсом.

У рамках тестування були використані як ручні, так і автоматизовані методи перевірки. Особливу увагу приділено функціональному тестуванню клієнтської та серверної частин, з урахуванням типових сценаріїв взаємодії користувача з системою.

Одним із ключових сценаріїв взаємодії користувача з редактором UML-діаграм є можливість редагування назв таблиць та їхніх полів (колонок). У межах функціонального тестування перевірялась правильність виконання таких операцій, а також реакція системи на невалідні введення.

Ціль тестування:

* перевірити, що користувач може змінювати назви таблиць і колонок;
* перевірити, що зміни зберігаються і відображаються у візуальному інтерфейсі;
* забезпечити унікальність назв таблиць і колонок в межах однієї діаграми.

Таблиця 5.1 – Результат тестування змін назв

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Змінити назву таблиці з NewTable на Users | Назва оновлюється у списку та на полотні | Позитивний |
| 2 | Змінити назву колонки з column1 на email | Поле змінюється у візуальному інтерфейсі | Позитивний |
| 3 | Створити дві таблиці з однаковими назвами (Orders) | З'являється повідомлення про помилку, перехід на наступний крок заблоковано | Позитивний |
| 4 | Додати у таблицю Products дві колонки з однаковою назвою (id) | Валідація не дозволяє зберегти, з’являється підсвітка помилки | Позитивний |
| 5 | Перейти до наступного кроку з діаграмою, де дублюються назви | Після натискання "Move to next step" - з'являється попередження/помилка | Позитивний |

На рисунку 5.1 показано дві створених таблиці з однаковою назвою.

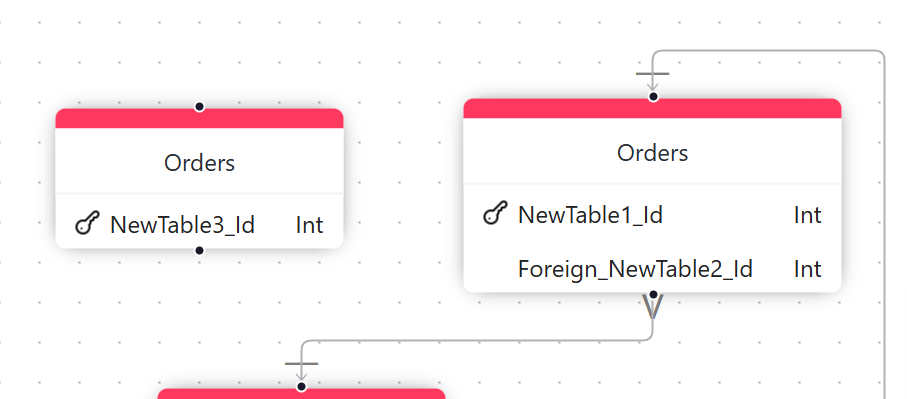


Рисунок 5.1 – Приклад двох таблиць

А на рисунку 5.2 показано що ми не можемо перейти на наступний етап, через те, що у нас є помилки.

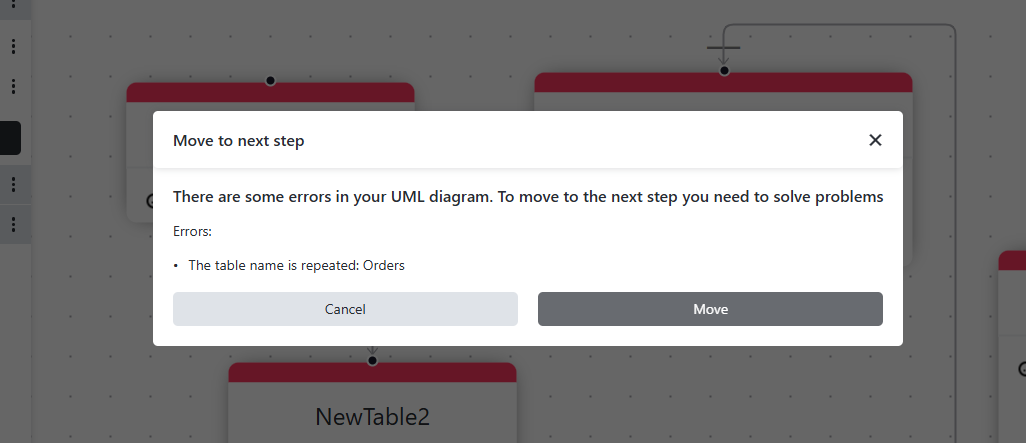


Рисунок 5.2 – Помилка при переході на настпуний етап

Тепер потрібно додати до таблиці дві однакових колонки. Для цього у таблицю «NewTable» додаємо два поля «id» (див .рис. 5.3).

