НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

"На правах рукопису"
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
"___"_____2024 р.

Магістерська дисертація

на здобуття ступеня другого (магістерського) рівня вищої освіти
за освітньою програмою "Цифрові технології в енергетиці"
зі спеціальності 122 "Комп'ютерні науки"

на тему «Інструменти розробника реляційної бази даних»_____

Виконав: студент _2__курсу, групи ТР-31мп
____Міщенко Андрій Андрійович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник: доцент, к.т.н. Ірина МИХАЙЛОВА _____
(посада, науковий ступінь, вчене звання, ім'я ПРІЗВИЩЕ) (підпис)

Рецензент: доцент кафедри ТАЕ, к.т.н., доцент Ірина ФУРТАТ _____
(посада, науковий ступінь, вчене звання, ім'я ПРІЗВИЩЕ) (підпис)

Н.контроль: старший викладач Ольга БЕСПАЛА_____ _____
(посада, ім'я ПРІЗВИЩЕ) (підпис)

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

Київ - 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – другий (магістерський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

МІЩЕНКУ Андрію Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема дисертації «Інструменти розробника реляційної бази даних»
керівник роботи Михайлова Ірина Юріївна, к.т.н.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «11» листопада 2024 р. № 5040-с

2. Термін подання роботи студентом: 06.12.2024 року

3. Вихідні дані до роботи: мови програмування Java та JavaScript, фреймворки Spring та Angular, СКБД InterSystems IRIS та Redis, платформа Docker, Інтернет стандарт JWT, REST API

4. Перелік питань, які потрібно розробити:

1) проаналізувати існуючі програмні рішення: виявити їхні переваги та недоліки
2) провести дослідження сучасних засобів та технологій для розробки, які дозволять ефективно вирішити поставлену мету;
3) спроєктувати архітектуру системи та структуру бази даних;
4) визначити варіанти взаємодії користувачів із системою;
5) надати можливість користувачу створювати та керувати БД, таблицями, стовпцями за допомогою інтерфейсу;

6) надати можливість користувачу взаємодіяти з даними;
7) реалізувати кінцевий програмний продукт, використовуючи результати досліджень;
8) провести тестування та апробацію системи.

5. Орієнтований перелік ілюстративного матеріалу: діаграми на яких зображено можливі взаємодії користувача з системою, високорівнева та низькорівнева архітектури системи, структуру бази даних, класи системи, алгоритми механізмів автентифікації та створення бази даних, стани об'єкта база даних.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми роботи	15.09.2024	виконано
2	Аналіз існуючих програмних рішень	16.09.2024 – 18.09.2024	виконано
3	Вибір засобів та технологій для розробки	19.09.2024 – 21.09.2024	виконано
4	Створення архітектури системи та структури баз даних	22.09.2024 – 25.09.2024	виконано
5	Реалізація серверної частини	26.09.2024 - 09.10.2024	виконано
6	Реалізація клієнтської частини	10.10.2024 – 17.10.2024	виконано
7	Тестування	18.10.2024 – 20.10.2024	виконано
8	Оформлення магістерської дисертації	20.10.2024 – 24.11.2024	виконано
9	Передзахист	25.11.2024	виконано
10	Захист	18.12.2024	виконано

Студент _____ **Андрій МІЩЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Ірина МИХАЙЛОВА**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Магістерська дисертація на тему “Інструменти розробника реляційної бази даних” складається зі вступу, 5 розділів, висновку, переліку використаних джерел з 25 найменувань, 2 додатків, містить 46 рисунки, 25 таблиць. Повний обсяг дипломної роботи складає 111 сторінок, з яких список джерел займає 2 сторінки, додатки – 12 сторінок.

Актуальність теми. З поширенням Інтернету кількість веб-застосунків почала сильно рости. Створення веб-сервісу, що має зв’язок із зовнішніми системами є непростим завданням. Найпоширенішим зовнішнім зв’язком є база даних, тому майже всі веб-сервіси використовують її для збереження даних. У результаті почала рости кількість розробників, що стикаються з проблемою управління базами даних та їх інтеграцією в створювані програми, адже ці процеси потребують великої кількості часу та знань, що може бути нераціональним для малого та середнього бізнесу, а також власних проєктів, якими займається одна людина або мала група. Зазвичай вони потребують базових CRUD операцій та певних користувацьких запитів на вибірку даних. Популярні рішення надають обмежені можливості для вирішення даної проблеми: неповноцінний користувацький інтерфейс; відсутність підтримки користувацьких запитів; відсутність свободи в створенні таблиць та зовнішніх зв’язків між ними. У результаті це грає ключову роль у виборі потрібного рішення для кінцевого користувача. Тому є актуальною необхідністю реалізація нових інструментів розробника реляційної бази даних, що вирішували дані проблеми.

Мета дослідження. Створення інструментів розробника реляційної бази даних, що надаватимуть клієнтам можливість ефективного управління базами даних та інтеграції запитів для роботи з даними. Щоб досягнути мету були сформульовані такі завдання:

- проаналізувати існуючі програмні рішення: виявити їхні переваги та недоліки;

- провести дослідження сучасних засобів та технологій для розробки, які дозволять ефективно вирішити поставлену мету;

- спроєктувати архітектуру системи та структуру бази даних;

- визначити варіанти взаємодії користувачів із системою;

- надати можливість користувачу створювати та керувати БД, таблицями, стовпцями за допомогою інтерфейсу;

- надати можливість користувачу взаємодіяти з даними;

- реалізувати кінцевий програмний продукт, використовуючи результати досліджень;

- провести тестування та апробацію системи.

Практичне значення одержаних результатів полягає в розробці програмного продукту, щоб надати можливість створювати бази даних та управляти ними за допомогою користувацького інтерфейсу, а також можливості інтегрувати управління даними в таблицях за допомогою API, що дозволяє виконувати CRUD та користувацькі запити, без витрачання на це великої кількості часу та необхідності додаткових знань для роботи з SQL. Створені інструменти розробника реляційної бази можуть бути корисними для одиночних чи малої групи розробників, а також малого чи середнього бізнесу, що не потребують складної інфраструктури довкола бази даних.

Апробація результатів дисертації.

Основні положення роботи були представлені на:

1. Міщенко А. А., Михайлова І. Ю. Інструменти розробника реляційної бази даних. *Діджиталізація науки як виклик сьогодення* : матеріали VII Міжнародної студентської наукової конференції, м. Полтава, 25 жовтня 2024 р. Вінниця. Україна «UKRGOLOS Group», 2024. С. 584 -586.

2. Міщенко А. А., Михайлова І. Ю. Інструменти розробника реляційної бази даних // Студентський науковий журнал Universum. 2024. №14. С. 284-288.

Ключові слова: інструменти, реляційні бази даних, CRUD операції, користувацькі запити, API.

ABSTRACT

Structure and scope of the thesis. The master's thesis on the topic "Relational database developer tools" consists of an introduction, 5 chapters, a conclusion, a list of 25 references from the sources used, 2 appendices, contains 46 pictures, 25 tables. The full volume of the thesis is 111 pages, of which the list of sources occupies 2 pages, the appendices - 12 pages.

Relevance of the topic. With the spread of the Internet, the number of web applications began to grow rapidly. Creating a web service that communicates with external systems is not an easy task. The most common external connection is a database, so almost all web services use it to store data. As a result, the number of developers faced with the problem of database management and their integration into the created programs began to grow, because these processes require a large amount of time and knowledge, which can be irrational for small and medium-sized businesses, as well as own projects that are handled by one person or a small group. Because they usually require basic CRUD operations and some custom queries to fetch data. Popular solutions provide limited opportunities to solve this problem: inferior user interface; lack of support for user requests; lack of freedom in creating tables and external connections between them. As a result, it plays a key role in choosing the right solution for the end user. Therefore, there is an urgent need to implement new relational database developer tools that solve these problems.

The aim of the research. Creation of relational database developer tools that will provide customers with the ability to effectively manage databases and integrate queries to work with data. To achieve the goal, the following tasks were formulated:

- analyze existing software solutions: identify their advantages and disadvantages;
- conduct research on modern tools and technologies for development that will effectively solve the set goal;
- design architecture of the system and structure of databases;
- determine options for user interaction with the system;
- provide the user with the opportunity to create and manage databases, tables, columns using the UI;

- provide the user with the opportunity to interact with the data;
- implement the final software product using the research results;
- test and approve the system.

The practical value of the obtained results of the development of the software product is to provide the ability to create databases and manage them using the user interface, as well as the ability to integrate data management in tables using an API that allows you to perform CRUD and user queries, without spending a lot the amount of time and the need for additional knowledge to work with SQL. Created relational database developer tools can be useful for single or small group of developers, as well as small and medium-sized businesses that do not need a complex infrastructure around the database.

Publications.

The main provisions of the work were presented at:

1. Mishchenko A., Mikhailova I. Relational database developer tools. *Digitalization of science as a challenge of today*: materials of the VII International Student Scientific Conference, Poltava, October 25, 2024. Vinnytsia. Ukraine "UKRGOLOS Group", 2024. C. 584 -586.

2. Mishchenko A., Mykhailova I. Relational database developer tools // Universum Student Scientific Journal. 2024. No. 14. P. 284-288.

Keywords: tools, relational databases, CRUD operations, user requests, API.

ЗМІСТ

ВСТУП

Розвиток та популяризація Інтернет технологій спричинили різкий ріст кількості веб-систем, що почали створюватися на запити ринку. Вони дозволяють користувачу взаємодіяти з нею онлайн за допомогою з’єднання з будь-якого місця.

Оскільки кожна веб-система повинна як надавати інтерфейс для взаємодії, так і обробляти клієнтські запити, тому вона розділяється на клієнтську та серверну частини, що в свою чергу ускладнює її реалізацію. Найскладнішим у їх створенні є необхідність встановлювати додаткові з’єднання з зовнішніми системами, наприклад, базами даних.

Тому більшість розробників при реалізації веб-систем мають проблеми, що змушують їх займатися створенням, налаштуванням та подальшим управлінням і підтримкою баз даних. Даний процес вимагає від людини володіння достатньою кількістю знань та часу, щоб правильно виконати всі кроки зі створення серверної частини з зв’язком до бази даних.

Ця проблема є широко поширена серед більшості розробників: головною задачею яких є створення сайтів для власного використання, малого чи середнього бізнесу, які потребують виконання базових CRUD операцій з даними, а інколи й складних запитів на вибірку даних. У такому випадку витрачати час на реалізацію веб-серверну є недоцільно.

Згідно з цим, розробка веб-системи, що б надавала наступний функціонал: створення та управління базами даних через користувацький інтерфейс без необхідності написання SQL запитів; управління даними в таблицях за допомогою прикладного програмного інтерфейсу, – може стати ефективним способом для розв’язання даної проблеми.

Сучасний інформаційно технічний ринок не пропонує достатню кількість програмних продуктів, що здатні задовольнити потреби користувачів. Провівши аналіз популярних рішень для вирішення поставленої проблеми, було встановлено ряд недоліків, які вони мають: цінова політика, що є занадто дорогою для більшості користувачів; відсутність безкоштовного чи обмеженого функціоналу для

випробування можливостей продукту. Такі рішення як Firebase, Supabase, Hasura, Strapi є доволі популярними, та намагаються вирішувати поширені проблеми, однак вони також мають свої недоліки, які можуть стати критичними для користувачів: неповноцінна підтримка користувацького інтерфейсу з необхідністю власноруч писати SQL запити; обмеження свободи користувача при створенні структур таблиць, а також зовнішніх зв’язків між ними; відсутність складних користувацьких запитів з умовами для фільтрації чи з’єднання даних; перегляд даних через веб-сайт.

У результаті, реалізація нового рішення інструментів розробника реляційної бази даних становить важливу проблема, оскільки б забезпечила користувача важливим функціоналом, який відсутній у лідерів ринку, серед якого є: можливість зберігання та використання користувацьких запитів за допомогою API, а також виконання їх на веб-сайті для перевірки їх роботи; повноцінний інтерфейс для користувача, що дозволить створювати бази даних та їх вміст, а також переглядати дані в таблицях з можливістю виконання CRUD операцій для них; відсутність обмежень при створенні таблиць та зв’язків між ними.

1 ПОСТАНОВКА ЗАДАЧІ ІНСТРУМЕНТІВ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Мета даної роботи – реалізація інструментів розробника реляційної бази даних, що дозволять зручно та ефективно створювати та управляти базами даних та їх вмістом за допомогою користувацького інтерфейсу та API, забезпечивши надійність та безпеку даних.

У даному розділі також аналізуються сучасні програмні рішення, які покликані вирішувати схожі задачі, розглядаються їх переваги та недоліки, шляхи покращення користувацького досвіду з розроблюваними інструментами розробника реляційної бази даних.

1.1 Постановка задачі інструментів розробника реляційної бази даних

У результаті популяризації Інтернет технологій усе більше почало з’являтися систем, які працюють через Інтернет [1]. Це є досить зручно, оскільки для роботи з ними не потрібно нічого встановлювати, достатньо мати стабільне з’єднання та браузер.

Як правило такі веб-системи створюються за клієнт-серверною архітектурою, де клієнтська частина відповідає за взаємодію з клієнтом, а серверна у свою чергу – за зв’язок із зовнішніми системами, наприклад, базами даних. Тож під час створення таких веб-застосунків велика кількість розробників потребує налаштовувати та управляти базами даних. Однак реалізація серверної частини, яка відповідає за зв’язок з базою даних, потребує достатньо досвіду та часу, тому не всім під силу це завдання.

Описана проблема є найбільш актуальною для людей, що розробляють:

- веб-сайти чи системи для власного використання або для навчання;

- веб-застосунки, яким достатньо базових CRUD операцій: CRM системи, блоги та інше.

У цьому випадку, створення інструментів для роботи з реляційною базою даних, дозволило б вирішити дану проблему.

Метою даної роботи є розробка інструментів розробника реляційної бази даних, що дозволять користувачу зручно та ефективно управляти базами даних за допомогою інтерфейсу, виконувати операції з записами в таблиці за допомогою веб-інтерфейсу, та створювати власні запити до баз даних.

Виконання поставленої мети, вимагає розв’язання наступних завдань:

- проаналізувати існуючі програмні рішення: виявити їхні переваги та недоліки;

- провести дослідження сучасних засобів та технологій для розробки, які дозволять ефективно вирішити поставлену мету;

- спроєктувати архітектуру системи та структуру бази даних;

- визначити варіанти взаємодії користувачів із системою;

- надати можливість користувачу створювати та керувати БД, таблицями, стовпцями за допомогою інтерфейсу;

- надати можливість користувачу взаємодіяти з даними;

- реалізувати кінцевий програмний продукт, використовуючи результати досліджень;

- провести тестування та апробацію системи.

Розроблені інструменти розробника реляційної бази даних повинні мати такий функціонал:

- реєстрація та авторизація;

- управління базами даних;

- управління таблицями для кожної з баз даних;

- управління стовпчиками в таблицях;

- управління користувацькими запитами до баз даних;

- створення та оновлення авторизаційних токенів для баз даних;

- базові CRUD операції з даними в таблицях;

- прикладний веб-інтерфейс для інтеграції з різними системами, що дозволить виконувати базові CRUD та користувацькі запити.

1.2 Аналіз сучасних рішень інструментів розробника реляційної бази даних

Намагаючись полегшити життя творцям нових систем, які потребують підключення до бази даних, було створено чималу кількість рішень, що дозволяють спростити даний процес, дозволяючи керувати базою даних за допомогою різних API, а не написанням SQL запитів.

Деякі такі рішення дозволяють повністю відмовитися від написання скриптів на користь прикладних програмних інтерфейсів, деякі ж просто надають можливість управляти даними з уже розробленим власноруч сховищем даних. До таких продуктів можна віднести Firebase, Hasura, Strapi, Supabase.

Ці системи не є повністю універсальними. Кожна має свій функціонал, який розроблений для вирішення конкретних проблем найбільш ефективніше, ніж конкуренти.

Розглянемо ці програмні продукти більш детальніше та виділимо їх переваги та недоліки.

Firebase [2] – це веб-платформа, власником якої є компанія Google, яка покликана полегшити розробку та підтримку як мобільних, так і веб-застосунків. За роки свого існування зарекомендувала себе дуже позитивно та отримала широкий спектр можливостей для роботи з базами даних, аутентифікацією, хмарним сховищем, відправкою push-повідомлень та багато іншого. Інтерфейс програми зображено на рисунку 1.1.

Перевагами даної платформи є:

- хмарне зберігання даних;
- масштабованість;
- можливість створення резервних копій;
- підтримка Google SDK та REST API для інтеграції з користувацькими системами у більш зручному для клієнта форматі;
- легка інтеграція з іншими Google сервісами;
- висока швидкість роботи за рахунок потужностей компанії Google.

Недоліками є:

- відсутність підтримки реляційних баз даних, тому є підтримка лише NoSQL баз даних;

- необхідність створювати зв’язки між таблицями в ручну, оскільки звичної підтримки зв’язків, як в SQL базах даних, нема;

- відсутність підтримки складних запитів з умовами по типу з умовами WHERE чи JOIN.

Інтерфейс Firebase для роботи з базами даних зображений на рисунку 1.1.

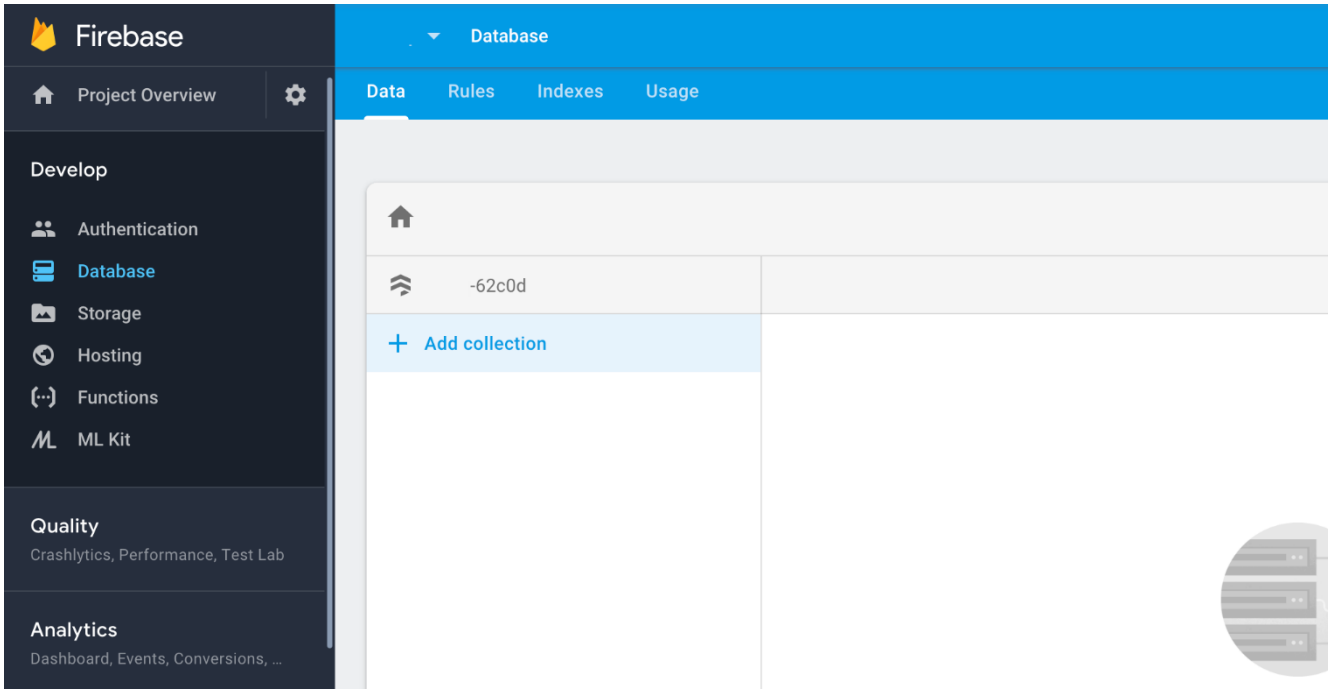

Рисунок 1.1 – Інтерфейс системи Firebase

Hasura [3] – це застосунок з відкритим кодом, що дозволяє створювати GraphQL API для роботи з даними в уже готовій PostgreSQL чи MongoDB базі даних, пришвидшуючи розробку нових систем у десятки разів.

Перевагами даного застосунку є:

- швидка та автоматична генерація GraphQL API для реляційних та нереляційних баз даних, що надає можливість виконувати запити будь-якої складності;

- можливість роботи з зовнішніми зв’язками через GraphQL API;

- інтеграція з будь-якими користувацькими системами через GraphQL API;

- підтримка гнучкого налаштування прав доступу користувачів для роботи з базами даних;

- можливість розгортання сервісу як на власному сервері, так і на Hasura Cloud.

До недоліків можна віднести:

- необхідність встановлення програми на власний ПК чи сервер для генерації GraphQL API;

- REST API не підтримується;

- створення баз даних, таблиць, зв'язків між ними повинне відбувається вручну розробниками через використання можливостей PostgreSQL чи MongoDB, оскільки даний інструмент генерує прикладний програмний інтерфейс лише для роботи з даними на готовій базі даних.

Інтерфейс Hasura для роботи з базами даних зображений на рисунку 1.2.

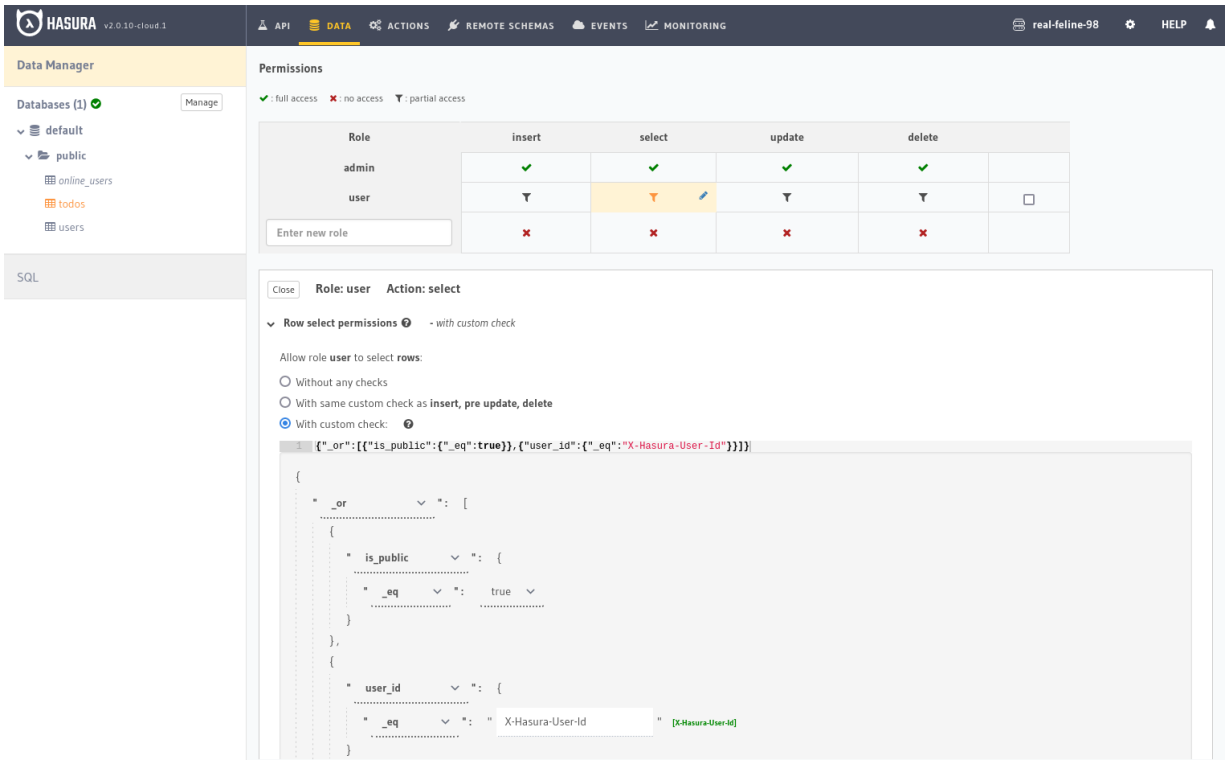

Рисунок 1.2 – Інтерфейс системи Hasura

Strapi [4] – це відкрита платформа для управління контентом (CMS). За допомогою неї розробники можуть створювати гнучкі API для власних веб-застосунків. Дане рішення фокусується на створенні серверної частини, на відміну

від готових веб-сайтів, та може бути використане з різними фреймворками для клієнтської частини: Angular, React, Vue та інші.

Перевагами даного рішення є:

- автоматична генерація REST API для інтеграції з клієнтськими частинами;

- можливість редагувати вихідний код для налаштувань системи за власними побажаннями;

- підтримка широкого спектру реляційних баз даних: PostgreSQL, MariaDB, SQLite, MySQL;

- підтримка широкого спектру як офіційних, так і користувацьких плагінів для розширення базових можливостей системи.

Недоліками є:

- необхідність займатися адміністративною роботою самотужки, оскільки для роботи з системою її необхідно розгортати на власному сервері;

- необхідність знання мови програмування Node.js для редагування вихідного коду;

- більшість базових функцій вимагає встановлення та налаштування плагінів.

Інтерфейс Strapi для роботи з базами даних зображений на рисунку 1.3.

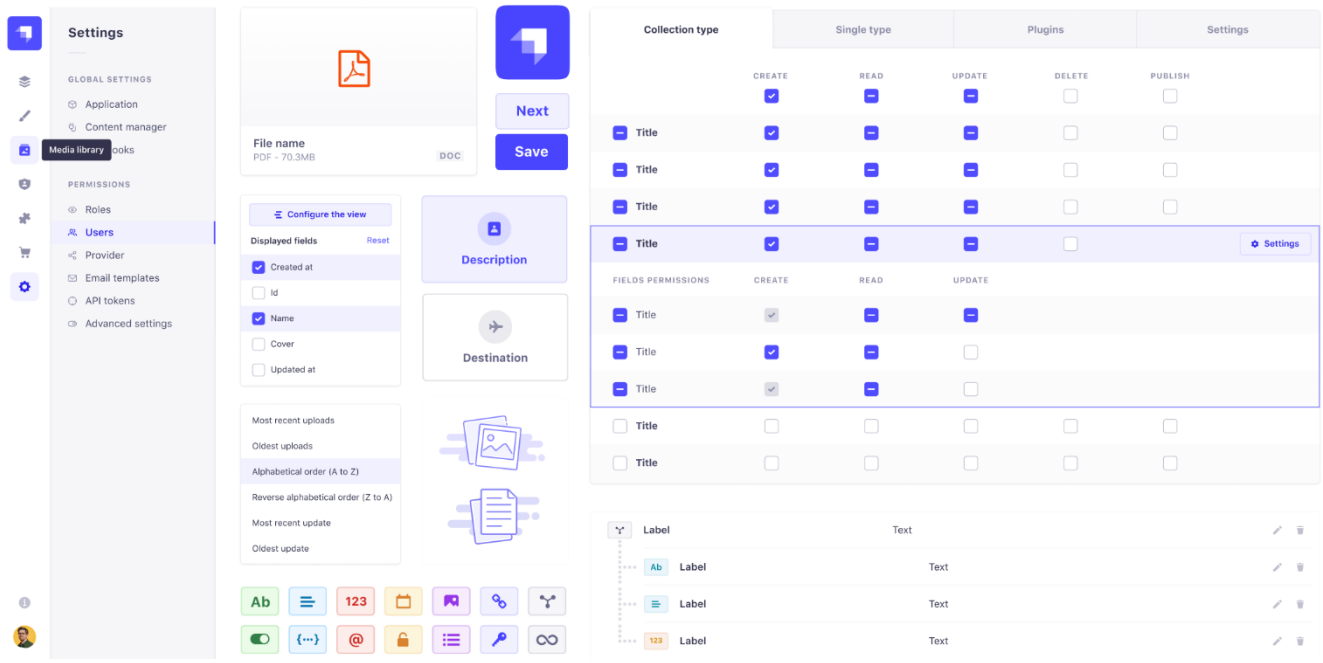

Рисунок 1.3 – Інтерфейс системи Strapi

Supabase [5] – це відкрита платформа, яка дозволяє розроблювати веб- та мобільні застосунки швидко та гнучко. За своїм функціоналом дещо схожа до Firebase, проте на відміну від Google альтернативи підтримує роботу лише з реляційною базою даних – PostgreSQL.

Інтерфейс Supabase для роботи з базами даних зображений на рисунку 1.4.

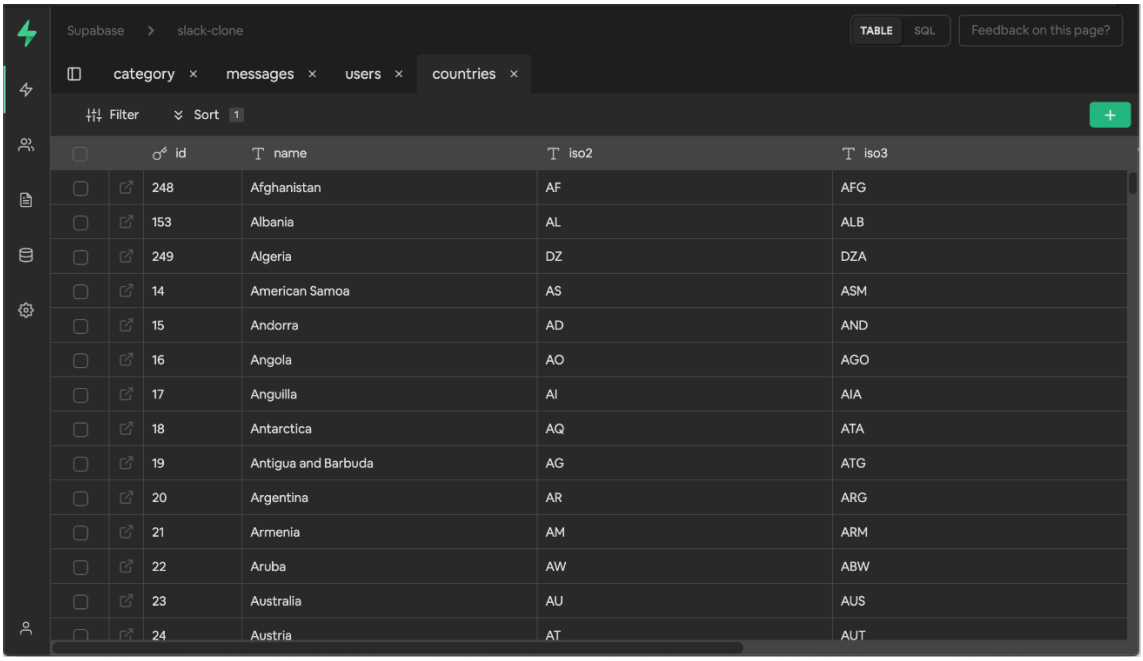

Рисунок 1.4 – Інтерфейс системи Supabase

Перевагами системи є:

- підтримка широкого спектру веб-інтерфейсів: REST API, GraphQL API;
- підтримка СУБД PostgreSQL;
- зручний інтерфейс для управління базами даних, таблицями та їх вмістом;
- можливість створювати зовнішні з’єднання за допомогою користувацького інтерфейсу;
- підтримка тригерів та процедур;
- перевірка доступу до ресурсів.

Недоліками системи є:

- відсутність гнучкості створення таблиці: усі вони створюються за замовчуванням з головним ключем (з назвою id);

- створення зв’язків між таблицями можливе лише за використання головного ключа, інші унікальні індекси не підтримуються;

- відсутність підтримки використання та зберігання власних SQL запитів для роботи з прикладними інтерфейсами.

Отже, розробка інструментів розробника реляційної бази даних, що дозволять зручно та ефективно створювати та управляти базами даних, таблицями та даними за допомогою користувацького та прикладного інтерфейсу є актуальною проблемою.

Проаналізувавши популярні рішення (Firebase, Hasura, Strapi та Supabase) можна прийти до висновку, що всі вони не є ідеальними та повністю універсальними, а також потребують певних удосконалень для покращення користувацького досвіду роботи з ними: створення інтерфейсу для роботи з базами даних та їх вмістом; надання можливості створювати на використовувати власні запити до таблиць; забезпечення повністю хмарної архітектури без необхідності розгортання та встановлення додаткових програм на власний сервер.

2 МЕТОДИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ ІНСТРУМЕНТІВ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Розробка програм розпочинається з підбору доступних на ринку технологій. Оскільки, це відіграє ключову роль у реалізації програмного продукту. Правильно підібрані інструменти полегшать та пришвидшать виконання поставленої мети. У цьому розділі розглядаються обрані технології для створення розроблюваної системи. Представлені засоби реалізації обиралися за наступними чинниками:

- сучасність, ефективність та доступність;
- безпечність та захищеність;
- актуальна та детальна документація;
- підтримка та інтеграція з обраними технологіями.

2.1 Spring як основний фреймворк для розробки веб-сервісу

Оскільки створення веб-сервісів мовою програмування Java часто вимагає виконувати одні і ті ж самі базові налаштування конфігурації та запуску, було прийнято рішення знайти інструмент, що б полегшив виконання поставлених задач.

Можливості відкритого Java API під назвою Jakarta дозволяють розробникам реалізовувати інтерфейси та створювати обгортки над класами, створюючи власні фреймворки, що в свою чергу дозволяє зменшити написання повторюваного коду. Найбільш популярним та гнучким рішенням на ринку зараз є Spring Framework. Був створений у 2002 році та підтримується і покращується дотепер, отримуючи оновлення старих та реалізацію нових модулів, що дозволяють вирішувати сучасні проблеми швидко та ефективно.

На сьогодні даний фреймворк дозволяє вирішувати наступні задачі: виконувати зв’язування об’єктів; підтримка сучасних баз даних; налаштовувати автентифікацію всіма популярними методами; підтримка конфігураційних

профілів та сервер для їх збереження та інше. Перелік модулів, що входять до Spring Framework можна побачити на рисунку 2.1.

Рисунок 2.1 – Модулі фреймворку Spring

Одною з ключових особливостей даного фреймворку є те, що він дозволяє перекласти обов’язки зі створення та зв’язування об’єктів з розробника на себе [9]. Програміст лише має вказати в конфігураційних файлах, що йому необхідно під час використання програми, і куди створений об’єкт має бути вставлений, а Spring зробить усю роботу за нього. Що в результаті дозволить зменшити кількість написаного коду, покращивши його читання та розуміння, а також дозволить зекономити час під час розробки та подальшої підтримки за рахунок гнучкої конфігурації.

Даний механізм є реалізацією патерну програмування, що називається ін’єкція залежності (англійською Dependency Injection). Метою якого є перекладання відповідальності за управління життєвим циклом об’єктів, що необхідні для правильної зв’язаних компонентів, на певний процес. У Spring це реалізовано за допомогою так званого Spring Context. Це звичайний контейнер, який зберігає в собі всі об’єкти, що були помічені розробником у конфігурації, як Bean. Фреймворк реалізовує три найпопулярніші механізми зв’язування компонентів, а саме через конструктор, метод та властивість. Даний процес у спрощеному представлені зображено на рисунку 2.2, він дає в певній мірі зрозуміти те, як відбувається зв’язування об’єктів між собою у Spring Context.

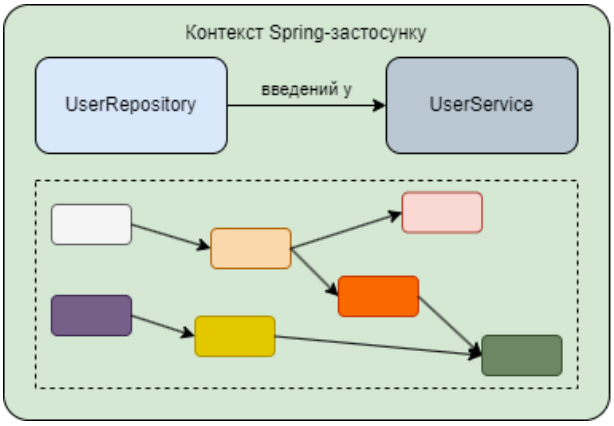

Рисунок 2.2 – Спрощене представлення зв’язування компонентів у Spring Context

До ще одної ключової особливості даного фреймворку можна віднести модуль, який має назву Spring Boot. Його можна умовно назвати обгорткою над звичайним Spring Framework, який покликаний спростити конфігурацію базових та рутинних для розробника завдань. Даний модуль реалізовує наступний функціонал [10]:

- підтримка автоматичної конфігурації для популярних систем та бібліотек: підключаючи залежність, що підтримується Spring Boot, розробник отримує підключенні та налаштування базових конфігурацій, що створюють необхідні компоненти для роботи з зовнішніми інструментами;

- пакети бібліотек: розробка великих систем зазвичай потребує підключення великої кількості бібліотек, за часту вони залежні одна від одної та потребують правильного підбору версій, щоб програма працювало злагоджено, тому в Spring Boot були створені пакети бібліотек, які пов’язані за одною темою і дозволяють програмісту спростити задачу підбору, підключення та налаштування залежностей;

- вбудований Tomcat сервер: до появи Spring Boot програмісти повинні були власноруч займатися скачуванням серверів та розгортанням на них своїх веб-сервісів, проте зараз достатньо підключити необхідну залежність, вказати параметри запуску, а всю іншу роботу зробить Spring;

Враховуючи представлені переваги даного фреймворку й було обрано його за основу для розробки інструментів розробника реляційної бази даних.

2.2 Фреймворк Angular

Під час пошуку інструменту, що б міг спростити та пришвидшити розробку мовою програмування JavaScript, було проаналізовано два найпопулярніші, що покликані полегшити розробку клієнтської частини: Angular та React. Кожен використовується провідними компаніями світу, має високі показники по швидкодії, підтримку різних зовнішніх систем та бібліотек, гарну документацію.

Проте для розробки інструментів розробника реляційної бази даних було обрано саме фреймворк Angular, оскільки він має ряд переваг над React, що розглядаються нижче [11].

Angular є повноцінним фреймворком, який містить підключені та налаштовані базові залежності, тому для повноцінної розробки достатньо встановити його використовуючи пакетний менеджер NPM, ніяких додаткових конфігурацій робити не потрібно. У той же час React є бібліотекою, яка використовується для побудови інтерфейсів, а тому для повноцінної розробки необхідно виконувати підключення додатких залежностей та проводити налаштування.

Angular має підтримку модульності, що дозволяє розділяти зв’язувати пов’язані між собою компоненти та відокремлювати не пов’язані. Це дозволяє використовувати його для побудови великих систем та масштабуватися за необхідності. З іншої сторони React повністю протилежний до цього. Він не має суворих правил до архітектури, тому розробка складних проектів у великих командах потребує додаткових ресурсів для перевірки дотримання стандартів кожним із розробників, щоб у подальшому не виникало проблем із масштабованістю.

Робота з формами в Angular, на відміну від React, підтримується за замовчування за допомогою включених модулів Forms Module та Reactive Forms Module, які в свою чергу мають підтримку перевірки введених даних на валідність. При роботі з React реалізовувати або шукати та встановлювати необхідні бібліотеки необхідно самому.

Angular розроблений з використанням мови програмування TypeScript. Це автоматично унеможливлює розробника допустити помилок з типізацією об’єктів під час роботи програми, оскільки всі проблеми будуть відображені на етапі компіляції. Також використання статичної типізації робить код більш кращим для розуміння та читання. React у свою чергу також може працювати з TypeScript, проте він не встановлений там за замовчуванням і не вимагає цього від програмістів, що може призвести до певних проблем у подальшій роботі веб-сайту.

До інших можливостей фреймворку Angular можна віднести: робота з роутингом; вбудований HTTP клієнт для відправки запитів на сервер, який має велику кількість налаштувань; можливість керувати станами. Усі ці можливості підтримуються за замовчуванням та не потребують додаткових бібліотек.

Усе це й робить даний фреймворк найкращим рішенням для розробки інструментів розробника реляційної бази даних.

2.3 СУБД InterSystems IRIS як постійна база даних

Розроблювані інструменти розробника реляційної бази даних містять в собі дані про наступні сутності: користувачі, бази даних, таблиці, запити до баз даних, авторизаційні токени для доступу до баз даних. Ці дані мають спільні риси, до яких можна віднести: чітко визначену структуру; зв’язок один з одним за конкретними характеристиками; мають зберігатися на постійній основі. Виходячи з цього можна визначити вимоги до сховища даних: стабільність та надійність при збереженні даних; можливість створювати чітко визначені сутності; підтримувати збереження великої кількості інформації та швидкому пошуку в ній по ключовим параметрам за необхідності.

Сучасний ринок баз даних досить великий. Кожна із яких покликана вирішувати конкретні задачі, через це вони поділяються на два різних підходи до збереження даних: реляційний (SQL) та нереляційний (NoSQL). До SQL систем управління базами даних можна віднести: InterSystems IRIS, MySQL, PostgreSQL. До NoSQL – Redis, MongoDB, Elasticsearch.

Реляційний та нереляційний підходи до збереження даних використовують різні структури даних, що дозволяють виконувати ефективніше конкретні операції з інформацією. Ще одною ключовою різницею між такими сховищами є те, що в реляційних СУБД використовується єдина стандартизована мова – Structured Query Language, на відміну від NoSQL, де може використовувати такі мови програмування, як: ObjectScript, JavaScript, C++.

Тож за рахунок цього різні підходи мають свої власні характеристики, для SQL притаманні наступні:

- сутності представлені у вигляді структурованих таблиць;
- дозволяють зберігати великі масиви інформації;
- усі СУБД використовують одну єдину мову SQL, за винятком певних відмінностей;
- можливість налаштовувати рівні ізоляції даних під час виконання запитів;
- можливість виконання транзакцій;
- підтримка правил ACID;
- мають різні види нормалізації даних.

У свою чергу для NoSQL характерно наступне:

- дані не є структурованими;
- мають високу швидкість для запитів на читання;
- добре підходять для кешу;
- підтримка теореми CAP;
- гнучкість та масштабованість.

Проаналізувавши особливості кожного з підходів до збереження та управління інформації, було встановлено, що для успішного виконання поставленого завдання по надійному та ефективному збереженні постійних даних у системі – найкращим рішенням є використання реляційної СУБД. Вибір був зроблений через те, що такі бази даних реалізовують властивості ACID, а виконання даних правил є критично важливим для розроблюваної системи.

У свою чергу, основною СУБД було обрано InterSystems IRIS. Конкурентами для неї є MySQL та PostgreSQL, які є популярними та широко розповсюдженими.

Усі вони забезпечують надійну та стабільну роботу з великими обсягами даних, кожна з них має свої сильна та слабкі характеристики. Однак, InterSystems IRIS має деякі переваги над ними [12]:

- підтримка скриптової мови програмування ObjectScript, що значно розширює базові можливості бази даних;
- розширені можливості з управління доступами до баз даних для різних користувачів;
- індексування та пошук великих даних реалізовано більш краще.

Врахувавши всі ці ключові переваги й було обрано SQL, оскільки даний підхід має переваги по збереженню структурованої інформації, зв'язуванню сутностей одна з одною та підтримкою ACID властивостей. Головною СУБД було обрано InterSystems IRIS, бо має розширені можливості з управління доступами до баз даних, які є ключовими при розробці інструментів розробника реляційної бази даних.

2.4 Використання Redis для збереження авторизаційних токенів

Окрім описаних постійної інформації з якої взаємодіє система, існують і не постійні дані: JSON Web Token користувачів. Їх тривалість життя в системі становить одну добу з моменту їх створення. Також вони потребують швидкого доступу до них, оскільки розроблювана система під час кожного запиту клієнта до неї – повинна перевіряти достовірність наданого токену.

Тому для збереження даної інформації було обрано нереляційну систему управління базами даних Redis. Redis – це NoSQL сховище даних, яка зберігає інформацію в оперативній пам'яті [13]. За рахунок цього є найкращим вибором для частого виконання простих запитів з читання, створення, редагування та видалення інформації. Проте оперативна пам'ять є дорогою, тому кількість інформації, яка має зберігатися в такій БД має бути не дуже великою. До переваг даної СУБД також можна віднести наступне:

- підтримка п’яти різноманітних структур із збереження даних;

- легко масштабується;

- безкоштовний доступ до офіційного онлайн сховища Redis Cloud.

Одним із основних та популярних, швидких та простих структур даних для збереження інформації є хеш. Вона має такі характеристики:

- складається з множини полів та значень, які належать їм належать;

- асоціативна колекція;

- значення пов’язані з унікальними ключами, за рахунок цього досягається максимальне опрацювання простих запитів до даних, наприклад, читання та зберігання.

Умовне зображення збереження даних у структурі хеш у спрощеному вигляді можна побачити на рисунку 2.3.

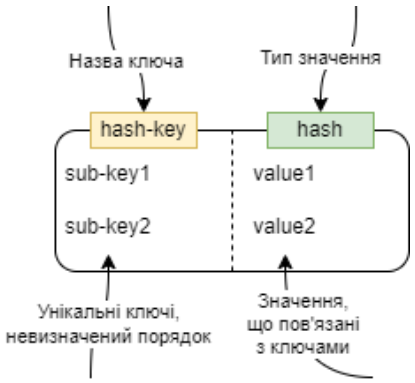

Рисунок 2.3 – Умовне зображення збереження інформації у хеші

Дана структура забезпечить швидке виконання частих запитів читання та запису, видалення записів із бази даних, яка розміщена в оперативній пам’яті. Через це й було зроблено вибір на користь даного підходу, оскільки дозволяє ефективно виконувати поставлені перед ним завдання.

2.5 REST API – інструмент взаємодії з даними

Оскільки інструменти розробника реляційної бази даних прийнято будувати у вигляді веб-системи з використанням клієнт-серверної архітектури, де система розділяється за відповідальністю на дві частини та потребує забезпечення

спілкування між ними, це вимагає використання єдиного формату спілкування між ними, тобто прикладного програмного інтерфейсу (API).

Також, оскільки розроблювана система має забезпечувати інтеграцію з користувацькими програмами з можливістю виконання різних запитів до даних у таблицях, то вона має реалізовувати API, що б виконувало поставлену задачу швидко та ефективно.

API називають набір інструкцій, які визначені одною структурою за чіткими правилами та використовуються для взаємодії різних програм між собою. У випадку взаємодії застосунків в Інтернеті існує декілька найбільш популярних та широко використовуваних прикладних програмних інтерфейсів. До таких можна віднести: REST, SOAP, gRPC, GraphQL. Кожен із яких був розроблений у свій період історії, щоб ефективно виконувати поставлені перед ним задачі.

Представлені API працюють за допомогою HTTP протоколу [14]. Протокол передачі гіпертексту (Hyper Text Transfer Protocol) – це мова, що дозволяє браузеру користувача чи програмам спілкуватися з різними веб-системами за допомогою Інтернет з’єднання. При взаємодії встановлюється захищене з’єднання по якому передаються всі необхідні дані, доступ до нього мають лише дві сторони. Коли обмін даними закінчується, то з’єднання закривається. Спрощений принцип взаємодії клієнта та сервера за допомогою HTTP зображений на рисунку 2.4.

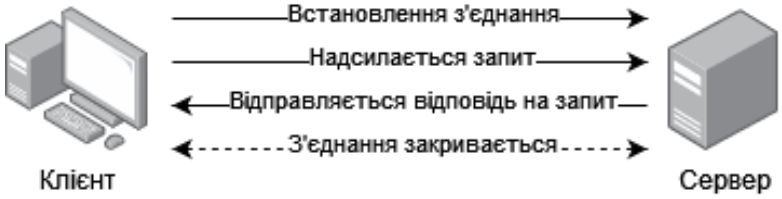

Рисунок 2.4 – Взаємодія клієнта та серверу за допомогою HTTP

Прикладний програмний веб-інтерфейс використовується ж на серверній частині. Він потрібний для опису можливих варіантів запитів на сервер та відповідей клієнту на ці запити, умовне зображення використання API на сервері зображено на рисунку 2.5.