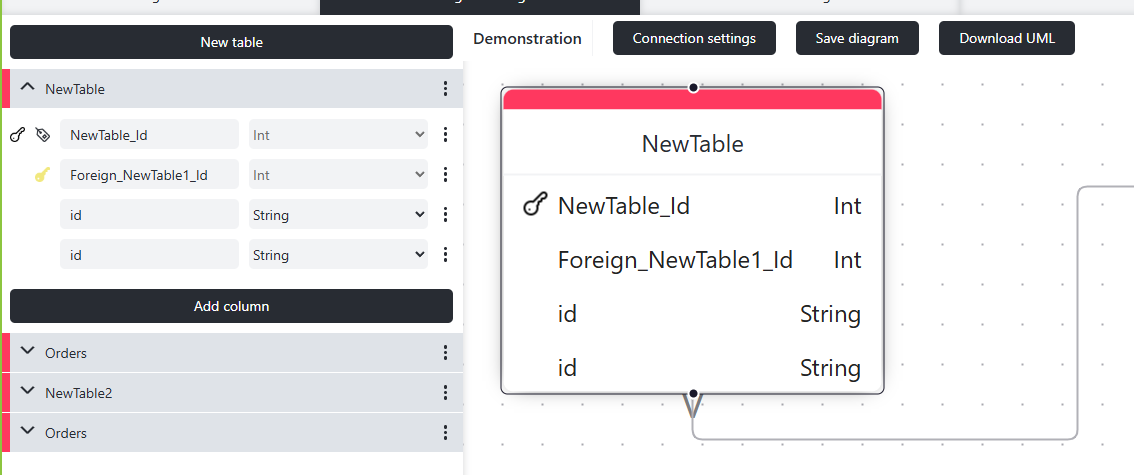


Рисунок 5.3 – Два однакових поля у таблиці

Тепер після натиснення на кнопку для переходу на наступний етап - будемо бачити наступну помилку (див. рис. 5.4).

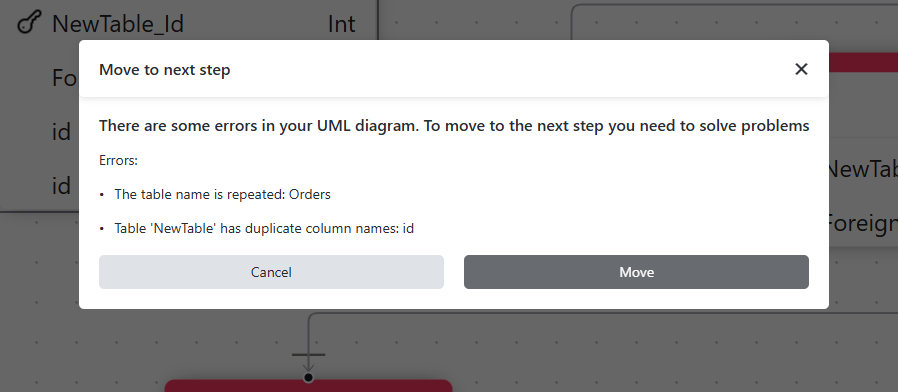


Рисунок 5.4 – Помилки після додавання двох однакових полів

Як можна побачити, все працює так як потрібно. На рівні інтерфейсу реалізовано кілька механізмів перевірки унікальності, а саме назви таблиць та назви колонок.

Назви таблиць - перевіряються глобально в межах всієї діаграми. При повторенні з’являється візуальне попередження, а перехід до наступного етапу заблоковано.

Назви колонок - перевіряються в межах кожної конкретної таблиці. У випадку дублювання поле підсвічується, а збереження змін стає неможливим.

Це дозволяє уникнути помилок на етапі генерації схеми бази даних та забезпечує структурну цілісність моделі.

У системі реалізовано можливість створювати логічні зв’язки між таблицями (один до одного, один до багатьох тощо), що дозволяє формувати повноцінну структуру бази даних або об'єктної моделі. Для забезпечення коректності цього функціоналу було проведено ручне тестування основних сценаріїв.

Ціль тестування:

* переконатися, що зв’язки можна створювати між існуючими таблицями;
* перевірити, що тип зв’язку правильно обробляється і відображається на діаграмі;
* перевірити обмеження, які не дозволяють створити некоректні або дубльовані зв’язки.

Таблиця 5.2 – Результат тестування зв’язків

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Створити зв’язок типу "1 до N" між таблицями Users і Orders | Стрілка з правильним напрямком з’являється між таблицями | Позитивний |
| 2 | Створити зв’язок між однією таблицею і собою (рекурсивний) | Відображається, якщо дозволено (наприклад, для self-reference), або блокується | Позитивний |
| 3 | Створити два однакові зв’язки між тими ж таблицями | Система не дозволяє дублювання | Позитивний |
| 4 | Видалити зв’язок через інтерфейс | Стрілка зникає, зв’язок видаляється зі схеми | Позитивний |
| 5 | Зберегти діаграму після створення зв’язку | Зв’язок коректно серіалізується у JSON і передається на сервер | Позитивний |

Приклад форми створення зв’язку показано на рисунку 5.4.