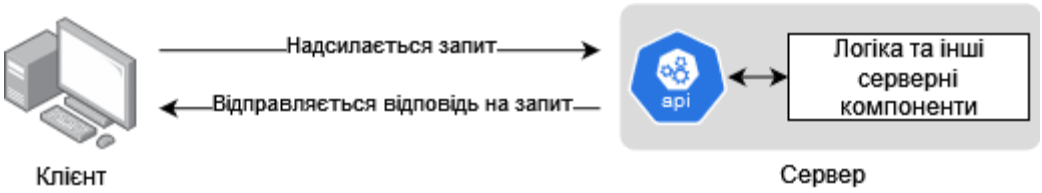

Рисунок 2.5 – Використання API на сервері

Серед найпопулярніших веб-інтерфейсів було обрано REST, оскільки він має набагато більше переваг на іншими та досить гарно підходить для виконання поставлених задач: відокремленню клієнтської та серверної частини, простоті та ефективності використання. До переваг REST можна віднести [15]:

- на відміну від gRPC, який вимагає сильний взаємозв'язок між клієнтом і сервером, не дозволяючи повноцінно відокремити клієнтську та серверну логіку, має слабкий взаємозв'язок, дозволяючи змінювати ту чи іншу частину без обов'язкової зміни іншої;

- є сучасним та легко реалізується на відміну від SOAP, який був розроблений досить давно, через що має багато складного та зайвого функціоналу, занадто складну стандартизацію: обов'язкове використання xml структури для обміну інформацією;

- не вимагає складної логіки та багато ресурсів від сервера на відміну в GraphQL, який може значно навантажити систему за рахунок великої кількості різноманітних та унікальних запитів;

- дозволяє використовувати великий спектр форматів для обміну даними, а найголовніше найпопулярніший та найпростіший JSON;

- підтримка кешування запитів, масштабованості;

- підтримка HTTP методів;

- легко інтегрується з будь-якими клієнтськими частинами та системами за рахунок своєї незалежності від мови програмування.

Виходячи з цих усіх переваг і було зроблено вибір у сторону даного прикладного програмного інтерфейсу для взаємодії клієнтської та серверної частини інструментів розробника реляційної бази даних. Також його було обрано

за основу розробки універсального API, щоб дозволити користувачам легко інтегрувати інструменти з власними системами для взаємодії з даними у таблицях.

2.6 Забезпечення безпеки та гнучкості за допомогою токенів

Інструменти розробника реляційної бази даних мають забезпечувати безпеку користувацьких даних, а тому повинні реалізовувати механізм, що дозволить вирішити дану задачу. Доступ до кінцевих точок серверу з клієнта має бути конфіденційним та захищеним. Авторизований користувач повинен мати можливість переглядати та керувати даними, які йому дозволені та не більше. Додатковим критерієм до обрання механізму: повинен інтегруватися з REST API без складнощів.

Оскільки система дозволяє користувачам керувати інформацією в таблицях з різних баз даних, то потрібно підібрати такий механізм, який дозволить створити унікальні ключі доступу для надійної та легкої ідентифікації до якого саме сховища направлений запит.

У сучасному світі Інтернет технологій існує декілька підходів, які дозволяють забезпечити безпеку користувацьких даних у системі. До найпопулярніших таких підходів можна віднести: автентифікація за допомогою сесії та токенів.

Механізм робити кожного з цих підходів відрізняється один від одного. У випадку сесії, коли користувач авторизується на сервері, то йому створюється власна унікальна сесія, яка зберігається на сервері, для неї присвоюється унікальний ідентифікатор, що відправляється на клієнт, який має зберегти його в кукі (cookie) браузера, і при кожних наступних запитах до них має прикріплюватися даний ідентифікатор, за яким сервер буде визначати чи надавати користувачу доступ до інформації. Цей підхід не є безпечним, оскільки легко cookie досить легко вкрасти, через що користувацька інформація може бути скомпрометована.

Через це було знайдено рішення, і зараз найбільш поширеним способом автентифікації користувача в системі є токени. Працює він наступним чином, при авторизації користувачу створюється не сесія, а токен, який може зберігатися на сервері, проте це не є обов'язковим. Після створення токену він передається клієнту, який може зберегти його в локальному сховищу чи cookie браузера, що не є безпечним варіантом, проте для забезпечення надійності клієнт має зберегти токен у пам'яті чи використовуючи інші захищені місця, які реалізовують популярні фреймворки, наприклад, Angular.

Ще одною перевагою токенів над сесіями є те, що в нього можна вкласти будь-яку інформацію, що може бути використана в майбутньому для перевірки доступу.

Виходячи з цієї інформації найкращим способом для реалізації механізму автентифікації в інструментах розробника реляційної бази даних є токени. Проте стандартів їх створення існує багато. Найпопулярнішим є JSON Web Token (JWT) та OAuth 2.0. У першому випадку токен створюється сервером, для цього можна використовувати будь-який механізм кодування та вкладання будь-якої інформації. У другому – створюється третьою стороною, наприклад, Google, Facebook, користувач авторизується використовуючи свій акаунт в соціальній мережі, яка в свою чергу перевіряє надану інформацію і в разу успішної автентифікації надає токен, що може бути використаним для майбутніх взаємодій з системою.

Оскільки нам потрібно забезпечити можливість вкладання будь-якої інформації в токен та кодувати її за власними механізмами, то найкращим варіантом є JWT.

JSON Web Token [16] – є відкритим стандартом (RFC 7519), що забезпечує безпечне представлення правовідносин між двома сторонами. Задачею даного стандарту є опис самодостатнього та компактного підходу для виконання умов безпечного обміну даними між програмами. Закодовані дані представляються в JSON форматі. До переваг даного стандарту можна віднести:

- уся інформація необхідна для виконання автентифікації знаходиться в закодованому вигляді в самому токені;

- можливість передачі токену між сторонами будь-яким доступним способом: за допомогою HTTP заголовку, URL параметру та інших;

- не потрібно зберігати в cookies;

- у токен можна вкласти будь-яку інформацію, що потрібна в системі, яку можна розкодувати за допомогою секретного паролю;

- легка інтеграція з REST API.

JWT складається з трьох головних частин, кожна з яких використовується для збереження потрібної інформації, що дозволяє встановлювати ідентичність та правильність токену. До цих частин відноситься:

- заголовок – містить інформацію про тип токену та алгоритм кодування, яким було його створено;

- корисні дані – знаходиться довільна інформація, наприклад, ідентифікатор, ім’я та роль користувача, дана інформація може бути розкодована та використана в системі для будь-яких цілей;

- підпис – створюється за допомогою додавання перших двох закодованих частин (заголовок та корисні дані), це використовується для того, щоб перевірити чи не були дані змінені протягом життєвого циклу токену.

Для більш наглядного прикладу того з яких частин складається даний стандарт токену – було створено діаграму, що зображена на рисунку 2.6.

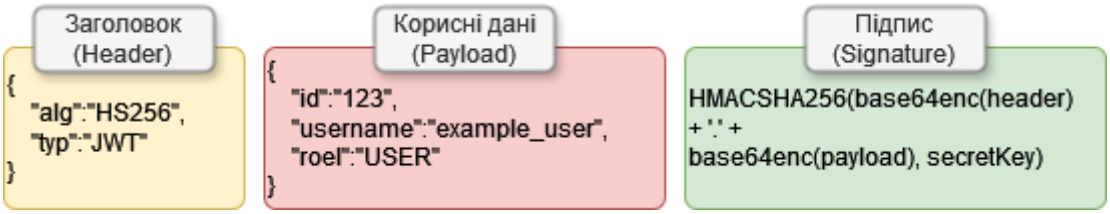

Рисунок 2.6 – Структура JWT

Ще одною перевагою цього стандарту є те, що він повністю підтримується модулем Spring Security та фреймворком Angular. Перший дозволить швидко на ефективно налаштувати генерацію токенів та їх перевірку на серверній частині, а другий забезпечить надійне зберігання їх на клієнтській частині та відправці при кожному запиті до серверу.

Враховуючи всі ці наведені переваги й було зроблено вибір у сторону підходу автентифікації з використанням токенів, а саме найбільш популярного, простого та гнучкого стандарту JWT. Оскільки дозволить вкласти в нього всю потрібну інформацію для забезпечення захисту обміну запитами між клієнтською та серверною частинами системи, а також дозволить генерацію унікальних токенів для кожної бази даних та їх перевірку при взаємодії користувачів із даними в таблицях за допомогою REST API.

2.7 Інструмент для розгортання компонентів системи – Docker

Інструменти розробника реляційної бази даних складаються з чотирьох окремих компонентів: серверна та клієнтська частини; системи управління базами даних InterSystems IRIS та Redis. Їх потрібно розгортати локально та віддалено, використовуючи одні параметри запуску. Також необхідно слідкувати за станом роботи системи, і в разі якоїсь нестандартної ситуації чи помилки виконувати певний алгоритм дій, щоб уся система не впала.

Для вирішення даної проблеми було створено платформу Docker [17], яка дозволяє розробникам створювати віртуальні контейнери з різними операційними системами, запускаючи на них програми, які виконуються виключно в одному контейнері і не впливають на роботу інших контейнерів чи самого сервера, на якому їх було запущено. Для запуску Docker використовує операційну систему Linux, через це всі контейнери запускаються на ній, оскільки вони використовують ядро головної системи. Для написання скриптів необхідно використовувати спеціальні Docker команди. Умовний приклад контейнерів у Docker зображено на рисунку 2.7.

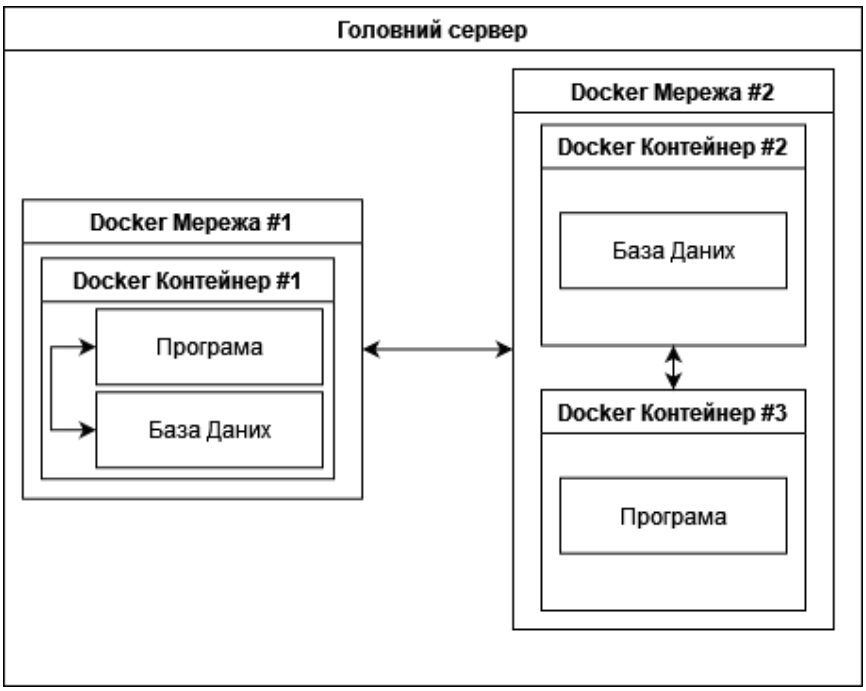

Рисунок 2.7 – Спрощене представлення роботи контейнерів у Docker

З рисунку 2.7 також видно основні можливості Docker:

- запуск декількох програм в одному контейнері;
- налаштування зв’язку між програмами в контейнерах;
- створення Docker мереж та налаштування зв’язку між ними;
- об’єднання контейнерів у окремі Docker мережі.

Проте окрім цього Docker має набагато більше переваг, які спрощують розгортання та підтримку програм на різних комп’ютерах, до них можна віднести:

- оскільки Docker для запуску використовує ядро Linux, то скрипти для запуску контейнерів можна виконувати на комп’ютерах з різними операційними системами (Linux, Windows, MacOS) без необхідності вносити якісь зміни;

- усі файли, які програми створюють під час своєї роботи за замочуванням зберігаються в тимчасовій пам’яті, яка очищується автоматично після видалення самого контейнера або ж вручну;

- програми в контейнерах запускаються в ізольованому середовищі, тому нічого на комп’ютері не вплине на їх роботу, також запущені застосунки за замовчуванням не мають доступу до комп’ютера та локальної мережі;

- можливість налаштування алгоритму дій при збою: сповіщення, спроба аварійного перезапуску, запуск запасних контейнерів;

- існує велика кількість готових контейнерів з базами даних, мовами програмуваннями, що дозволяє швидко піднімати потрібні контейнери без складнощів;

- Docker Compose для спрощеного запуску мультиконтейнерних програм [18];

- вертикальне масштабування за рахунок запуску додаткових контейнерів з потрібними програмами.

Враховуючи ці всі можливості Docker, і було зроблено вибір його за основний інструмент, який дозволить запускати систему та бази даних для тестування та запуску на сервері. Оскільки він дозволяє використати готові контейнери для запуску Java та JavaScript частин програми, а також баз даних InterSystems IRIS та Redis, надавши серверу доступ до них. Для цього можна написати один скриптовий файл та використовувати локально та віддалено, в якому буде передбачено алгоритм запуску та план дій на екстрений випадок.

Отже, основною мовою програмування для написання серверу було обрано Java через ряд переваг над іншими мовами: багатопотоковість; компіляцію в байт код, який можна запускати на будь-якій операційній системі; відкриті інтерфейси для роботи з базами даних та розробки веб-застосунків, що в результаті дало змогу створити багато різноманітних фреймворків, які пришвидшують розробку програм.

Щоб спростити реалізацію веб-серверу було вирішено використати фреймворк Spring Framework. Він дозволяє зменшити написання повторюваного коду, перекласти відповідальність за створення об’єктів та їх управління з розробника на себе, має компоненти для роботи з реляційними та нереляційними базами даних, REST API та налаштуванням захисту для його кінцевих точок.

Для розробки сторінок для взаємодії з системою було обрано мову розмітки HTML, що дозволить додати основні компоненти. Для налаштування стилів використано таблицю стилів SCSS з фреймворком Bootstrap, який дозволить зручніше налаштовувати зовнішній вигляд, використовуючи готові стилі.

Було використано фреймворк Angular, щоб спростити написання JavaScript коду. Оскільки він має повноцінний вбудований функціонал для роботи з: формами, з HTTP-клієнтами для відправки запитів на сервер, мовою програмування TypeScript для статичної типізації об’єктів, налаштування захисту сторінок. Окрім цього дозволяє розділяти групи сторінок на компоненти, що дає змогу створювати масштабовані проєкти.

Система під час свого життєвого циклу взаємодіє з даними, які було розділено на дві групи. Перша група це постійні дані про: користувачів, бази даних, таблиці, користувацькі запити до БД, авторизаційні токени доступу до БД, – вони мають чітко визначену структуру та мають зберігатися протягом усього часу. Інша група це тимчасова інформація (кеш) про актуальні JSON Web Token авторизованих користувачів у системі, вони не є структурованими, до них часто звертається сервер, тому потрібно забезпечувати їх швидкий запис та читання.

Для виконання поставлених умов по зберіганню даних було використано дві системи управліннями базами даних: InterSystems IRIS та Redis. Перша є реляційною та має зберігати постійні дані, має ряд переваг на альтернативами: розширений доступ управління доступами до БД; покращене індексування та пошук у великих обсягах даних. Друга є нереляційною та має зберігати тимчасові дані, оскільки має тип даних hash, який забезпечує швидкий запис та читання інформації з нього.

Зв’язок між клієнтською та серверною частинами системи та реалізацію універсального API для взаємодії клієнтів з даними в сховищах було прийнято робити за допомогою RESTful веб-інтерфейсу. Є популярним та зручним рішенням для встановлення правил запитів до сервер та відповідей з нього, має чітко визначену структуру даних, підтримує кешування, не потребує великої кількості ресурсів від серверу, дозволяє повністю розділити клієнтську та серверну логіку, працює за протоколом HTTP. Повністю підтримується фреймворками Angular та Spring.

Захист кінцевих точок REST API вирішено реалізувати за допомогою компоненту Spring фреймворку – Spring Security. Використовуючи відкритий

Інтернет стандарт JSON Web Token. Він дозволяє шифрувати всю необхідну інформацію в середині токену, легко отримувати її за допомогою потрібного секретного паролю для подальшого використання в середині програми – це дасть змогу використовувати токен для авторизації користувачів та визначення доступу до баз даних при використанні розробленого універсального веб-інтерфейсу.

Розроблювані інструменти розробника реляційної бази даних складаються з чотирьох основних компонентів. Для спрощення розгортання та налаштування їх було вирішено використовувати платформу Docker, що спрощує цей процес за рахунок використання готових контейнерів з використовуваними мовами програмування (Java, JavaScript) та базами даних (InterSystems, Redis). Для потрібно написати спеціальний скрипт, який можна однаково виконувати як локально, так і віддалено на будь-яких операційних системах. Скрипт передбачає можливість опису алгоритму дій на випадок збою.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ІНСТРУМЕНТІВ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Створення надійної, ефективної та масштабованої архітектури системи, бази даних, реалізація швидких програмних алгоритмів є важливим кроком у розробці будь-якої системи.

3.1 Архітектура веб-системи інструментів розробника реляційної бази даних

Архітектурою програмної системи називають план побудови розроблюваної системи, ідея якої полягає в тому, щоб описати взаємодію та організацію компонентів у ній на високому рівні абстракції, не вдаючись у деталі реалізації кожного з них. Під час побудови архітектури ставиться за мету розробити її масштабованою, надійною та стабільною, логічною, простою в реалізації. Даний етап є дуже відповідальним, оскільки по його закінченню та початку реалізації зміни в архітектуру буде вносити набагато складніше, тому йому потрібно приділити багато уваги, проаналізувавши різноманітні варіанти.

У сучасному світі існує велика кількість різноманітних підходів до побудови архітектури для власної системи. Кожен такий шаблон вирішує різноманітні завдання, тому, обираючи якийсь із них для побудови програмного продукту, потрібно чітко визначити задачі, які він має вирішувати [19].

Було проведено дослідження достатньої кількості шаблонів, які призначені для вирішення поставленої задачі: розробка інструментів розробника реляційної бази даних. Реалізацію веб-системи було прийнято робити використовуючи клієнт-серверну архітектуру, розділивши її на клієнтську та серверну частини [20], результат зображений на рисунку 3.1.

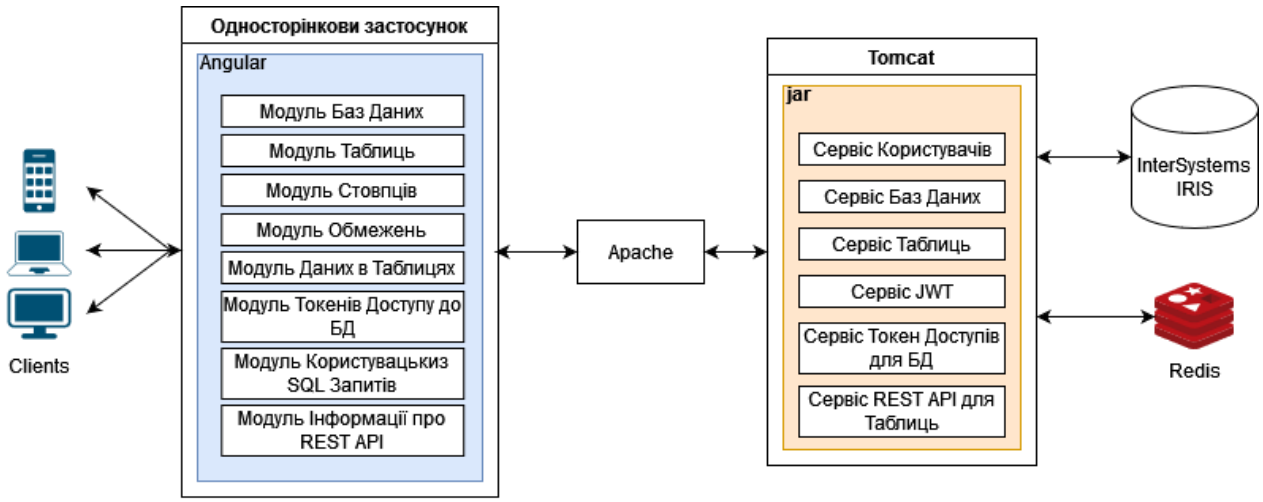

Рисунок 3.1 – Клієнт-серверна архітектура для інструментів розробника реляційної бази даних

Розробку клієнтської частини було вирішено робити використовуючи односторінковий застосунок (Single Page Application) з використанням архітектури низького рівня: компонентно-орієнтованої. Вона є ефективним рішенням для побудови великих та масштабованих веб-сайтів, оскільки дозволяє розділити всі елементи на компоненти, які об’єднуються в модулі за спільними особливостями. Результат даної архітектури для розроблюваною клієнтської частини зображено на рисунку 3.2.

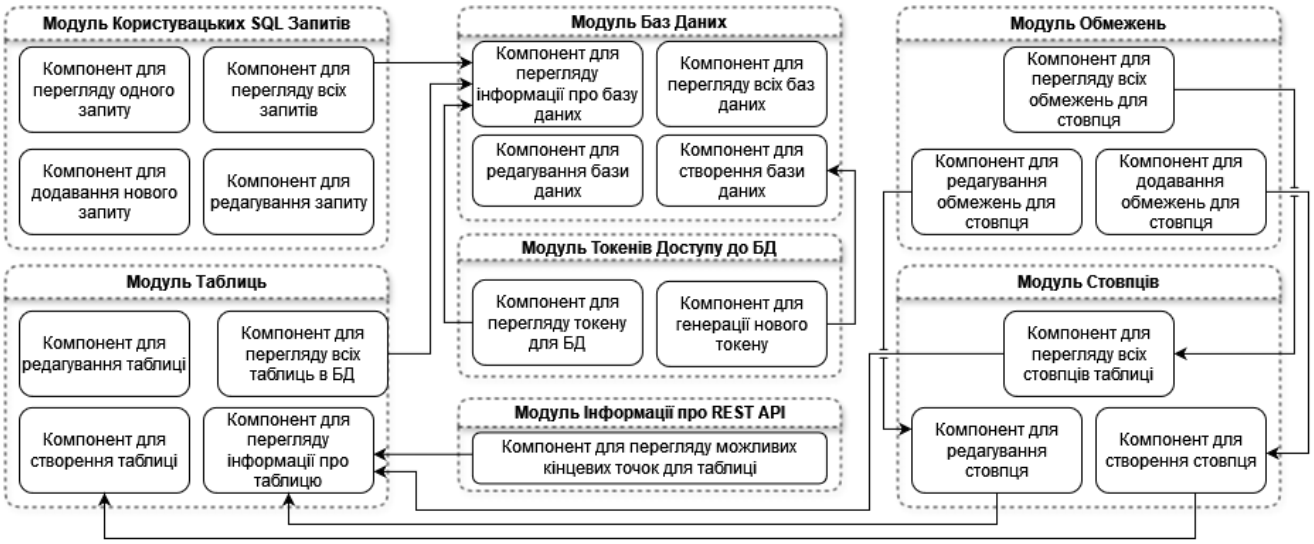

Рисунок 3.2 – Компонентно-орієнтована архітектура клієнтської частини інструментів розробника реляційної бази даних

Серверну відповідно – за допомогою багаторівневої архітектури (Multitier architecture). Вона у свою чергу дозволяє відділити різні типи логіки (представлення, бізнес, доступу до даних) один від одного для визначення чітких кордонів. Дана діаграма знаходиться на рисунку 3.3.

Рисунок 3.3 – Багаторівнева архітектура клієнтської частини інструментів розробника реляційної бази даних

Використання описаних архітектур дало можливість організувати компоненти системи ефективно, розділивши їх повноваження та чітко визначивши межі відповідальності.

3.2 Діаграма класів системи інструментів розробника реляційної бази даних

Архітектура інструментів розробника реляційної бази даних відображає верхній рівень абстракції, проте окрім нього в системі існує нижній. Він складається з одинадцяти основних класів, які використовуються для роботи системи. Вони є спільними як для серверної, так і клієнтської частини, щоб забезпечити повну сумісність програмного коду для правильного виконання.

Для компонування та виділення компонентів на низькому рівні було використано діаграму класів. Нею називають схематичне представлення на якому зображені структурні елементи (класи), що використовуються в програмному забезпеченні. Найчастіше класи ділять на три головних типи:

- границя – це якась вхідна чи вихідна форма з якою може взаємодіяти користувач чи інша система;

- контролер – це елемент через який сутності потрапляють з форми в систему, чи навпаки;

- сутність – це клас, який описує якийсь об’єкт, наприклад, користувача.

При описі компонентів на діаграмі також вказують їх параметри, методи та їхні зв’язки з іншими елементами.

Для кращого представлення та встановлення чітких обмежень до класів на нижньому рівні було побудовано діаграму, що зображена на рисунку 3.4.

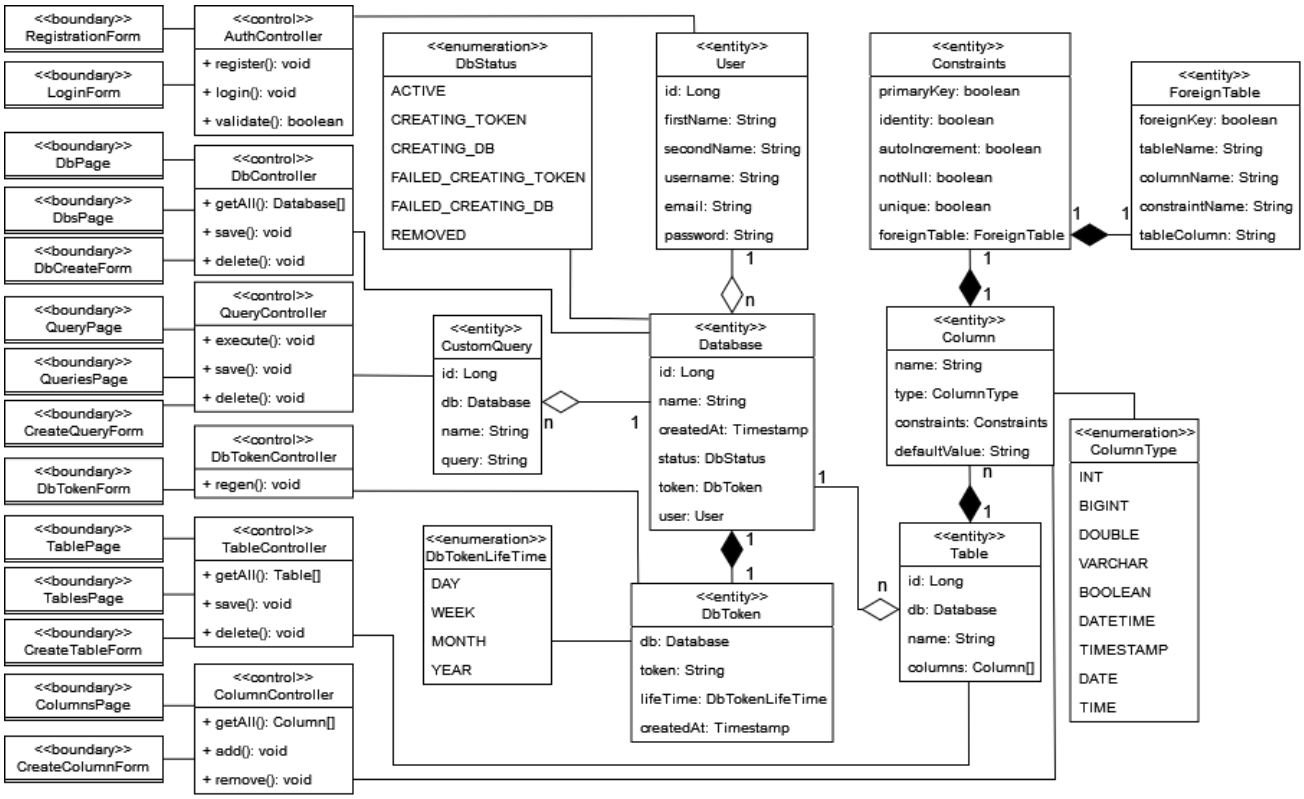

Рисунок 3.4 – Діаграма класів інструментів розробника реляційної бази даних

З діаграми 3.4 видно ключові форми, які використовуються для введення та виводу даних, контролери, з’єднують сутності та сторінки. Ключовою сутністю

системи є “База даних”, навколо неї будується вся логіка, а також від неї залежать усі інші об’єкти.

Система має 8 сутностей, 3 переліки, 6 контролерів, 14 меж. Розглянемо властивості кожного з класів. Розпочнемо з сутностей.

Користувач (User):

- id – числовий ідентифікатор, генерується автоматично;

- firstName – символьне значення, зберігає ім’я;

- secondName – символьне значення, використовується для збереження прізвища;

- username – символьне значення, що зберігає логін;

- email – символьне значення, зберігає пошту;

- password – символьне значення, зберігає пароль у зашифрованому вигляді.

База даних (Database):

- id – числовий ідентифікатор, який генерується;

- name – символьне значення, що представляє назву таблиці;

- createdAt – поле формату мітка часу (Timestamp), використовується для збереження дати та часу створення бази даних;

- status – може приймати значення з переліку DbStatus, використовується для відстежування стану бази даних у конкретний момент часу;

- token – є об’єктом класу DbToken, що представляє собою токен доступу до бази даних;

- user – є об’єктом класу User, зберігає в собі власника бази даних.

Токен доступу до бази даних (DbToken):

- db – об’єкт класу Database, використовується замість ідентифікатора, оскільки ним виступає поле id цього об’єкта;

- token – символьне значення, що зберігає код доступу до бази даних у вигляді JWT;

- lifetime – приймає значення з переліку DbTokenLifeTime, використовується для встановлення часу життя даного токену;

- createdAt – поле форму мітка часу, містить дату та час створення токена.

Таблиця (Table):

- id – числовий ідентифікатор, створюється автоматично;

- db – є значенням класу Database, використовується для зв’язування таблиці та бази даних;

- name – символьне значення, що зберігає назву таблиці, що була вказана користувачем;

- columns – список об’єктів типу Column, зберігає в собі інформацію про всі стовпці таблиці.

Стовпчик (Column):

- name – символьне значення, яке зберігає назву стовпця;

- type – є значенням з переліку ColumnType, що вказує на тип значень, які може приймати стовпець;

- constraints – є значенням класу Constraints, що вказує на обмеження, що були вказані для стовпця;

- defaultValue – символьне значення, що зберігає значення за замовчуванням для стовпця.

Обмеження (Constraints):

- primaryKey – булеве значення, вказує на те, чи є стовпець головним ключем;

- identity – булеве значення, вказує на те, чи генерується значення стовпчика автоматично базою даних;

- autoIncrement – булеве значення, вказує на те, чи даний стовпчик є числовим та його значення автоматично збільшується базою даних;

- notNull – булеве значення, вказує на те, чи може даний стовпець мати порожнє значення;

- unique – булеве значення, вказує на те, чи повинен стовпець мати унікальні значення;

- foreigntTable – значення класу ForeignTable, містить інформацію про зовнішню таблицю.

Зовнішня таблиця (ForeignTable):

- foreignKey – булеве значення, вказує на те, чи є цей стовпець зовнішнім ключем на іншу таблицю;

- tableName – символьне значення, зберігає назву зовнішньої таблиці;

- columnName – символьне значення, зберігає назву стовпця з зовнішньої таблиці;

- constraintName – символьне значення, зберігає назву індекса для зовнішнього ключа;

- tableColumn – символьне значення, зберігає назву стовпця в таблиці на який встановлене зовнішнє з'єднання.

Користувацький запит (CustomQuery):

- id – ідентифікатор, містить автоматично згенероване числове значення базою даних;

- db – значення класу Database, містить інформацію про базу даних для якого запит був створений;

- name – символьне значення, назва запиту введена користувачем;

- query – символьне значення, SQL запит введений користувачем.

До переліків відносяться 3 наступні класи: DbStatus, ColumnType, DbTokenLifeTime.

Статус бази даних (DbStatus) – містить 6 станів, які може приймати база даних під час свого циклу життя.

Тип стовпчика (ColumnType) – містить можливі типи даних, які підтримуються для таблиць.

Час життя токена доступу до бази даних (DbTokenLifeTime) – містить 4 часових проміжки для часу життя токена.

Контролери здебільшого мають однакові методи, що дозволяють виконувати операції читання, зберігання, оновлення та видалення із сутностями з якими вони працюють. Наприклад, контролер для баз даних (DbContoller) має три методи: getAll – отримує всі бази даних для користувача; save – створює нову базу даних; delete – видаляє базу даних.

Схожі методи мають й інші контролери, окрім контролера авторизації (AuthController). Він містить 3 методи для авторизації та реєстрації користувачів, а також перевірки валідності авторизаційного токена. Також контролер для токенів доступу до бази даних (DbTokenController) містить лише 1 метод, який дозволяє створювати новий код.

Межами системи є сторінки (відображають користувачам інформацію, яку вони запитали) та форми (дозволяють користувачам вводити дані для створення/оновлення/видалення сутностей).

3.3 Діаграма взаємодії користувача з системою інструментів розробника реляційної бази даних

Визначення головних можливостей та користувачів системи є обов'язковим етапом перед реалізацією будь-якого програмного продукту. Для вирішення даної задачі прийнято використовувати діаграму прецедентів. За рахунок неї можна визначити акторів, тобто користувачів, які зможуть взаємодіяти з системою, а також прецеденти, іншими слова підтримувані дії, які кожен із визначених акторів зможе виконувати в програмному застосунку [21].

У системі передбачені два актори: гість та зареєстрований користувач. Перший може лише реєструватися, після чого він стає зареєстрованим користувачем, і в результаті чого отримує повний доступ до можливостей програмного продукту, наприклад, може створювати базу даних, таблиці та отримує доступ до використання API для взаємодії з даними в таблицях.

Тож на етапі планування розробки інструментів розробника реляційної бази даних було побудовано діаграму прецедентів, що дозволила відобразити ключових акторів системи, а також їх можливі варіанти взаємодії з нею. Результат зображений на рисунку 3.5.

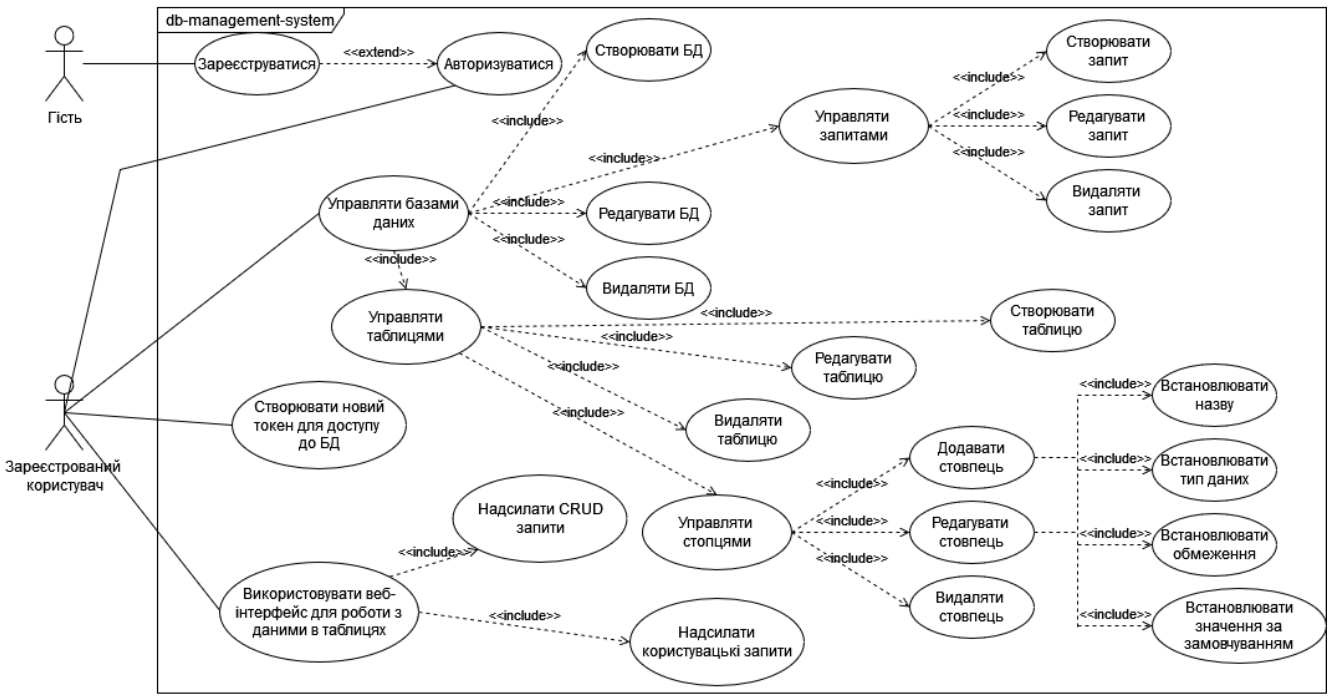

Рисунок 3.5 – Діаграма прецедентів інструментів розробника реляційної бази даних

З діаграми 3.5 також видно порядок дій, який має виконати користувач для досягнення певної цілі. Наприклад, сценарій створення нової таблиці в базі даних можна описати наступним чином:

1. Користувач авторизується в системі.
2. Переходить у меню керування базами даних та створює нову.
3. Відкриває створену базу даних та додає нову таблицю.

Забезпечення обмежень повноважень для кожного типу користувача в системі відбувається за допомогою можливостей модулю Spring Security: дозволяє встановити права доступу та необхідні ролі до кожної кінцевої точки.

3.4 Структура баз даних InterSystems IRIS та Redis інструментів розробника реляційної бази даних

На етапі проектування системи було встановлено дані з якими мають взаємодіяти розроблювані інструменти розробника реляційної бази даних. До них можна віднести інформацію про: користувачів, бази даних, таблиці, авторизаційні

токени користувачів, токени доступу до баз даних, користувацькі запити до таблиць. Представлені сутності мають спільні та відмінні риси за якими їх було розділено на дві групи. До першої групи віднесено дані, що зберігаються в системі постійно, мають великі обсяги та не потребують досить швидкої взаємодії з ними. До другої ж було додано дані, що потребують частого швидкого запису та читання з бази даних, та є тимчасовими.

Щоб забезпечити дані вимоги до збереження інформацію було прийнято рішення використовувати два різних сховища даних, які справлятимуться з поставленими перед ними задачами. У результаті було використано систему управління даними InterSystems IRIS (для постійних даних) та Redis (для тимчасових даних). На рисунку 3.6 зображена діаграма, на якій зображено сутності та базу даних до якої вони відносяться.

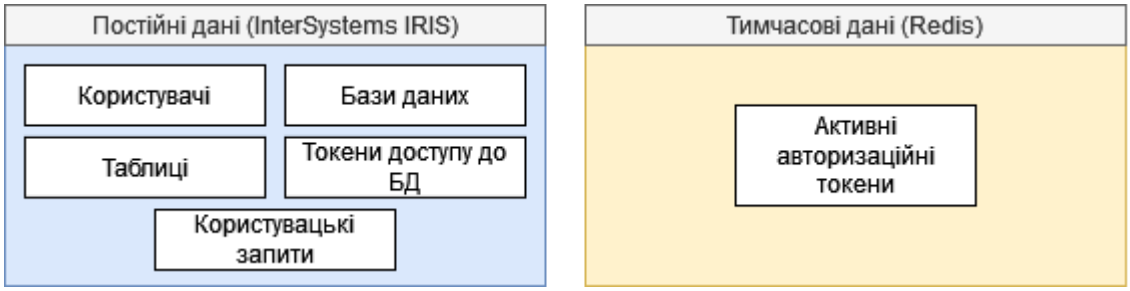

Рисунок 3.6 – Розподіл даних за властивостями між СУБД InterSystems та Redis

Наступним етапом було визначення властивостей та зв’язків між сутностями, що зберігаються в системі. Дану проблему було вирішено за допомогою побудови логічної моделі даних.

На діаграмі (рисунок 3.7) було зображено 6 сутностей, для яких було визначено властивості, які вони мають зберігати в собі. Між сутностями було визначено зовнішні зв’язки один до одного (1 – 1) та один до багатьох (1 – n), щоб забезпечити цільність даних у системі.

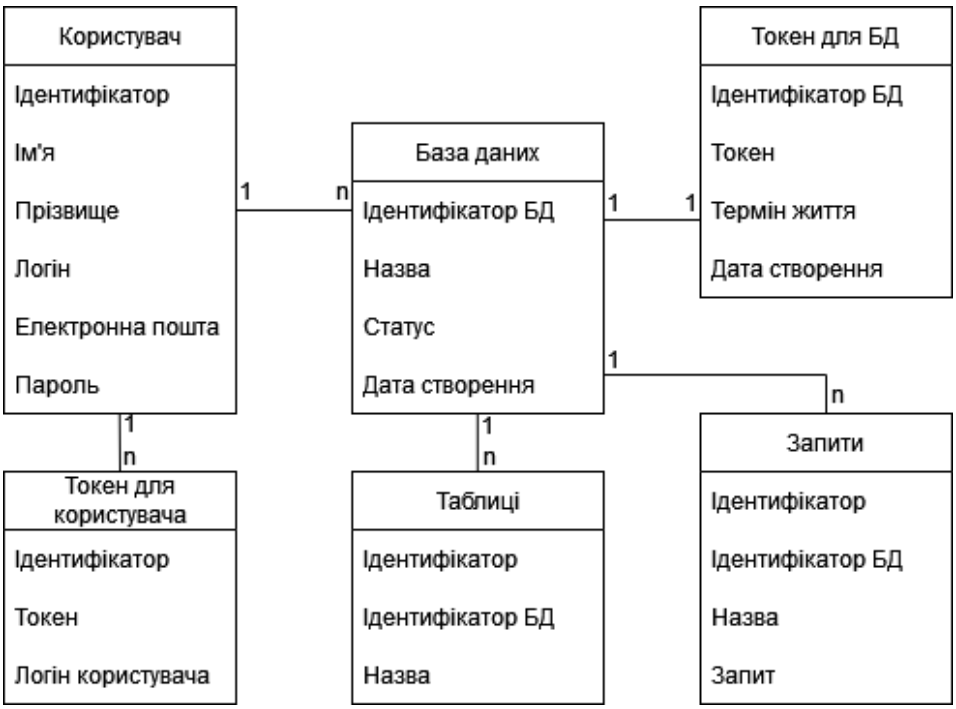

Рисунок 3.7 – Логічна модель даних предметної області

Окрім логічної моделі було побудовано структури кожної з використовуваних баз даних [22]. Спочатку розглянемо архітектуру для СУБД Redis, бо вона має лише одну сутність, яка зберігає інформацію про актуальні авторизаційні токени користувачів (JWTs). Структурою збереження даних було обрано хеш (hash), який складається з наступних властивостей:

- id – тип даних String, використовується як ключ хешу;

- token – тип даних String, містить унікальний згенерований JWT;

- username – тип даних String, використовується для зв’язування даних між таблицею користувачів та цим хешем.

Описана архітектура зображена на рисунку 3.8.

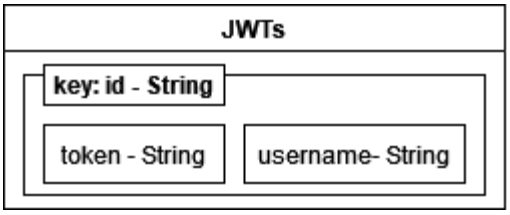

Рисунок 3.8 – Архітектура бази даних Redis

На відмінну від Redis основна база даних має більш об’ємну архітектуру. Вона складається з п’яти сутностей, що представлені таблицями, та дозволяє ефективно зберігати дані за рахунок нормалізованої схеми даних. На рисунку 3.9 зображена діаграма з архітектурою бази даних InterSystems IRIS.

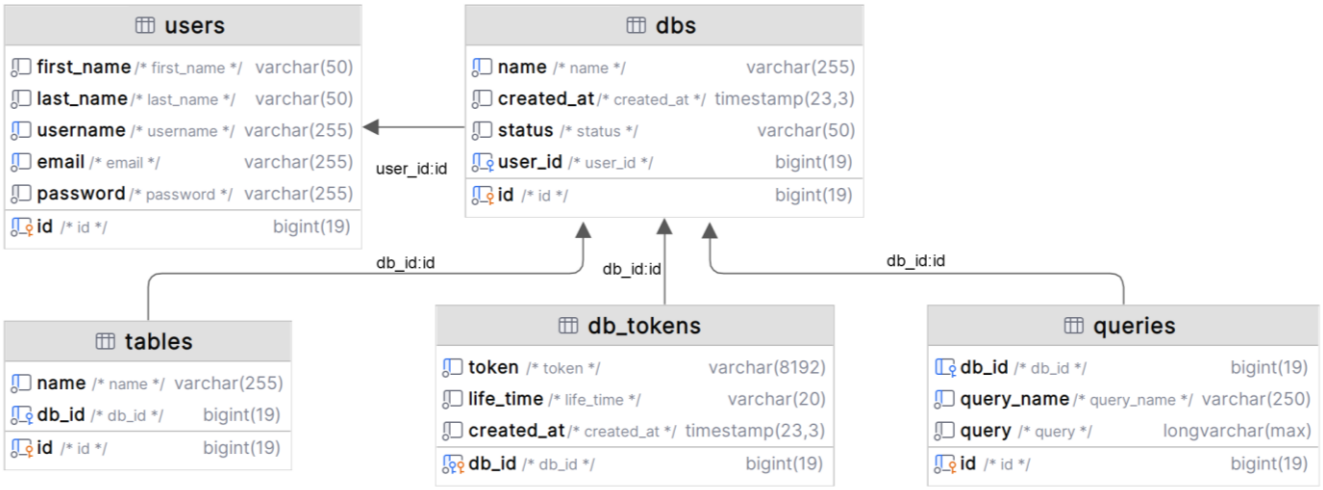

Рисунок 3.9 – Архітектура бази даних InterSystems IRIS

Під час побудови даної архітектури було встановлено набір загальних правил, які використовувалися під час написання SQL запитів для створення таблиць. За основу було взяті рекомендації та практики від провідних технічних компаній, які є широко використовувані по всьому світу. У результаті було визначено наступні вимоги до створення обмежень для стовпців.

Найменування для індексу головного ключа створюється за наступним шаблоном: pk_{назва таблиці}. Назва таблиці має відповідати тій, для якої це обмеження створюється.

Найменування індексу зовнішнього ключа утворюється за таким шаблоном: fk_{назва таблиці з індексом}_{назва зв’язуваної таблиці}. Тобто першою встановлюємо назву таблиці ту, в якій дане обмеження створюється, а другою – ту, яка містить значення та зв’язується з першою.

Для створення індексу перевірки значення на унікальність встановлено такий шаблон: uq_{назва таблиці}_{назва поля}. Назву поля беремо ту, на яке дане обмеження встановлюється.

Для прикладу того, як створювалися таблиці в базі даних InterSystems IRIS, наведемо приклад створення таблиці «Запити», оскільки вона включає в себе всі представлені вище обмеження та може слугувати найкращим прикладом у даному випадку. Складається вона з наступних властивостей та обмежень:

- id: тип даних BIGINT; використовується як головний ключ; позначений як ідентифікаційний стовпець, тому значення генеруються базою даних автоматично; назва індекса – pk_queries;

- db_id: тип даних BIGINT; є зовнішнім ключем на таблицю «Бази даних»; назва індекса – fk_queries_dbs;

- query_name: тип даних VARCHAR(250), що зберігає символьні значення з максимально довжиною 250;

- query: тип даних TEXT, не має обмежень по довжині;

- властивості db_id та query_name використовуються для створення індекса, який відповідає за перевірку унікальності введених даних, щоб не можна було вводити однакову назву запиту для одної і тої ж самої таблиці; дане обмеження має наступну назву – uq_queries_db_id_query_name.