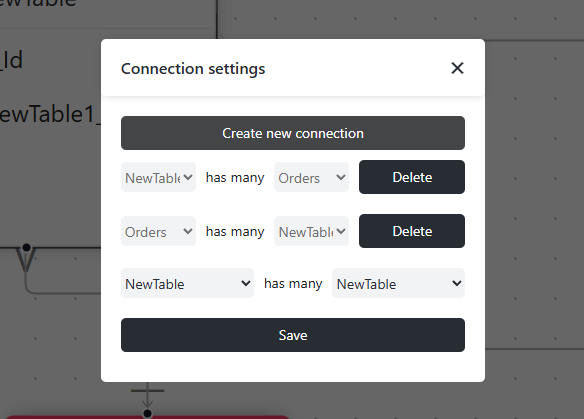


Рисунок 5.4 – Форма створення та перегляду зв’язків між таблицями

У результаті бачимо (див. рис. 5.5) що зв’язок між таблицею Users та Orders було створенно правильно.

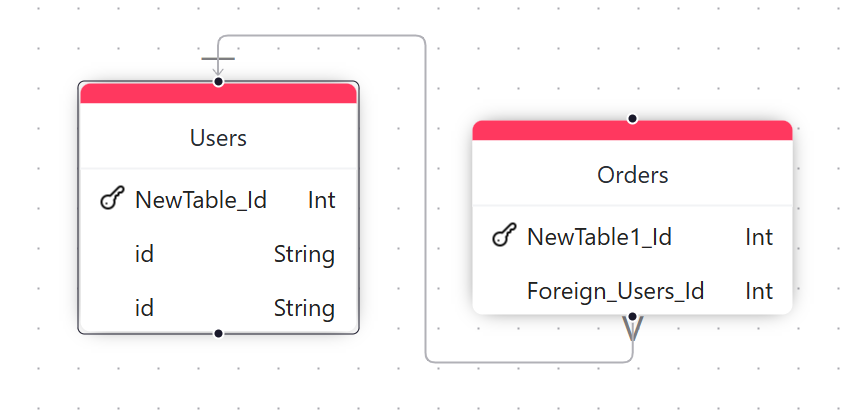


Рисунок 5.5 – Створений зв’язок між таблицею Users та Orders

Також бачимо що у таблиці Orders створено ключ «Foreign\_Users\_Id»

Проведене тестування підтвердило коректність реалізації основних функціональних можливостей редактора UML-діаграм. Усі перевірки, пов’язані з редагуванням назв таблиць і колонок, створенням зв’язків та валідацією даних, показали позитивні результати. Завдяки реалізованим обмеженням на рівні інтерфейсу вдалося запобігти типових помилок моделювання, що забезпечує цілісність і надійність структури проєкту. Система адекватно реагує на некоректні введення, надає зворотний зв’язок користувачу і не допускає збереження неконсистентних даних.

# ВИСНОВКИ

У процесі кваліфікаційоної роботи було успішно реалізовано програмну систему для автоматизованої генерації вебзастосунків на основі ER-діаграм. Було здійснено повний цикл розробки — від аналізу предметної області та виявлення проблем до реалізації клієнтської та серверної частин системи. В процесі виконання завдання були враховані сучасні вимоги до зручності інтерфейсу, гнучкості налаштувань та надійності збереження даних.

Особливу увагу було приділено моделюванню структури бази даних у зручному графічному редакторі, налаштуванню відображення колонок, застосуванню валідаційних правил та шаблонів дизайну. Реалізація генерації коду забезпечила можливість створення повноцінного CRUD-застосунку на основі збережених параметрів, що значно скорочує час розробки та знижує вимоги до рівня підготовки кінцевого користувача.

Клієнтська частина була реалізована на базі фреймворку React з використанням TypeScript, що забезпечило високу швидкість роботи, адаптивність та розширюваність. Серверна частина створена з використанням ASP.NET Core, а для збереження даних використано MS SQL та MongoDB, залежно від конфігурації. Це забезпечує надійність, масштабованість та простоту обслуговування програмної системи.

Таким чином, цілі кваліфікаційоної роботи були досягнуті повністю: створено інноваційний, ефективний та гнучкий інструмент для автоматизованої генерації вебзастосунків, який може бути використаний як у навчальному процесі, так і у реальних комерційних проєктах.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Microsoft Learn, документація ASP.NET Core. URL: <https://learn.microsoft.com/aspnet/core/> (дата звернення 06.05.2025).
2. React Documentation URL: https://react.dev/ (дата звернення 06.05.2025).
3. MongoDB Documentation URL: https://www.mongodb.com/docs/ (дата звернення 06.05.2025).
4. Entity Framework Core Documentation URL: https://learn.microsoft.com/ef/core/ (дата звернення 06.05.2025).
5. W3Schools: HTML, CSS, JavaScript tutorials URL: https://www.w3schools.com/ (дата звернення 06.05.2025).
6. Axios GitHub repository URL: https://github.com/axios/axios (дата звернення 06.05.2025).
7. Swagger — OpenAPI Specification URL: https://swagger.io/specification/ (дата звернення 06.05.2025).
8. Postman Tool — API testing URL: <https://www.postman.com/> (дата звернення 06.05.2025).
9. Stack Overflow — Спільнота розробників. URL: <https://stackoverflow.com/> (дата звернення 06.05.2025).

# ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

# Додаток Б

Слайди презентації