За даним принципом були побудовані всі таблиці з цього сховища даних.

3.5 Механізм забезпечення безпеки в системі інструментів розробника реляційної бази даних

Захист системи реалізований з використанням одного модуля з фреймворку Spring, який має назву Spring Security [23]. Даний частина фреймворку реалізовує весь необхідний функціонал: управління сесіями користувачів, що успішно пройшли авторизацію; шифрування та перевірка валідності введених паролів; зручну конфігурацію ввімкнення обмеження доступу до кінцевих точок програмного продукту; перевірку необхідних ролей для доступу; підтримку JWT.

Тож захист кінцевих точок був реалізований з використанням даного модулю та JSON Web Token. Під час кожної успішної автентифікації в системі Spring Security створює JWT, який віддає клієнту для подальшої взаємодії з системою в

повній мірі. На рисунку 3.10 зображено діаграму послідовності, яка в спрощеному вигляді відображає суть реалізованого механізму.

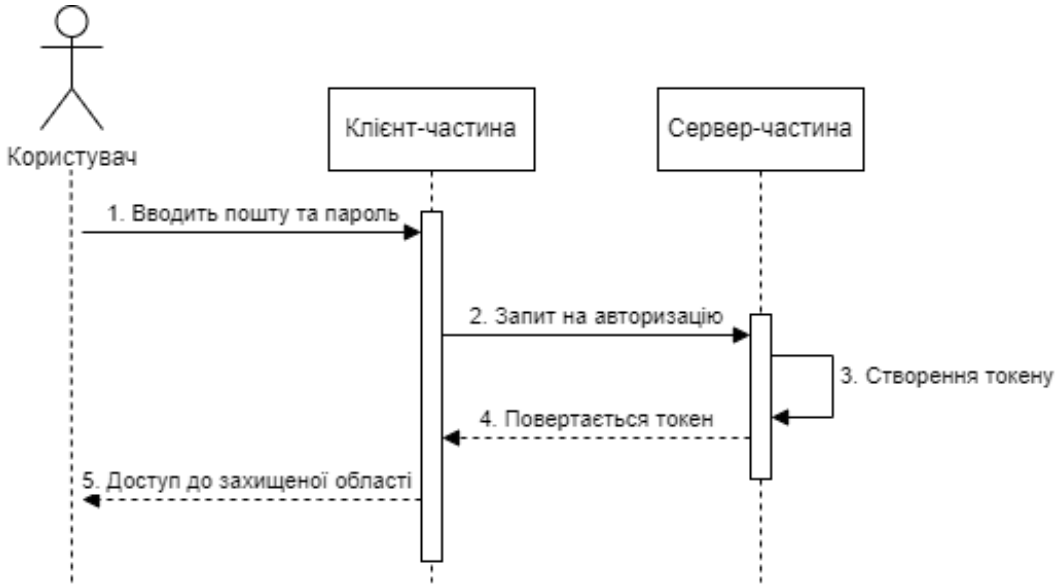

Рисунок 3.10 – Механізм авторизації користувача за допомогою JWT

З рисунка 3.10 видно, що процес автентифікації ініціюється користувачем, який вводить свої авторизаційні дані. Після чого клієнтська частина надсилає введені дані на сервер, який здійснює перевірку валідності отриманої інформації. Якщо перевірка пройшла успішно, то користувач отримає новий токен, використовуючи його він зможе користуватися програмним продуктом без будь-яких обмежень.

Окрім цього було розроблено спеціальну кінцеву точку – /auth/validate, яка дозволяє перевіряти правильність та актуальність користувацького токену, щоб надавати можливість працювати з інструментами розробника реляційної бази даних. Даний процес зображено за допомогою діаграми послідовності на рисунку 3.11.

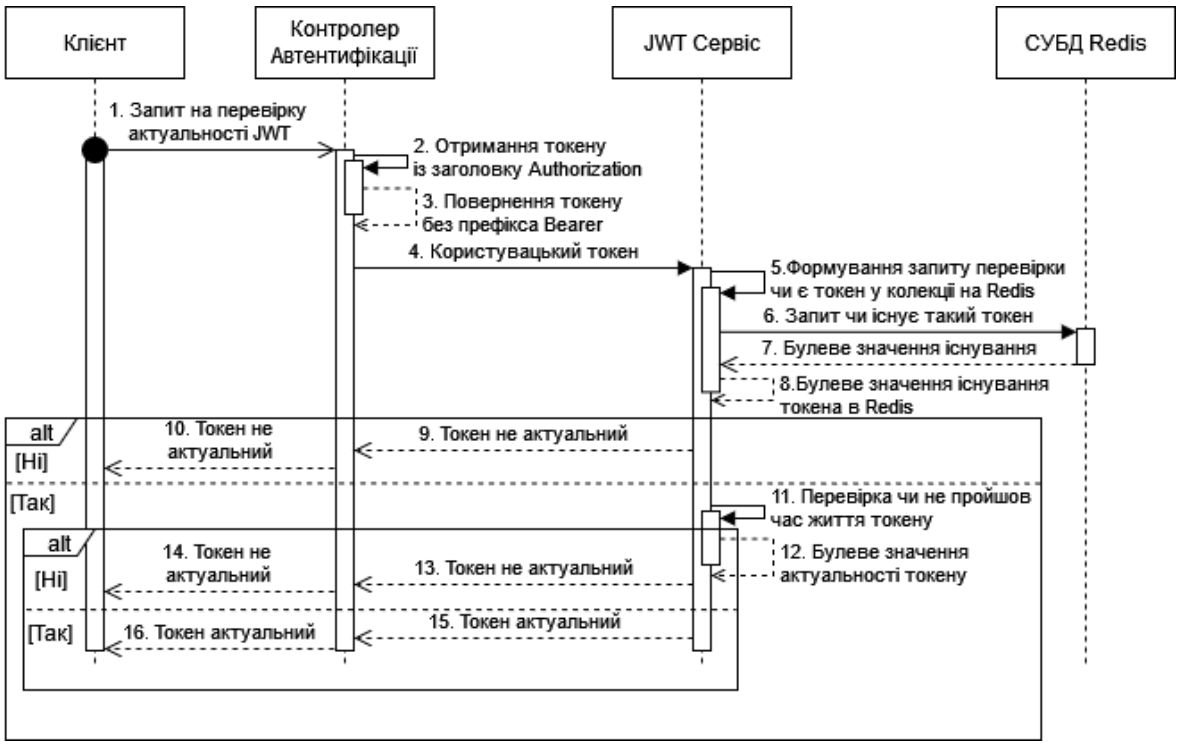

Рисунок 3.11 – Діаграма послідовності перевірки JWT на актуальність

Послідовність кроків на рисунку 3.11:

1. Клієнт надсилає запит з JWT за допомогою HTTP-заголовку “Authorization” на кінцеву точку /validate.

2. Сервер отримує запит та дістає токен із заголовку, прибираючи префікс Bearer.

3. Повертається чистий JWT.

4. Отриманий токен передається у сервіс, що займається їх генеруванням та перевіркою.

5. Створюється запит для перевірки того, чи збережений токен у колекції Redis.

6. Запускається запит до БД.

7. Отримується булева відповідь існування токену в Redis.

8. Повертається булеве значення існування в поточний потік.

9. Токен не присутній у колекції, тому його можна вважати неактивним, у цьому випадку повертаємо з сервісу негативну відповідь.

10. Відправляємо клієнту негативну відповідь клієнту.

11. У випадку, коли токен присутній, то виконується перевірка на те, чи не пройшов його час життя.

12. Отримується булева відповідь чи не “живий” ще токен.

13. Якщо токен не актуальний, то повертаємо негативну відповідь із JWT сервісу.

14. Відправляємо на клієнт негативну відповідь.

15. Якщо токен актуальний, то відповідь сервісу позитивна.

16. Повертаємо позитивну відповідь клієнту.

3.6 Процес створення сутності “База даних” в інструментах розробника реляційної бази даних

Одною з основних та головних можливостей системи є створення нової бази даних для користувача. Цей етап включає в себе виконання багатьох підпроцесів, які відіграють ключову роль у подальшій правильній роботі системи. До таких можна віднести створення:

- нового запису про об’єкт “База даних” у базі даних;
- нової бази даних на кластері;
- нового користувача для цієї бази даних;
- токену доступу до цієї бази даних за допомогою REST API.

Перші три підпроцеси виконуються за допомогою можливостей реляційної бази даних – транзакції. Оскільки вони мають виконатися всі, або ж не виконатися взагалі.

Підпроцес створення токену доступу до БД не є сильно важливим, оскільки користувач має змогу згенерувати його повторно у разі невдалого першого його виконання. Тому його можна виконувати асинхронно, не чекаючи на його завершення, що б не затримувати користувача.

Для детального та наочного представлення цього механізму було створено діаграму послідовності, що зображена на рисунку 3.12.

Рисунок 3.12 – Діаграма послідовності створення нової бази даних

З схеми на рисунку 3.12 видно всі підпроцеси, які відбуваються під час створення нової бази даних. Як і було сказано вище: створення токену доступу до БД відбувається асинхронно, щоб пришвидшити очікування клієнта.

Представлений порядок дій на діаграмах можна описати словами так:

1. Користувач відправляє форму створення БД.
2. Клієнтська частина проводить первинну перевірку даних.
   2.1. Якщо дані введені неправильно, то відразу повертається помилка та завершується процес.
   2.2. Якщо дані введено правильно, то вони надсилаються на сервер.
3. Дані проходять перевірку на сервері.
   3.1. Якщо перевірка не пройдена, то повертається помилка та процес завершується.
   3.2. Якщо перевірку пройдено, то продовжується процес створення БД.
4. Інформацію про сутність «База даних» зберігається в системі.

5. Створюється нова відповідна база даних на кластері.

6. Створюється користувач, який має до неї доступ.

7. Запускається асинхронний процес створення токену доступу до бази даних.

8. Відправляється повідомлення про успішне збереження на клієнтську частину.

9. Повідомлення про успішне збереження показується користувачів.

Для цього механізму створення нової баз даних було створено блок-схему алгоритму, яка зображена на рисунку 3.13.

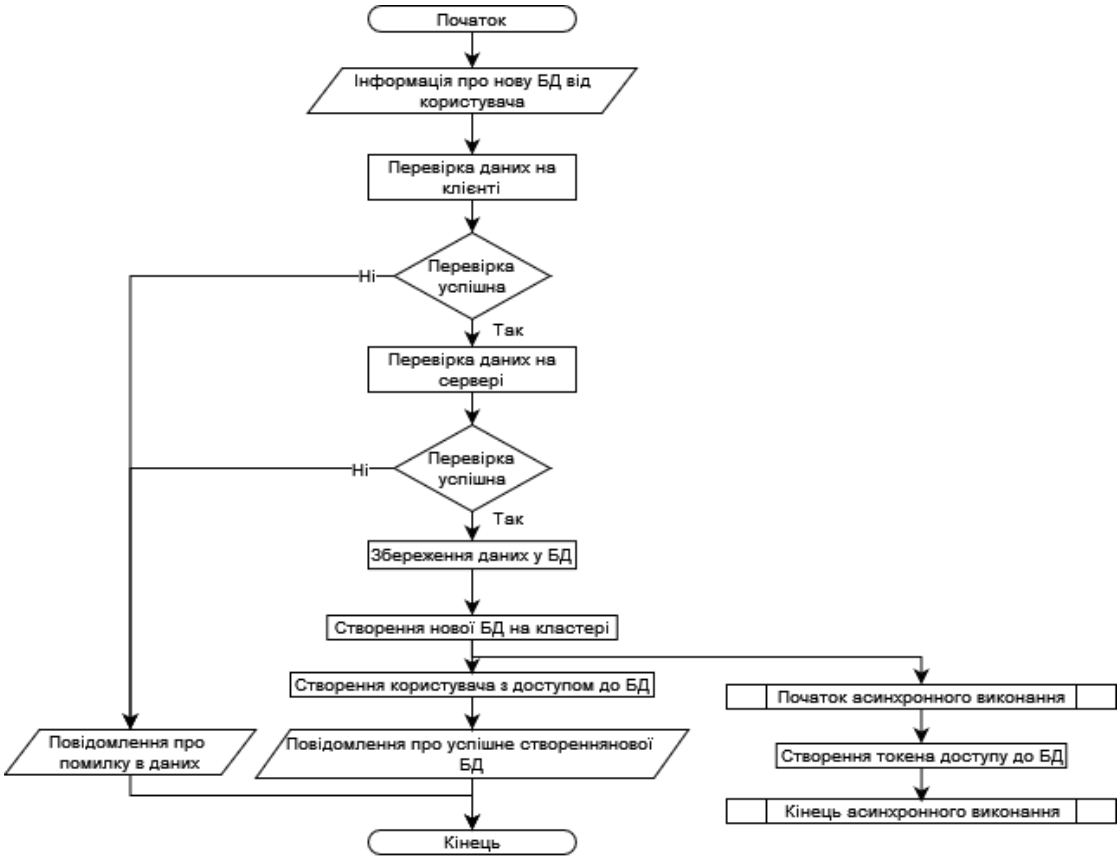

Рисунок 3.13 – Блок-схема алгоритму створення нової БД

Під час виконання підпроцесів об’єкт “База даних” змінює свої стани, щоб дозволити ефективно відслідковувати успішність її створення, а в разі помилки дозволити користувачу повторити процеси, які не вдалося виконати. Також зміна станів дозволяє зрозуміти на якому саме етапі виникла проблема, щоб у майбутньому виправити дану проблему.

Усього для відслідковування життєвого циклу об’єкту “База даних” було створено 6 статусів:

- СТВОРЕННЯ БД – створюється запис про об’єкт “База даних” у таблиці;

- ПОМИЛКА СТВОРЕННЯ БД – на етапі створення нової бази даних виникла помилка, яку неможливо виправити, та процес не може бути продовженим далі – необхідно перезапустити його;

- СТВОРЕННЯ ТОКЕНУ – свідчить про те, що інформація про об’єкт успішно була збережена, нова база даних створена успішно та було запущено асинхронний процес створення токену доступу;

- ПОМИЛКА СТВОРЕННЯ ТОКЕНУ – токен не може бути створений, потрібно повторити генерацію;

- СТВОРЕНА – усі кроки по створенню нової користувацької бази даних було виконано успішно;

- ВИДАЛЕНА – користувач видалив базу даних.

Описані вище статуси зображені на UML діаграмі станів об’єкту “База даних”, яка знаходиться на рисунку 3.14. На ній графічно описано умови переходів з одного в інший стан, а також дії, які виконуються в системі при їх зміні. Так, наприклад, умовою переходу зі статусу “СТВОРЕННЯ БД” в “СТВОРЕННЯ ТОКЕНУ” є успішне створення запису в таблицю про нову користувацьку базу даних, а результатом зміни стану є запуск асинхронного створення токену доступу. Також з діаграми видно, що в разі виникнення будь-яких помилок на етапах створення бази даних або токена доступу до неї – користувач самостійно може повторно зініціювати процес, який не вдалося виконати. Наприклад, якщо було встановлено значення “ПОМИЛКА СТВОРЕННЯ ТОКЕНУ”, то існує можливість запустити даний процес повторно, та перевести базу даних в стан “СТВОРЕННЯ ТОКЕНУ”. На будь-якому кроці створення чи в результаті отримання статусу помилки – передбачена можливість видалити базу даних, у цьому випадку система встановить статус “ВИДАЛЕНА”, що унеможливить подальші взаємодії з даним об’єктом.

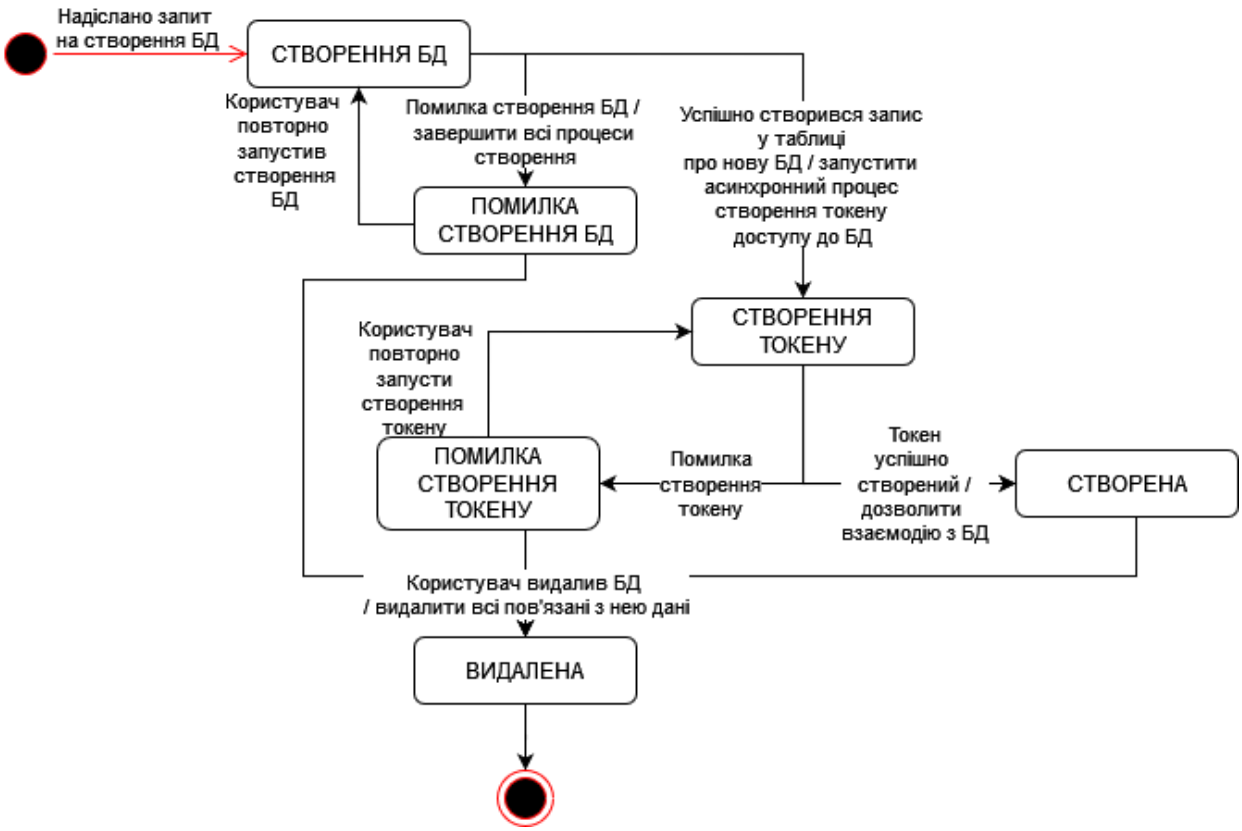

Рисунок 3.14 – UML діаграма станів об’єкту “База даних”

Розроблений алгоритм створення нової бази даних дозволяє ефективно виконувати поставлені задачі по збереженню інформації в таблицю, створенню нової БД на кластері, надання прав доступу на читання користувачу, створення токену доступу до БД через REST API. Останній підпроцес відбувається в асинхронному потоці, оскільки він не впливає на подальшу взаємодію користувача на інтерфейсі, через це можна не чекати на його завершення. Користувач зможе продовжити створення таблиць для БД, а до того часу токен буде створений, що дозволить його використання. Протягом виконання більшості підпроцесів у описаному механізмі об’єкт “База даних” змінює свої стани на шість можливих варіантів.

3.7 REST API для роботи з даними в таблицях в інструментах розробника реляційної бази даних

Оскільки головною задачею інструментів розробника реляційної бази даних є можливість створення потрібних сховищ, які можна інтегрувати в користувацькі програми. Тому для вирішення даної проблеми було розроблено спеціальний програмний прикладний інтерфейс, за стандартом REST. Він містить п’ять запитів для взаємодії з даними в таблицях. Вони забезпечують базові операції (створення, читання, оновлення та видалення даних) та виконання користувацьких запитів. Для кращого розуміння оглянемо кожен з них.

Для початку розглянемо спільне для всіх запитів. Усі вони захищені та потребують автентифікації для виконання. Щоб забезпечити зручність та надійність було використано JWT, оскільки в нього можна вкласти потрібну для автентифікації інформацію. У випадку захисту даного API для кожної бази даних створюється унікальний токен, в який вкладається її ідентифікатор, після чого кодується з використанням HS256 алгоритму та використанням секретного коду. Приклад структури токену зображена на рисунку 3.15.

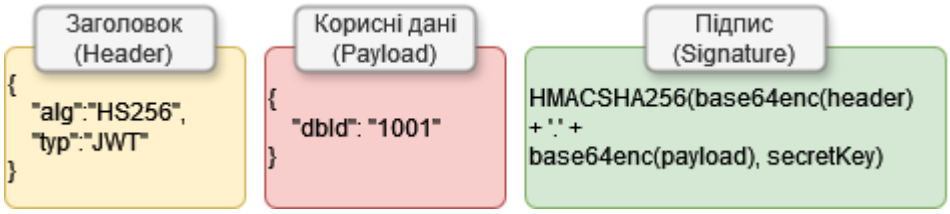

Рисунок 3.15 – Структура токену доступу до БД

Через це будь-який запит до REST API має містити HTTP заголовок з назвою “Authorization”. Значення має створюватися за патерном, що зображений на рисунку 3.16, для цього необхідно використовувати префікс “Bearer” та автоматично згенерований токен для конкретної бази даних.

```
'Authorization': Bearer {{токен доступу до БД}}
```

Рисунок 3.16 – Шаблон значення для заголовку “Authorization”

Слід зауважити, що час життя кожного такого токену є обмеженим, щоб забезпечити надійний захист користувацьких даних у таблицях. Тож у разі виходу терміну дії коду, його потрібно буде згенерувати повторно. При генерації токенів доступні чотири різні тривалості: день, тиждень, місяць, рік. Це дає змогу користувачу самостійно обрати найкращий варіант для себе.

Ще одною спільною рисою для всіх кінцевих точок API є те, що запит до них повинен надсилатися з тілом, що має JSON формат та використовувану правильну структуру для конкретного запиту, що описані нижче. Відповідь від сервера також приходить у форматі JSON, і також має чітко визначену структуру для кожного випадку.

Розпочнемо огляд запитів та відповідей кінцевих точок із запиту створення даних у таблиці. Запит має виконувати з використанням HTTP методу POST на URI /queries/{tableName}, де tableName – це назва таблиці в базі даних. Загальний вигляд тіла запит та відповіді представлено на рисунку 3.17.

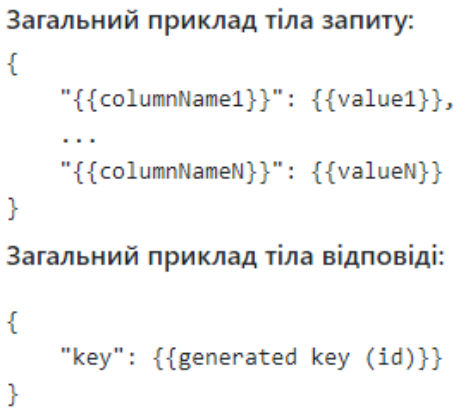

Рисунок 3.17 – Загальний вигляд тіла запиту та відповіді для створення запису в таблиці

Із рисунку 3.17 видно, що тіло запиту складається з переліку назв стовпців таблиці та їх значень. Даний запит підтримує збереження лише одного запису за один раз. У результаті сервер повертає автоматично згенерований унікальний ідентифікатор для створеного рядка в значенні параметру “key”.

Наступним розглянемо запити двох кінцевих точок для читання записів із таблиць:

- з використанням фільтру по одному стовпчику, використовуючи метод POST та URI: /queries/{tableName}/getByColumn;

- без фільтрів, використовуючи GET метод та URI: /queries/{tableName}.

На рисунку 3.18 зображений приклад тіла запиту та відповіді для читання записів із використанням умови.

Загальний приклад тіла запиту:

```
{
    "columnName": {{columnName}},
    "value": {{value}},
    "operator": {{Одне із значень: EQUALS, NOT_EQUALS, GREATER, LOWER}}
}
```

Загальний приклад тіла відповіді:

```
{
    "objects": [{
        "{{columnName1}}": "{{value1}}",
        ...
        "{{columnNameN}}": "{{valueN}}"
    }]
}
```

Рисунок 3.18 – Загальний вигляд тіла запиту та відповіді для отримання всіх записів за умовою

Як показано на рисунку 3.18, тіло запиту має містити назву стовпця, його значення та один із перелічених операторів, щоб програма могла створити умову фільтрації в SQL запиті. У результаті виконання має повернутися масив об’єктів, які були вичитані з таблиці.

Друга ж кінцева точка для читання дозволяє отримати всі записи без фільтрації. Вона не потребує передавання умови в тілі запиту. А її відповідь також складається зі списку об’єктів з таблиці, як і в першої.

Оскільки використання цих двох кінцевих точок для читання інформації з бази даних дещо обмежують користувача, бо не підтримують складних фільтрів з декількома умовами в SQL запиті, також не дозволяють обирати окремі поля та інше. Через це було реалізовано можливість створення користувацьких запитів для читання даних, в які можна передавати іменовані параметри. Приклад запиту з використанням двох таких параметрів (minPrice та maxPrice) знаходиться на рисунку 3.19.

```
SELECT * FROM items i WHERE i.price > :minPrice AND i.price < :maxPrice;
```

Рисунок 3.19 – Приклад користувацького запиту з параметрами

Для виконання цього запиту в тілі потрібно передавати іменовані параметри та відповідно значення для них. Шаблон URI є /queries/custom/{queryName}, де queryName – назва SQL запиту, що вказав користувач. Приклад виконання цього запиту та структура відповіді зображена на рисунку 3.20.

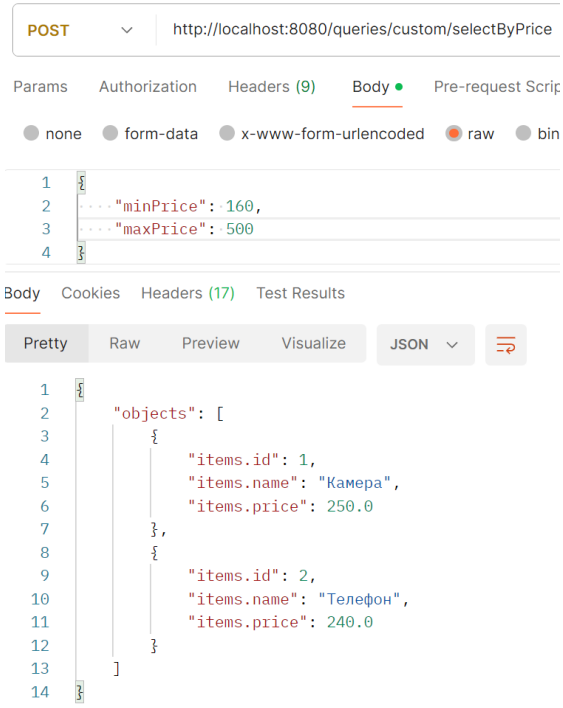

Рисунок 3.20 – Приклад тіла запиту та відповіді для користувацьких запитів

З рисунку 3.20 видно, що відповідь є масивом об’єктів з переліком їх параметрів, що відповідає попереднім двом попереднім універсальним запитам для читання.

Наступною є кінцева точка для оновлення даних. Знаходиться за URI /queries/{tableName}, для виконання потрібно використовувати запит PUT. Загальна структура тіла запиту та відповіді знаходиться на рисунку 3.21.

Загальний приклад тіла запиту:

```
{
    "condition": {
        "columnName": {{columnName}},
        "value": {{value}},
        "operator": {{Одне із значень: EQUALS, NOT_EQUALS, GREATER, LOWER}}
    },
    "object": {
        "{{columnName1}}": {{value1}},
        ...
        "{{columnNameN}}": {{valueN}}
    }
}
```

Загальний приклад тіла відповіді:

```
{
    "message": "Updated"
}
```

Рисунок 3.21 – Загальний приклад структури тіла запиту та відповіді для оновлення запису

Із зображення 3.21 видно, що тіло запиту поєднує в собі два об’єкти: перший – умова по якому стовпцю потрібно проводити оновлення; другий – нове значення параметрів об’єкта, не обов’язково мають бути вказані всі існуючі параметри. Відповідь у свою чергу завжди складається з об’єкту, який має параметр “message” та значення “Updated”.

Останньою кінцевою точкою є видалення запису за умовою. Для цього потрібно надсилати запит з використанням DELETE методу на URI /queries/{tableName}. Приклад структури тіла запиту та відповіді зображено на рисунку 3.22.

Загальний приклад тіла запиту:

```
{
    "columnName": {{columnName}},
    "value": {{value}},
    "operator": {{Одне із значень: EQUALS, NOT EQUALS, GREATER, LOWER}}
}
```

Загальний приклад тіла відповіді:

```
{
    "message": "Removed"
}
```

Рисунок 3.22 – Структура тіла запиту та відповіді для кінцевої точки видалення запису

Як показано на рисунку 3.22: тіло запиту має умову по якій виконувати видалення запису з таблиця, а відповідь завжди складається з об’єкту, який має параметр “message” та значення “Removed”.

Отже, для створення архітектури високого рівня абстракції було використано клієнт-серверний підхід: дозволило розділити логіку між двома великими частинами. Клієнт займається надсиланням клієнтських запитів та обробкою відповідей на них; сервер опрацьовує запити, запускаючи відповідні процеси, та відповідає на них. Реалізація клієнтської частини відбувалася за допомогою односторінкового застосунку та використанням компонентно-орієнтованої архітектури низького. Для побудови серверної частини було використано багатошарову архітектуру.

За допомогою діаграми класів було визначено основні сутності з якими взаємодіє система як на клієнті, так і на сервері. Для сутностей було встановлено контролери, які займаються їх обробкою, та представлення, з яких надходить інформація від користувача та які надсилаються результати обробки даних. Це дозволило чітко визначити взаємозв’язки елементів у системі та швидко реалізувати поставлені задачі.

Перед розробкою інструментів розробника реляційної бази було визначено наперед можливих користувачів: гість та зареєстрований користувач. Гість не має доступу до функціоналу, окрім як реєстрації Зареєструвавшись він стає повноцінним користувачем системи.

Оскільки дані в системі були розділені на постійні та кеш, а для їх збереження вирішено було використовувати InterSystems IRIS та Redis відповідно, то для ефективного виконання цього процесу було побудовано логічну модель в якій визначено ключові параметри кожної сутності. Також було створено архітектури для кожної СУБД. Redis містить одну хеш колекцію, що забезпечую швидку обробку CRUD операцій. InterSystems IRIS у свою чергу зберігає інформацію в п’яти таблицях, які були побудовані за сучасними стандартами та нормалізацією даних, забезпечивши цим надійність збереження даних та їх взаємозв’язки.

Для забезпечення процесу автентифікації було використано відкритий Інтернет стандарт для токенів – JSON Web Token. Використовуючи його було реалізовано механізм захисту всіх кінцевих точок веб-сервера. Для опису процесу було побудовано діаграму послідовності. Також було створено спеціальну кінцеву точку для перевірки правильності токена, щоб надати доступ користувачу до веб-сайту, даний процес також було описано на діаграмі послідовності.

Процес створення нової користувацької бази даних містить багато підпроцесів. Виконання кожного є важливим, оскільки тільки статус про успішність виконання етапу дозволяє перейти на інший. Для чіткого представлення даного механізму було побудовано діаграму послідовності, яка дозволила визначити поряд виконання підпроцесів. Також було створено блок схему для опису алгоритму, який необхідно реалізувати. А оскільки під час створення відбувається багато етапів, то за кожним потрібно слідкувати. Для цього було використано статуси для бази даних, отримання яких було описано за допомогою діаграми станів.

Оскільки головною задачею інструментів розробника реляційної бази даних є забезпечення користувачів веб-інтерфейсом, який можна інтегрувати в їх систему для управління даними в таблицях. Для цього було створено універсальне REST API. Доступ до нього доступний лише за спеціальним кодом доступу, що є JWT, який генерується для кожної бази даних окремо. API має шість кінцевих точок. П’ять забезпечують звичайні CRUD операції. Шоста – можливість виконувати складні користувацькі запити для вибірки даних, які включають розширену фільтрацію та зовнішні зв’язки між таблицями.

4 РОБОТА КОРИСТУВАЧА З ІНСТРУМЕНТАМИ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Розроблені інструменти дозволяють швидко та ефективно створювати бази даних для клієнтських систем без будь-яких знань та досвіду роботи з реляційними базами даних. У даному розділі представлений простий та базовий сценарій створення нової бази даних, таблиці, керування даними через інтерфейс, а також підключення налаштування веб-інтерфейсу для роботи з таблицями в зовнішній системі.

4.1 Запуск системи

Розроблені інструменти розробника реляційної бази даних є веб-системою. Вона розроблена двома мовами програмування: Java та JavaScript. Кожна з яких потребує встановлення додаткових інструментів, бібліотек та виконання налаштувань для правильної роботи. Для спрощення даного процесу було використано платформу Docker, зокрема інструмент Docker Compose.

Даний підхід дозволив описати всі необхідні кроки для запуску системи під час розробки. У результаті для розгортання програмного продукту достатньо виконати одну команду.

Проте перед її виконанням необхідно встановити Docker, завантаживши його з офіційного сайту https://www.docker.com/get-started/, якщо він відсутній. Встановлення виконується досить просто та не має викликати ніяких труднощів. Якщо використовується операційна система Windows, то інсталятор запропонує налаштувати WSL 2.0 [24], щоб запускати віртуальну машину Linux, використовуючи сучасні можливості від Microsoft, тому рекомендується скористатися даною можливістю, оскільки це пришвидшить роботу Docker.

Після встановлення можна переходити до розгортання системи на локальній чи віддаленій машині. Для цього необхідно відкрити командний рядок у кореневій папці проекту. Та виконати команду: docker-compose up. Після чого необхідно

зачекати поки встановляться всі необхідні компоненти та піднімуться всі 4 сервіси. У разі успішного розгортання в терміналі буде виведено те, що зображено на рисунку 4.1.

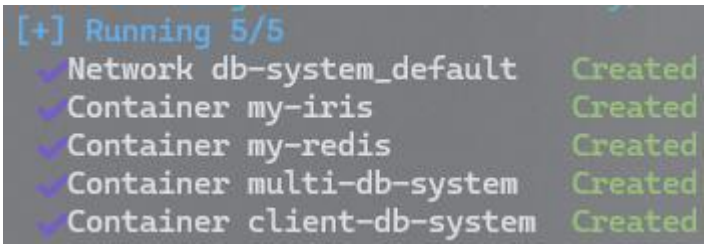

Рисунок 4.1 – Приклад успішного запуску контейнеру

Також можна відкрити програму Docker Desktop [25], відшукати контейнер з назвою db-system та переконатися, що все відповідає рисунку 4.2.

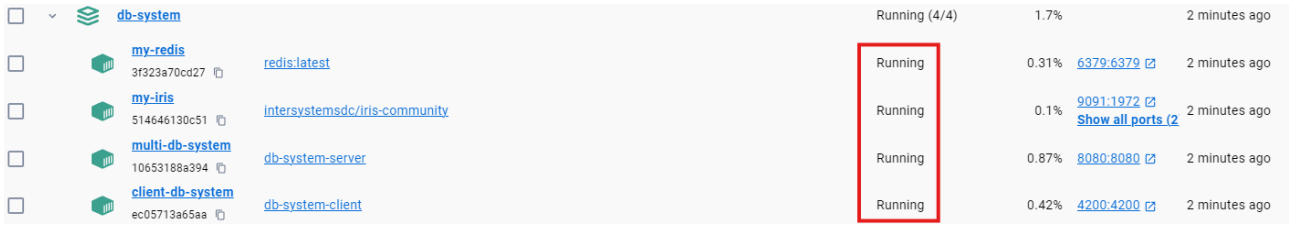

Рисунок 4.2 – Зображення успішного запуску з Docker Desktop

Якщо все вдалося успішно, то можна переходити до взаємодії з користувацьким інтерфейсом та REST API для роботи з даними.

4.2 Взаємодія з інструментами розробника реляційної бази даних через користувацький інтерфейс

Перше налаштування баз даних та подальше керування ними відбувається через користувацький інтерфейс, оскільки лише тут можливо отримати повний доступ. Для роботи необхідно використовувати будь-який пристрій, що підтримує сучасний браузер, щоб виключити всі можливі помилки через застаріле ПЗ.

4.2.1 Реєстрація

Перш за все необхідно створити обліковий запис у системі, щоб отримати повний доступ до функціоналу та мати змогу створити свою першу базу даних. Для цього необхідно перейти за посиланням http://localhost:4200/register. На екрані з’явиться форма, яку необхідно заповнити та натиснути кнопку “Submit”. У разі введення неправильних даних система повідомить про це та не пропустить на наступний крок, тому потрібно бути уважним. На рисунку 4.3 зображено приклад коректного заповнення форми.

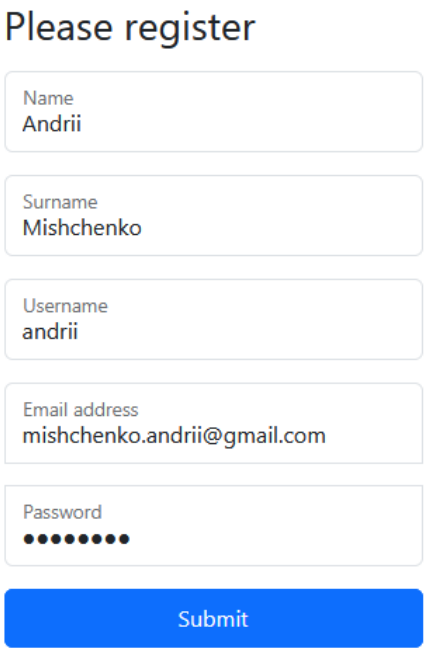

Рисунок 4.3 – Коректне заповнення реєстраційної форми

У разі успішної реєстрації в правому куті вікні браузера з’явиться повідомлення, що зображене на рисунку 4.4.

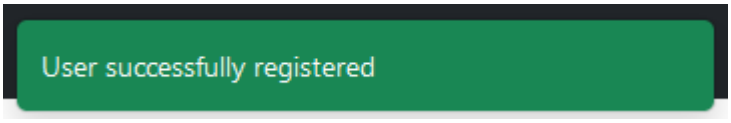

Рисунок 4.4 – Повідомлення про успішну реєстрацію

Якщо створення нового акаунту відбулося успішно, то можна переходити до наступного кроку.

4.2.2 Авторизація

Кожна взаємодія з системою починається з авторизації в ній. Оскільки сесія після автентифікації зберігається на сервері одну добу, то даний процес потрібно буде повторювати з плином цього часу.

Для авторизації в системі необхідно перейти за посиланням http://localhost:4200/login та заповнити форму, вказавши логін та пароль, які були використані під час реєстрації. Після заповнення необхідно натиснути кнопку “Sign In”. Приклад успішно заповненої форми для авторизації зображено на рисунку 4.5.

Рисунок 4.5 – Приклад заповнення форми для авторизації

Якщо авторизація пройшла успішно, то має відбуватися переадресація на домашню сторінку (рисунок 4.6), де в подальшому будуть знаходитися всі створені бази даних.

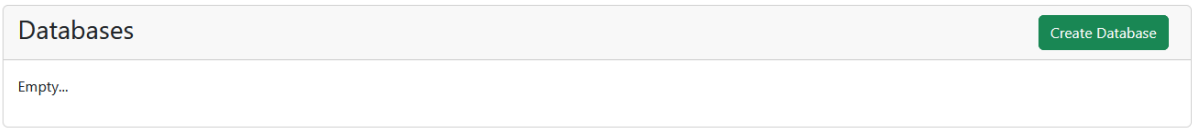

Рисунок 4.6 – Домашня сторінка користувача з відсутніми базами даних

Після успішного виконання авторизації можна переходити до наступного кроку – створення нової бази даних.

4.2.3 Створення першої бази даних

Для створення бази даних необхідно натиснути на кнопку з написом “Create Database”. Після чого з’явиться нова форма, яку необхідно заповнити, вказавши бажану назву та обрати з випадаючого списку тривалість життя токена доступу. Після чого натиснути на підтверджуючу кнопку. Заповнена форма зображена на рисунку 4.7.

Рисунок 4.7 – Форма для створення нової бази даних

У результаті знову відбудеться переадресація на сторінку з переглядом усіх баз даних, проте на цей раз він не буде пустим (рисунок 4.8).

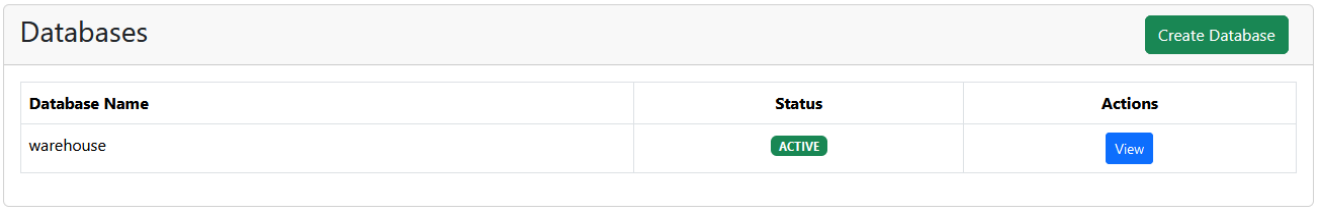

Рисунок 4.8 – Сторінка зі списком баз даних користувача

Якщо все виконано правильно, то можна переходити до наступного кроку – створення першої таблиці в новій БД.

4.2.4 Створення таблиці

Для створення таблиці необхідно натиснути кнопку “View” на сторінці з базами даних. Після чого відкриється сторінка з інформацією про сховище даних, приклад зображено на рисунку 4.9.

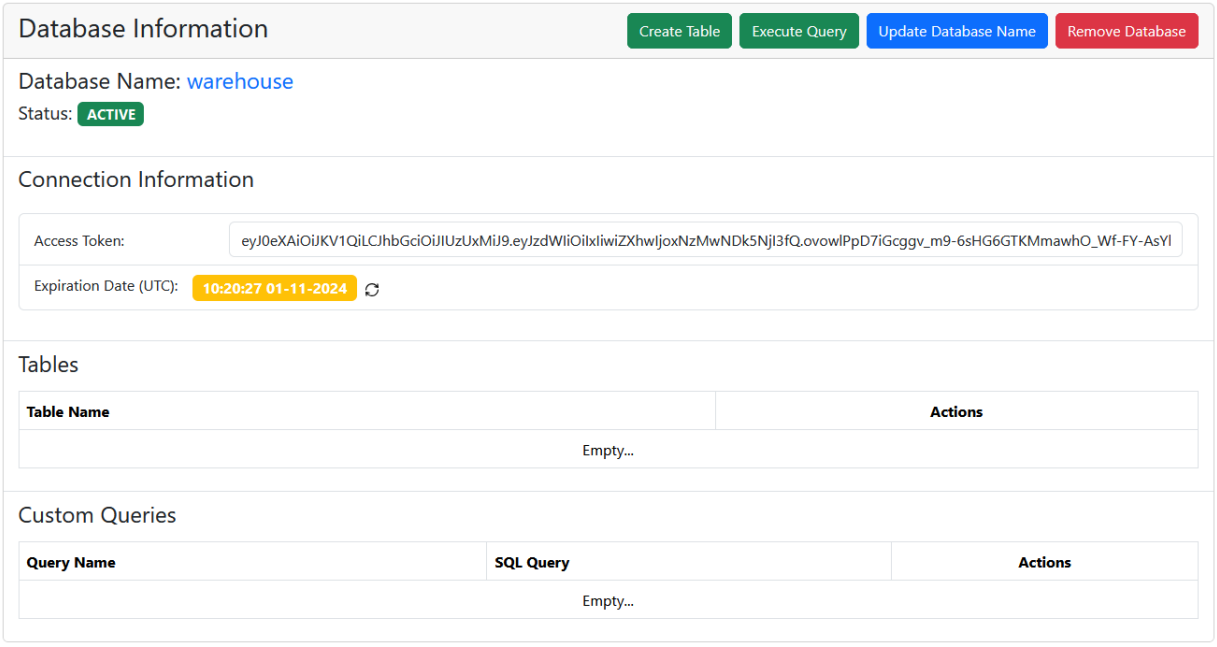

Рисунок 4.9 – Приклад сторінки з інформацією про базу даних

Використовуючи кнопку “Create Table” можна створити таблицю. Після натискання на неї відкриється форма, яку необхідно заповнити. При створенні таблиці можна передавати довільну кількість стовпців, обмеження може накладатися лише використовуваною СУБД. Приклад заповнення форми зображено на рисунку 4.10.

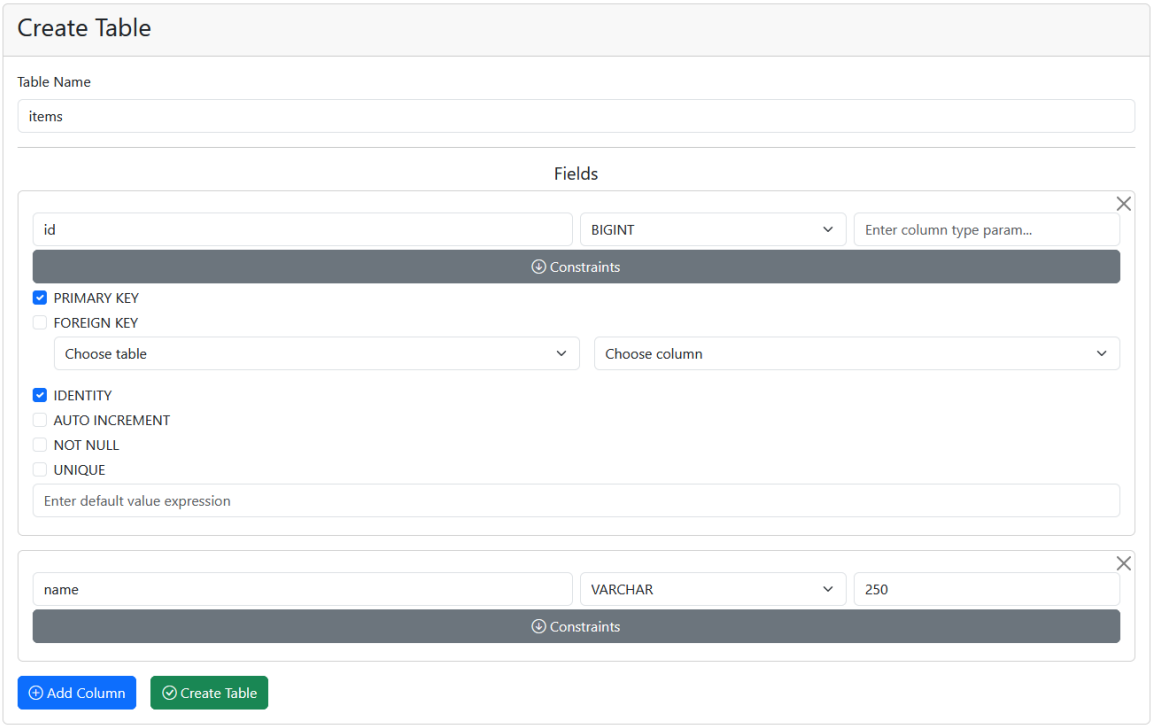

Рисунок 4.10 – Приклад заповнення форми для створення таблиці

У разу успішного створення таблиці в списку таблиць можна буде побачити новий запис (рисунок 4.11).

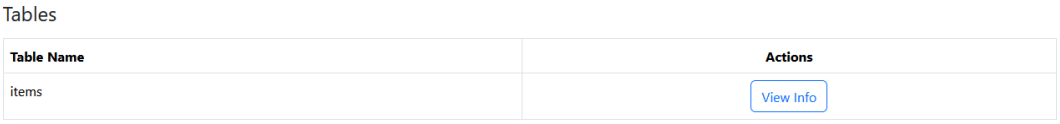

Рисунок 4.11 – Список таблиць в базі даних

Створивши таблицю можна переглянути інформацію про неї натиснувши кнопку “View Info”, де буде відображена вся інформація про таблицю, приклад такої сторінки зображено на рисунку 4.12.

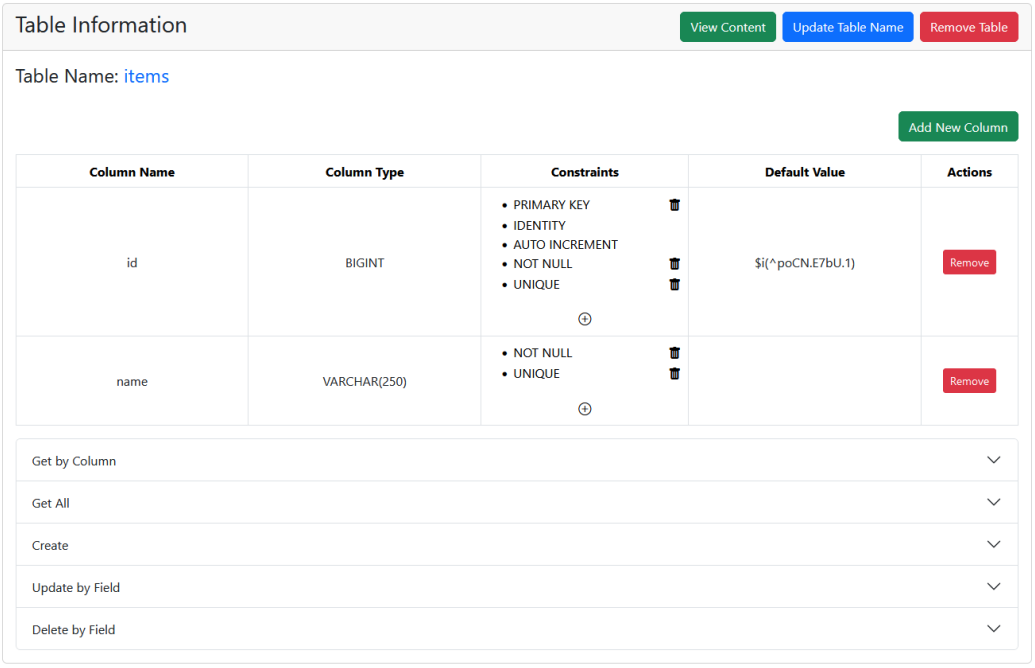

Рисунок 4.12 – Сторінка з інформацією про таблицю

З рисунку 4.12 видно, що на цій сторінці знаходиться інформація про всі наявні стовпці та їх властивості. Тут же можна відредагувати чи видалити стовпці. Також внизу сторінки знаходиться приклади до REST API, яке можна використовувати для виконання запитів до даної таблиці, щоб виконувати прості CRUD та користувацькі запити.

4.3 Інтеграція інструментів розробника реляційної бази даних за допомогою REST API

Після створення бази даних та принаймні одної таблиці для неї – відкривається доступ до універсального REST API. Використання якого, надає змогу керувати даними в таблицях, виконуючи базові CRUD операції та складні користувацькі SQL запити.

Надсилати запити можна використовуючи будь-які доступні інструменти, що дозволяють надсилати HTTP методи, наприклад, Postman. Проте основною ціллю системи є надати можливість інтегруватися з іншими сервісами користувачів, тому найкращим рішенням у даній ситуації є використання мови програмування

JavaScript, саме на ній наведені приклади інтеграції, оскільки найчастіше використовується для розробки клієнтських частин. Проте можна використовувати будь-яку мову програмування, що дозволяє відправляти HTTP запити.

Першим кроком для успішної інтеграції є перегляд документації для API, що міститься на сторінці кожної таблиці. Для прикладу візьмемо запит на створення запису в таблиці, його детально було описано в підрозділі 3.7.

Для надсилання запиту на створення запису в таблиці потрібно використовувати POST метод та вказану в документації URL. Також необхідно додати HTTP заголовок Authorization із значенням токену доступу до бази даних, якій ця таблиця належить. Тіло запиту має складатися з переліку назв параметрів у таблиці та їх значень.

Введемо всі потрібні дані в Postman для генерації запиту мовою JavaScript. Результат коду та прикладу виконання цього збереження запису в таблиці зображено на рисунку 4.13.

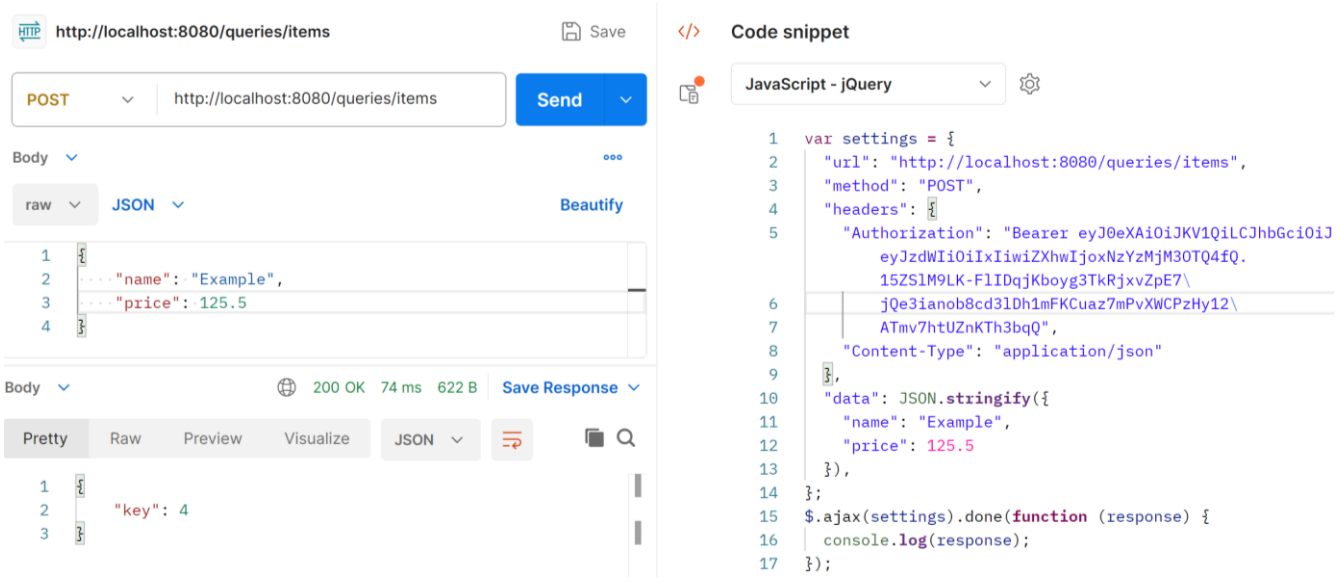

Рисунок 4.13 – Фрагмент згенерованого коду для збереження запису в таблицю

Таким чином, використовуючи документацію для кожної кінцевої точки REST API, що було описано детально в розділі 3.7, можна повністю інтегрувати інструменти розробника реляційної бази даних у власну систему для отримання

повністю функціонального програмного продукту з базою даною та серверною частиною, роль якої будуть виконувати інтегровані інструменти.

Отже, даний розділ містить інформацію про вимоги, які потрібно виконати для запуску програмного забезпечення на комп’ютері чи сервері. Дану можливість надає використана платформа Docker. За допомогою неї всі чотири компоненти можна запускати використовуючи один файл зі скриптом на будь-якій системі з однаковими налаштуваннями. Основною вимогою є встановлення Docker. Встановлення інструментів для збірки та розгортання серверної та клієнтської частини виконає сама платформа.

Також було описано базовий функціонал для роботи з інструментами розробника реляційної бази даних. Показано приклад реєстрації та входу користувача в систему, створення бази даних та таблиці в ній.

Наведено приклад фрагменту коду мовою програмування JavaScript для інтеграції універсального REST API для роботи з таблицями в базах даних у користувацькій системі, що дозволяє замінити повноцінний сервер, який з’єднаний з реляційною базою даних.

5 РОЗРОБКА СТАРТАП-ПРОЄКТУ

Даний розділ містить інформацію про маркетинговий аналіз стартап-проєкту. Аналіз проводиться з метою встановлення оцінки перспективності інструментів розробника реляційної бази даних на інформаційно технічному ринку послуг. Для впровадження готового програмного продукту визначаються можливі шлях, щоб провести даний етап успішно та ефективно.

Детальний опису проєкту чітко надає відповідь на питання: яка задачі що дозволяє вирішити розроблена система; хто є потенційними кінцевими користувачами програмного продукту; які можливості та переваги здатна запропонувати система, щоб зацікавити користувачів; які відмінності розроблених інструментів від конкурентів. Також досліджуються сучасні технології та інструменти, що здатні вирішити поставлені задачі ефективно та швидко. Встановлюється необхідність розробки додаткових, якщо вони є відсутніми на технічному ринку. Розглядаються можливі варіанти монетизації програмного продукту, оптимальна цінова політика порівняно з аналогами та шляхи збуту.

Даний підхід надає можливість створити стратегії, які дозволяють бути конкурентоспроможним, успішно розвиватися та просуватися, отримувати дохід від інструментів розробника реляційної бази даних.

5.1 Опис ідеї стартапу

Створення інструментів розробника реляційної бази даних, які дозволяють ефективно створювати бази даних без необхідних навичок, витрат значної кількості часу та в подальшому легко інтегруватися в користувацькі системи для заміни серверної частини, що відповідає за зв’язок з базою даною, є досить важливим завданням. Вони можуть бути використані в різних проєктах, які потребують підключення реляційної бази даних для керуванням інформацією з використанням CRUD операцій та складних запитів на вибірку.

Створення покращених інструментів розробника реляційної бази даних, що дозволяють ефективно керувати базами даних, таблицями та даними в них, мають

зручний прикладний програмний інтерфейс для інтеграції з іншими системами є основною задачею реалізації нового рішення. У таблиці 5.1 представлено опис основної інформації про стартап-проєкт.

Таблиця 5.1 – Основна інформацію про стартап-проєкт.

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Реалізація інструментів розробника реляційною бази даних, що надають можливості: керувати БД та їх вмістом за допомогою користувацького інтерфейсу; інтеграції за допомогою REST API з клієнтськими системами.	1. Користувацькі системи для вивчення нових технологій, не пов’язаних зі створенням серверу з БД	1. Користувач, що не володіє необхідним рівнем досвіду для створення серверів з підключенням до БД, отримує інтерфейс та API для керування БД, що повністю задовольнить його потреби
	2. Створення MVP	2. Дає можливість сфокусуватися на бізнес логікі програмного продукту, а не на технічній інфраструктурі під час створення прототипу
	3. Альтернатива CMS	3. Дозволяє більш гнучкіше створювати бази даних для виконання поставлених вимог
	4. Веб-сервіси для малого та середнього бізнесу	4. Дозволяє зекономити ресурси та витрати на створенні серверної частини

Реалізація інструментів розробника реляційної бази даних із зручним користувацьким інтерфейсом та API – дозволяє повністю керувати базою даних через веб-сайт без необхідності написання SQL запитів, а також надає можливість простої інтеграції з клієнтськими системами через REST API. Ідея розробки є в

тому, щоб полегшити розробку нових програмних продуктів, які потребують взаємодію з базами даних.

Проаналізувавши ринок програмних продуктів, що були створені зі схожою метою, було встановлено: на даний момент існує не велика кількість рішень, що дозволяють у більшій мірі виконувати поставлені задачі. Для дослідження було обрано найбільш релевантний аналог. Кожен із інструментів має як переваги, так і недоліки, що значною мірою впливають на досвід їх використання користувачами.

Дані, отримані після аналізу аналогів, було занесено в таблицю 5.2.

Таблиця 5.2 – Характеристики обраної ідеї для проєкту

№ п/п	Економічні характеристики	Концепція стратегія конкурентів		Слабкі сторони	Нейтральні сторони	Сильні сторони
		Мій проєкт	Supabase			
1	Підтримка БД	InterSystems IRIS	PostgreSQL		+	
2	Головні ключі для таблиць	Надається можливість додавання будь-якого головного ключа (назва, тип), або взагалі не додавати	Головний ключ завжди створюється з назвою id			+
3	Зовнішні зв’язки між таблицями	Дозволяє створювати за будь-якими унікальними індексами	Підтримує зв’язок лише за головним ключем			+
4	Підтримка користувацьких запитів	Дозволяє зберігати складні запити на вибірку даних із таблиць у БД	Не підтримує даної можливості			+
5	Підтримувані API для інтеграції	REST API	REST, GraphQL APIs	+		

Отже, дослідивши та порівнявши переваги та недоліки конкурентів та розроблюваних інструментів розробника реляційної бази даних, було визначено технічні та економічні переваги обраної ідеї. Спираючись на отримані дані, можна отримати заключення, що задум реалізації програмного продукту має гарні шанси отримати підтримку клієнтів у своїй ринковій ніші зі схвальними відгуками під час релізу. На це впливає концентрація на ключових задачах: самостійно обирати структури, головних та зовнішніх ключів для таблиць при створенні та редагуванні; зберігати та виконувати користувацькі запити через REST API.

5.2 Технологічний аудит для ідеї стартапу

Реалізація кінцевого програмного продукту потребує проведення огляду існуючих технічних інструментів, що дозволяють розробляти ідеї, які були визначені для стартапу, та визначення їх доступності (необхідності купувати ліцензію чи права) для використання під час створення кінцевого програмного продукту. Результати для проведеного аудиту технологій знаходяться у таблиці 5.3.

Таблиця 5.3 – Результати аудиту технологій для реалізації ідеї

№ п/п	Ідея проєкту	Технології для реалізації	Наявність технології	Доступність технології
1	Інтерфейс для взаємодії користувача з системою	HTML, SCSS, JavaScript (TypeScript, Angular)	Наявна	Доступі та безкоштовні
2	Реалізація серверу та бізнес логіки	Java (Spring Framework)	Наявна	Доступі та безкоштовні
3	Збереження постійних даних	InterSystems IRIS	Наявна	Доступна та безкоштовна
4	Збереження кешу про авторизацію користувачів	Redis	Наявна	Доступна та безкоштовна
5	Автоматизація розгортання компонентів	Docker	Наявна	Доступна та безкоштовна

Технології та інструменти, що були обрані під час проведення аудиту, забезпечують успішність та ефективність реалізації інструментів розробника реляційної бази даних. Вони мають ключові характеристики, що дозволяють спростити процес розробки, виконувати поставлені задачі швидко та надійно за рахунок їх поширеному застосуванні серед користувачів та якісній документації.

5.3 Дослідження ринкових можливостей для запуску стартап-проєкту

Запуск програмного продукту на ринку несе за собою багато ризиків та загроз. Вони можуть спричинити великі проблеми на шляху реалізації задумів. Тому необхідно проводити аналіз можливостей, що надає ринок. Їх використання дає змогу краще підійти до питання побудови планів розвитку кінцевого рішення за рахунок урахування особливостей ринку конкретної області, потреб та побажань клієнтів, переваг та недоліків конкурентів.

Аналіз розпочався з проведення попередньої характеристики ринкової сфери для інструментів розробника реляційної бази даних, результати були занесені в таблицю 5.4.

Таблиця 5.4 – Попередня характеристика ринкової області для стартапу

№ п/п	Показники стану ринку	Характеристика
1	Кількість ключових опонентів на ринку, од	4
2	Сумарний оборот продажів, грн/ум. од	Невідомий
3	Ринкова динаміка (якісна оцінка)	Попит зростає, пропозиція ні
4	Наявність обмежень для виходу на ринок (вказати характер обмежень)	Відсутні
5	Специфічні умови для стандартизації та сертифікації	Відсутні
6	Медіана норми рентабельності в ринковій області, %	Висока, 70%

З отриманих результатів аналізу передніх ринкових характеристик у таблиці 5.4 можна зробити висновок, що обрана область інформаційно технологічної сфери є приваблива для виходу на неї з стартапом.

Наступним кроком дослідження ринкових можливостей є визначення потенційних категорій користувачів програмного продукту. Кожній групі надається детальний опис та встановлюється попередній список вимог до кінцевого продукту. Таблиця 5.5 містить результат даного етапу.

Таблиця 5.5 – Опис потенційних користувачів

№ п/п	Потреба, яка формує ринок	Цільові категорії користувачів (сегменти ринку)	Відмінності поведінки для різних потенційно цільових груп клієнтів	Вимоги користувачів до кінцевого продукту
1	Створення та керування БД з використанням інтерфейсу користувача, можливість виконувати операції з даними в таблицях за допомогою спеціального API	Малий та середній бізнес; початківці, що створюють веб-сервіси; стартапи (для створення MVP)	1. Тривалість використання БД для зберігання даних. 2. Обсяги даних у БД. 3. Кількість запитів на API.	Стабільність роботи системи, швидкість оброки запитів, використання системи безкоштовно з обмеженими можливостями, захист даних

Виходячи з результатів аналізу на даному етапі можна сказати, що цільовою аудиторією подібних систем є користувачі, що створюють не навантажені системи.

Складання таблиць факторів, які допомагають ефективно проводити впровадження проєкту на ринок та навпаки цьому заважають цьому, є наступним кроком після з’ясування цільових категорій клієнтів.

Отримані дані було занесено у відповідні таблиці 5.6 (фактори загроз) та 5.7 (фактори сприяння успішності).

Таблиця 5.6 – Фактори загроз

№ п/п	Назва фактору	Опис загрози	Можлива реакція компанії
1	Підтримувані БД	Існують конкуренти, що підтримують різні реляційні та нереляційні бази даних для збереження даних	Реалізація підтримки нових популярних баз даних у системі
2	Неповноцінна підтримка всіх типів користувацьких запитів	Відсутність підтримки користувацьких запитів для запису, редагування та видалення запитів	Доопрацювання існуючого функціоналу та надання користувачам права виконувати такі запити
3	Недостатня фінансова та медійна підтримка	Залучення нових інвестиційних надходжень є складним процесом; обмежена кількість цільових користувачів через обмежені фінансові надходження для проведення рекламних компаній	Відкриття інших сфер з яких можна залучати інвесторів; використання безкоштовних ресурсів для реклами

Таблиця 5.7 – Фактори сприяння успішності

№ п/п	Назва фактору	Опис можливості	Можлива реакція компанії
1	Відсутність обмежень при створенні структур таблиць	Користувача надається повне право створювати будь-яку таблицю з використанням підтримуваних можливостей конкретної СУБД	Надання ще більших можливостей клієнтам в створенні таблиць
2	Зручний користувацький інтерфейс з великим функціоналом	Інструменти дозволяють повністю створювати БД та їх вміст не виходячи з користувацького інтерфейсу	Покращення інтерфейсу, додавання ще більше можливостей для спрощення створення та керування БД
3	Підтримка складних запитів на вибірку даних	Інструменти дозволяють зберігати та використовувати користувацькі запити на вибірку	Розширення існуючого функціоналу

У результаті було отримано фактори, що дозволять ефективно зайти на ринок, а також ті, на яких потрібно сконцентруватися після запуску та покращувати для кращого просування на ринку.

Наступним етапом є дослідження пропозицій. На цьому кроці потрібно з’ясувати особливості конкуренції в даній області ринку. Використовуючи існуючі типи конкуренції, проводиться аналіз пропозиції. Результат зображено в таблиці 5.8.

Таблиця 5.8 – Ступеневий аналіз ринкової конкуренції

Особливості конкурентного середовища	У чому відбувається проявлення даної ознаки	Вплив на життєдіяльність підприємства (можливі кроки компанії, що дають змогу бути конкурентоспроможною)
Монополістична конкуренція	Велика кількість гравців, що пропонують подібні можливості проте фокусуються на різні цільові аудиторії, мають відмінну цінову позицію	Покращення сервісу спираючись на пропозиції та відгуки клієнтів, реалізація нового функціоналу
Міжнародний рівень конкурентної боротьби	Подібні програмні продукти існують у різних країнах	Адаптація під світові реалії та виклики
Має міжгалузеву ознаку	Бази даних присутні в різних галузях	Робити систему універсальною, адаптуючись під специфічні сфери застосування
Товарно видова конкуренція за видами товарів	Змагання між конкурентами зі схожим набором функціоналу	Акцентування на конкретній цільовій аудиторії, створення простого та ефективного рішення
Нецiновий характер конкурентних переваг	Перевагами системи є не ціна, а функціональність та підтримка унікальних можливостей	Покращення існуючого функціоналу та додавання нового на основі клієнтських запитів
Марочна інтенсивність	Власний бренд для пізнаваності на ринку	Рекламні кампанії для просування бренду

У результаті виконання ступеневого аналізу було сформовано шляхи просування продукту в середовищі, яке має конкуренцію.

Наступним кроком є необхідність проведення глибшого та детальнішого аналізу умов конкуренції у цільовій сфері. Стартап фокусується на створенні бренду, який вирізняється на ринку своєю якістю та простотою системи, надійністю роботи, фокусуванням на потреби клієнтів. Ці ключові особливості дозволяють бути впізнаваним попри наявність аналогів, що існують на ринку тривалий час.

Було проведено конкурентний аналіз за М. Портером, результати зображено в таблиці 5.9.

Таблиця 5.9 – Конкурентний аналіз області за М. Портером

	Прямі конкуренти	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Ринок України не має схожих альтернатив, проте світовий ринок має подібні програмні продукти (Firebase, Supabase, Hasura)	Ймовірність появи нових рішень є середньою	Невелика залежність від використовуваних СУБД, а також хмарних сервісів для розгортання	Широкий вибір аналогів, бажання доступних та гнучких рішень, відтік користувачів через відсутність потрібного функціоналу	Ручна розробка серверної частини, готові CMS, використання таблиць замість БД
Висновки	Конкуренція є високою	Існує загроза появи нових рішень	Фокусування на відритих та безкоштовних інструментах, використання дешевших хмарних рішень	Створення розширеного функціоналу, надання можливості безкоштовного використання, акцент на автоматизації процесів	Спрощення механізму керування та інтеграції дасть можливість отримати додаткові симпатії

Провівши аналіз можна прийти до висновку, що немає універсальних продуктів, що могли б забезпечити потреби користувача на повну. Оглянуті системи мають як позитивні, так і негативні риси, проте це ніяким чином не змінює ситуацію на ринку. Тобто можна сказати, що запуск нового проєкту в даній ринковій ніші є можливим. Для збільшення шансу успішного старту потрібно лише запропонувати функціонал, що аналоги не мають, а також покращити вже реалізовані ними функцію, створити приємні умови для задоволення побажань клієнтів. Під час входу на ринок насамперед потрібно розуміти сильні сторони проєкту, що виокремлюють його серед інших, та фокусуватися на них.

У таблиці 5.10 наведено фактори та їх обґрунтування, що дозволяють новому проєкту бути стійким до конкуренції, а також отримати схвальні відгуки від клієнтів.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Назва фактору	Обґрунтування (визначення чинників, які надають описуваному фактору вагомості для порівняння конкурентних проєктів)
1	Забезпечення безпеки даних	Захист даних реалізований з використанням останніх Інтернет стандартів
2	Простий та зрозумілий інтерфейс для користувача	Інтерфейс користувача не має зайвих елементів, щоб могли його перевантажити, а містить лише потрібні елементи
3	Підтримка користувацьких запитів	Продукт дозволяє зберігати та використовувати складні користувацькі запити на вибірку
4	Відсутність обмежень у створенні структури таблиці	Користувач має змогу самостійно обирати структуру для створюваної таблиці
5	Низька ціна та безкоштовний функціонал	Ціна на послуги є меншою за конкурентів, а також присутня можливість використання безкоштовних ресурсів
6	Орієнтація на потреби користувач	Додавання функціоналу, який є бажаним від клієнта
7	Універсальне використання	Клієнт ніяк не обмежений у способі використанні та інтеграції інструментів у власну систему для будь-якої предметної області

Використовуючи дані з таблиці 5.10 було виконано оцінку переваг та недоліків стартап-проєкту, результати було занесено у таблицю 5.11.

Таблиця 5.11 – Оцінка переваг та недоліків стартап-проєкту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг порівняння товарів від конкурентів з інструментами розробника реляційної БД (даний продукт)						
			-3	-2	-1	0	1	2	3
1	Забезпечення безпеки даних	15					+		
2	Простий та зрозумілий інтерфейс для користувача	12				+			
3	Підтримка користувацьких запитів	20	+						
4	Відсутність обмежень при створенні структури таблиці	16	+						
5	Низька ціна та безкоштовний функціонал	8			+				
6	Орієнтація на потреби користувач	5				+			
7	Універсальне використання	15				+			

Заключним етапом проведення аналізу ринкових перспектив реалізації проєкту виконання SWOT-аналізу, іншими словами складання матриці для аналізу слабких (Strength) та слабких (Weak) сторін, а також загроз (Troubles) та можливостей (Opportunities) за допомогою використання даних отриманих із таблиці 5.11. Результати даного аналізу наведено в таблиці 5.12.

Таблиця 5.12 – Результати SWOT-аналізу стартап-проєкту

Сильні сторони: - підтримка користувацьких запитів; - відсутність обмежень при створенні структури таблиці;	Слабкі сторони: - забезпечення безпеки даних; - відсутність репутації для бренду.

Кінець таблиці 5.12

- низька ціна та безкоштовний функціонал.	
Можливості: - створення та управління складними користувацькими запитами для вибірки даних; - відсутність необхідності використання SQL запитів для створення та керування БД, даними.	Загрози: - конкуренція не занадто висока, але вже є відомі готові рішення; - складність створення рекламної кампанії.

Наступним кроком є визначення можливих інших шляхів для успішного впровадження стартапу на ринку. Для кожної такої можливості встановлюється термін, який знадобиться на повну її реалізацію, а також ймовірність з цього отримати ресурси.

Таблиця 5.13 – Альтернативи для впровадження стартап-проєкту на ринок

№ п/п	Назва альтернативи (можливий комплекс заходів) для ринкової поведінки	Ймовірність отримання ресурсів	Час, потрібний для реалізації
1	Концентрація на потребах малого та середнього бізнесу	70%, більше такого бізнесу захоче інтегрувати продукт у власну систему, оскільки для них будуть створенні кращі умови	1 рік
2	Орієнтація на універсальності	90%, дозволить будь-кому використовувати систему, виключивши більшість проблем, які можуть виникнути з конкретними типами даних	2 роки
3	Проведення більш ширшої рекламної кампанії для просування існуючих можливостей	50%, більше потенційних клієнтів дізнаються про існуючі можливості програмного продукту	6 місяців

З отриманих результатів у таблиці 5.13 можна побачити можливі шляхи на яких варто сконцентруватися у разі необхідності для успішного впровадження програмного продукту на ринку. Кожен такий варіант має свій термін реалізації, а також ймовірність успішного залучення нових клієнтів. Найбільшу ймовірність, 90%, має альтернатива під назвою “Орієнтація на універсальність”, адже дозволить вирішити майже всі проблеми, які можуть виникнути у клієнтів з особливими системами під час використання та інтеграції програмного продукту. Однак реалізація є довгою, 2 роки, оскільки потребує детального розбору всіх нюансів.

Іншим варіантом, який має дещо меншу ймовірність, 70%, та менший термін на впровадження – 1 рік, є “Концентрація на потребах малого та середнього бізнесу”, оскільки для їх переходу на розроблений продукт будуть створені найкращі умови.

Третій варіант проведення широкої рекламної кампанії не відкриє нових горизонтів для розвитку, проте дозволить залучити потенційних клієнтів для існуючого функціоналу з ймовірністю успіху в 50% та терміном на реалізацію в 6 місяців.

5.4 Створення ринкової стратегії для стартап-проєкту

Після визначення ринкових можливостей, що допоможуть запустити стартап ефективно, необхідно розробити стратегії запуску програмного продукту на ринку. Для цього необхідно виконати ряд кроків. Першим із яких є створення детального опису кожної групи потенційних споживачів. Метою проведення цього етапу є з’ясування цільових груп, яким буде запропоновано використання нового програмного продукту; визначення стратегії, що дозволить охопити більшу частину ринку.

Відповідно у таблиці 5.14 наведено результати створення опису для цільових груп.

Таблиця 5.14 – Характеристика цільових груп споживачів

№ п/п	Характеристика профілю для цільової групи потенційних користувачів	Готовність користувачів прийняти продукт	Очікуваний попит від цільової групи (сегмента)	Рівень конкуренції у сегменті	Легкість входу у сегмент
1	Малий та середній бізнес	Готові (такий бізнес зацікавлений у зменшенні витрат на технічному ресурсі)	80%	Середня	Легко
2	Клієнти з власними проєктами для навчання	Готові (дозволить концентруватися на вивченні нових технологій, використовуючи готове рішення для управління даними)	60%	Відсутня	Легко
3	Користувачі CMS	Готові (дозволить більш гнучко налаштовувати структури для даних)	50%	Мала	Легко
4	Команди для створення MVP	Готові (дозволить менше фокусуватися на технічному аспекті при створенні MVP)	40%	Мала	Легко
Цільові групи, які було обрано: малий та середній бізнес, клієнти з власними проєктами					

Визначившись з потенційними клієнтськими групами, необхідно обрати цільову групу, яка буде обрана за головну, і для неї буде запропонований

реалізований продукт, а також концентруючись на ній буде розроблена стратегія охоплення ринку. На практиці існують три такі стратегії.

Масовий маркетинг: розглядати ринок потрібно, як одне ціле, не ділячи його на окремі частини за відмінностями. Даний підхід дозволяє охопити ширшу аудиторію, спростити маркетингові зусилля та виробництво.

Диференційований маркетинг: у цьому випадку ринок розділяється на умовні сегменти, для кожного з них створюється своя маркетингова стратегія, що дозволяє охоплювати декілька таких ринкових часток, і при невдачі в одній, не зазнати її в іншому сегменті.

Цільовий маркетинг: уся увага концентрується на одному ринковому сегменті з використанням спеціалізованого маркетингу, який задовольняє вимоги даної частки. Є корисним для компаній, що мають вузьку спеціалізацію.

Кожен із цих підходів є гарним у різних ситуаціях та для різних компаній, тому при виборі потрібно керуватися характеристиками продукту, що планується запускати.

У таблиці 5.15 наведено визначення базової стратегії для розвитку, що дозволяє зобразити підхід для охоплення ринку та дасть змогу встановити зв’язки з цільовою клієнтською групою для подальшої взаємодії.

Таблиця 5.15 – Визначення базової стратегії для розвитку

№ п/п	Обрана альтернатива для розвитку проєкту	Стратегія для охоплення ринку	Ключові конкурентоспроможні пропозиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Реалізація продукту, що є універсальним для всіх сфер ринку	Стратегія диференційованого маркетингу	Створення універсального функціоналу, що зможе зацікавити більшість потенційних клієнтів	Стратегія диференціації
2	Концентрація на побажаннях клієнтів та їх реалізація	Стратегія диференційованого маркетингу	Прислухання до побажань клієнтів підвищить репутацію	Стратегія диференціації

З таблиці 5.15 видно, що за фундаментальну стратегію розподілення ринку на сегменти, тобто диференційний підхід, звідси випливає, що основна охоплення ринку відбуватиметься за допомогою диференційованого маркетингового підходу.

Наступним етапом є обрання стратегії для поведінки, щоб бути конкурентоспроможним на ринку. Результати проведеного визначення наведено у таблиці 5.16.

Таблиця 5.16 – Визначення базової стратегії для конкурентної поведінки

№ п/п	Чи є проєкт першим на ринку?	Чи буде компанія залучати нових споживачів, чи відбирати існуючих від конкурентів?	Чи буде компанія дублювати основні можливості конкурента, і які?	Стратегія конкурентної поведінки
1	Ні, у цій сфері на ринку існують аналоги	Головною метою є залучення нових користувачів, відкриття для них нових можливостей, можливі випадки добровільного переходу через зручність для клієнтів	Компанія не збирається повністю дублювати конкурентів, проте основний функціонал може бути схожим	Стратегія, що базується на занятті конкурентної ніші

Після встановлення стратегії для розвитку на ринку та потребах від цільових груп, на її основі будується наступна стратегія – позиціювання, що наведена у таблиці 5.17. Вона дозволяє сформувати в клієнтів образ для бренду, який в свою чергу буде вирізняти продукт серед конкурентів, та дозволить закріпитися на конкурентних позиціях на ринку.

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару від цільової аудиторії	Базова стратегія для розвитку	Основні конкурентоспроможні позиції	Вибір асоціацій, що дозволяють сформувати комплексну позицію
1	Підтримка користувацьких запитів	Стратегія диференціації	Дозволяє клієнтам створювати власні запити для вибірки та використовувати їх	Унікальність, ексклюзивність, оригінальність
2	Можливість самостійно створювати структури для таблиць та зв’язки між ними	Стратегія диференціації	Гнучкість при створенні таблиць	Гнучкість, унікальність, ефективність, оригінальність

У результаті проведення комплексного аналізу було встановлено, що основною стратегією для розвитку на ринку є диференційний підхід. Використовуючи дану стратегію необхідно розділити ринок на сегмент, створювати маркетинг для кожного сегменту окремо, щоб не зазнати невдачі відразу на всьому ринку, а також реалізовувати унікальний продукт, який буде відрізнятися від конкурентів.

Для конкурентної поведінки було обрано стратегію заняття конкурентної ніші, як базову. Розроблюваний продукт має відповідати наступним вимогам: гнучкість, унікальність, якість.

5.5 Створення маркетингової програми стартап-проєкту

На основі результатів попереднього аналізу конкурентоспроможності підводиться підсумок, що дозволяє визначити ключові переваги для концепції потенційного товару (таблиця 5.18).

Таблиця 5.18 – Визначення основних переваг концепції потенційного товару

№ п/п	Потреба	Вигода, що товар може запропонувати	Основні переваги перед аналогами (існуючі або ті, що необхідно реалізувати)
1	Створення та використання користувацьких запитів	Самостійне створення потрібного запиту на вибірку даних з таблиць для зручності інтеграції в системі	Конкуренти не підтримують даний функціонал по створенню та використанню користувацьких запитів
2	Відсутність обмежень при створенні таблиць та зовнішніх зв’язків	Вимоги клієнтів до таблиць чи зв’язків при створенні не обмежуються	Конкуренти, що мають такий функціонал у певній мірі обмежують користувача у структурі таблиць та використанні зовнішніх зв’язків

Наступним кроком є розробка маркетингової моделі для товару, що складається з трьох рівні: надається ідея для продукту, визначаються його фізичні складові; встановлюються особливості в процесі його надання.

Таблиця 5.19 – Опис трирівневої моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Інструменти розробника реляційної бази даних		
II. Товар під час реального виконання	Характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Безпека даних	М	Тх
	Підтримка користувацьких запитів	М	Тх, Е
	Відсутність обмежень при створенні таблиць чи зовнішніх ключів	М	Тх, Е
	Підтримка прикладного програмного інтерфейсу	М	Тх
	Відповідає Інтернет стандартам по захисту користувацьких даних		
	Пакування: використання можливе за допомогою веб-браузеру через особистий кабінет після реєстрації		

Кінець таблиці 5.19

	Марка: DbManagementTool
III. Товар із підкріпленням	Клієнт має можливість познайомитися з функціоналом товару використовуючи наукові публікації та безкоштовну версію. Для уточнення функціоналу та надання побажань можливе звернення до компанії. У разі отримання прохання на доопрацювання функціоналу для заключення продажу чи його продовження, компанія може це виконати
Товар буде захищено від копіювання за допомогою використання ліцензії MIT та захисту інтелектуальної власності	

Визначення цінових меж, які необхідно використовувати для встановлення вартості потенційного товару, є наступним кроком, зображено у таблиці 5.20.

Таблиця 5.20 – Визначення меж для встановлення вартості

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової аудиторії	Нижня та верхня межі для встановлення вартості товару
$ 0-50	$ 25 - 100	$ 10 - 20 000	$ 5 - 15

Наступним етапом є визначення оптимальної системи для збуту товару, результати даного процесу зображені в таблиці 5.21.

Таблиця 5.21 – Формування системи збуту

№ п/п	Особливості поведінки цільових клієнтів під час закупівель	Функції зі збуту, що повинен виконувати постачальник товару	Глибина каналу для збуту	Оптимальна система для збуту
1	Продукт надається в обмеженій/безкоштовній та повній/платній версії. Сплата відбувається через платіжні сервіси	Підтримка налаштування, доопрацювання системи в разі критичної необхідності	Прямий канал збуту, оскільки доступ отримується онлайн без посередників	Безкоштовна обмежена версія для залучення клієнтів, цифровий маркетинг, розвиток спільноти

Система збуту, що представлена в таблиці 5.21 відображає особливості даного процесу. З таблиці видно, що оптимальним підходом для її реалізації є створення безкоштовного обмеженого функціоналу для залучення нових клієнтів, розвиток маркетингу та спільноти, що буде розповідати про переваги продукту.

Кінцевим етапом та складовою для створення маркетингової програми є розробка концепції комунікаційної стратегії (таблиця 5.22), що базується на попередніх результатах: основа для позиціювання; специфіку клієнтської поведінки.

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Особливості клієнтської поведінки	Канали для комунікації, що використовуються клієнтами	Основні позиції, що обрані для позиціювання	Завдання повідомлень для реклами	Концепція для рекламного звернення
1	Клієнти відкривають для себе продукт з наукових праць, від рекомендацій спільноти, після чого відвідують офіційний сайт для особистого знайомства та заключення співпраці	Електронні пошти, менеджери	Унікальність, ефективність, оригінальність, гнучкість, клієнтоорієнтованість	Розповісти про переваги використання готових рішень для заміни веб-серверів у клієнтських системах	Створення та управління базами даних з можливістю інтеграції в власну систему за допомогою REST API

У результаті було отримано концепцію, на ній потрібно будувати маркетингові комунікації з потенційними клієнтами. Обрано сильні сторони та

можливості, на яких потрібно концентруватися в рекламній кампанії, щоб досягнути кращих результатів.

Отже, аналіз, що було проведено в цьому розділі, для визначення потенційної маркетингової стратегії для інструментів розробника реляційної бази даних, дозволив виявити можливості для ефективного впровадження програмного продукту на IT ринку, а також способи, якими можна дані впровадження виконати. Кожен етап аналізу було описано, а визначення результатів представлено у вигляді таблиць.

У першу чергу було створено детальний опис для ідеї проєкту з детальною характеристикою та визначенням базових напрямків розвитку.

Наступним етапом було проведення аудиту, що дозволив визначити інструменти, які дозволять реалізувати задум. Визначено їх наявність, ціну, а також ідеї, які вони дозволять реалізувати.

Потім було визначено ринкові можливості для ефективного запуску стартапу. Під час цього етапу було досліджено ринок та конкурентів, які на ньому існують. Описано фактори загроз та можливостей для програмного продукту. Проведено порівняння та оцінку функціоналу, що здатний зацікавити користувача, у порівнянні з аналогами. Встановлено альтернативні шляхи розвитку та конкурентоспроможності.

На основі даних отриманих на попередніх етапах - створено ринкову стратегію для стартапу, що включає в себе наступні стратегії: розвитку, конкурентної поведінки та позиціонування.

Використовуючи всі отримані результати було розроблено маркетингову програму, що включала в себе: побудову трирівневої моделі; визначення цінової політики, ринків збуту, спроможностей розробленого функціоналу та концепції для проведення маркетингових комунікацій.

ВИСНОВКИ

У результаті виконання магістерської дисертації було створено інструменти розробника реляційної бази даних, для чого було проаналізовано існуючі програмні рішення, виявлено їх переваги та недоліки, та поставлено задачу розробки; проведено дослідження сучасних засобів та технологій для розробки, які дозволять ефективно вирішити поставлену мету; спроєктовано архітектуру та структуру системи; реалізовано програмний продукт та проведено його тестування та апробацію.

Реалізацію інструментів розробника реляційної бази даних було виконано за допомогою використання наступних методів та засобів: мов програмування Java для реалізації серверної частини, JavaScript – клієнтської частини; фреймворки Spring та Angular відповідно, оскільки саме вони мали весь необхідний функціонал, що дозволило виконати поставлені задачі швидко та ефективно.

Використовувані дані в системі були розподілені за їх спільними властивостями, у результаті було утворено дві групи: постійну, яка включає інформацію про користувачів, бази даних, таблиці, запити та токени доступу до БД; тимчасову – авторизаційні JSON Web Token користувачів. Для управління цими даними було використано два різних сховища даних: InterSystems IRIS та Redis. Вони різняться підходами збереження та швидкості обробки інформації. Для забезпечення надійності та цілісності постійних даних було використано InterSystems IRIS. У випадку тимчасових даних, які потребують швидкості в їх обробці – Redis.

На етапі проектування інструментів розробника реляційної бази даних було прийнято рішення використовувати клієнт-серверну архітектуру. Що дозволило розділити обов’язки, які має виконувати кожна з частин, та дозволити в подальшому масштабуватися за необхідності без проблем. У свою чергу серверну частину було реалізовано з використанням багатошарової архітектури, бо вона дозволяє розділяти різні види логік, наприклад, бізнес від збереження даних. Для

клієнтської частини було використані підхід односторінкового застосунку (Single Page Application) та компонентно-орієнтованої архітектури.

Для забезпечення можливості інтеграції розроблених інструментів з існуючими системами було розроблено універсальне API з використанням підходу REST, оскільки має ряд переваг над альтернативами: швидкість та простота в реалізації; слабкий взаємозв'язок. Воно дозволяє користувачу виконувати базові CRUD операції та власні запити на вибірку даних до таблиць з баз даних за допомогою звичайних HTTP запитів.

За захист даних та кінцевих точок у системі відповідає модуль Spring Security. Він дозволив налаштувати автентифікацію в системі за допомогою JSON Web Token. Перевагою JWT є можливість вкладати в його тіло потрібні значення, які в подальшому можна використовувати, що дозволить ефективно виконувати перевірку доступу. Тому для отримання можливості повністю взаємодіяти з системою користувачу необхідно створити обліковий запис та авторизуватися, під час чого буде створено новий унікальний токен, який відповідатиме одному клієнту та забезпечить доступ до конференційних даних. Для виконання запитів через REST API за даними в таблицях також використовується JWT, який генерується для кожної БД окремо та має свій час життя: день, тиждень, місяць та рік.

Було розроблено ефективний алгоритм створення нових баз даних для користувачів. Оскільки під час цього процесу відбувається багато дій, які мають відбуватися асинхронно для забезпечення швидкодії та зменшення часу очікування користувача. До таких дій можна віднести створення токену доступу до бази даних, цей процес може відбуватися в окремому потоці та не змушувати клієнта чекати на його завершення, як тільки база даних була створена – це дозволяє користувачу продовжити взаємодію з системою, проте доступ до REST API буде тимчасово недоступним.

Для полегшення розгортання системи на будь-якій системі було використано платформу Docker. За допомогою спеціального файлу та мови було описано набір команд, які виконуються послідовно та автоматично для встановлення всіх необхідних бібліотек, баз даних, збирання та запуску програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 2022 App vs. Website Trend Report. *AMPLITUDE LAB* : веб-сайт. URL: https://amplitude.com/guides/2022-app-vs-website-report (дата звернення: 26.10.2024).

2. Realtime Database Documentation. *Firebase Documentation* : веб-сайт. URL: https://firebase.google.com/docs/database (дата звернення: 20.11.2024).

3. Hasura DDN Documentation. *Hasura Docs* : веб-сайт. URL https://hasura.io/docs/3.0/index/ (дата звернення: 20.11.2024).

4. Strapi Documentation. *Strapi Docs 5* : веб-сайт. URL: https://docs.strapi.io/ (дата звернення: 20.11.2024).

5. Supabase Documentation. *Supabase Docs* : веб-сайт. URL: https://supabase.com/docs (дата звернення: 20.11.2024).

6. Васильєв О. М. Програмування мовою Java. Тернопіль : Навчальна книга – Богдан, 2019. 696 с.

7. Scott A. D., MacDonald M., Power S. JavaScript Cookbook: Programming the Web. 3rd ed. O'Reilly Media, 2021. 538 p.

8. Documentation. *SASS LANG DOCUMENTATION* : веб-сайт. URL: https://sass-lang.com/documentation/ (дата звернення: 20.11.2024).

9. Walls C. Spring in Action. 6th ed. Manning Publications Co. LLC, 2022. 520 p.

10. Musib S. Spring Boot in Practice. 1st ed. Manning Publications Co. LLC, 2022. 584 p.

11. Heckers R. Effective Angular. Packt Publishing, 2024. 400 p.

12. Ricardo C. M., Urban S. D., Davis K. C. Databases Illuminated. Jones & Bartlett Learning, 2022. 682 p.

13. Kulkarni C., Kale S. D. Redis: Deep Dive. Amazon Digital Services LLC, 2021. 230 p.

14. Jankov T. HTTP/3 Explained. Packt Publishing, 2022. 257 p.

15. Sharma S. Modern API Development with Spring and Spring Boot. Packt Publishing, 2021. 582 p.

16. Madden N. API Security in Action. Manning Publications Co. LLC, 2020. 576 p.

17. What is Docker? *Docs Docker* : веб-сайт. URL: https://docs.docker.com/get-started/docker-overview/ (дата звернення: 26.10.2024).

18. Docker Compose overview. *Docs Docker* : веб-сайт. URL: https://docs.docker.com/compose/ (дата звернення: 26.10.2024).

19. Christudas B. Practical Microservices Architectural Patterns. Apress, 2019. 902 p.

20. Посвістак В. С., Демківська Т. І. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення. *Інформаційні технології в науці, виробництві та підприємництві : зб. наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій* / за заг. наук. ред. В. Ю. Щербаня. Київ: Освіта України ; ФОП Маслаков, 2020. С. 78-81.

21. Kundu S. Modern Software Engineering Guidebook. Bpb Publications, 2024. 364 p.

22. Stephens R. Beginning Database Design Solutions. Wiley, 2023. 736 p.

23. Spica L. Spring Security in Action. 2nd ed. Manning Publications Co. LLC, 2024. 440 p.

24. What is the Windows Subsystem for Linux? *Learn Miscrosoft* : веб-сайт. URL: https://learn.microsoft.com/en-us/windows/wsl/about (дата звернення: 26.10.2024).

25. Introduction to Docker Desktop. *Docker Desktop* : веб-сайт. URL: https://www.docker.com/products/docker-desktop/ (дата звернення: 26.10.2024).

ДОДАТОК А

ІНСТРУМЕНТИ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Апробація наукових досліджень

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТР-31мп_24

Аркушів 4

Київ – 2024

Міщенко Андрій Андрійович, здобувач магістерського освітнього ступеня інституту атомної та теплової енергетики
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

Науковий керівник: Михайлова Ірина Юріївна, канд. техн. наук, доцент кафедри цифрових технологій в енергетиці
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

ІНСТРУМЕНТИ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

З розвитком та популяризацією Інтернет технологій почало з'являтися все більше веб-систем, які дають змогу з будь-якої точки світу за наявності з'єднання отримувати доступ та працювати з ними. Створення таких веб-системи не є простим завданням, оскільки кожна така система має складатися з декількох частин (клієнтська та серверна), кожна із яких має виконувати свою задачу. А найголовніше, ці системи нерідко мають додаткові зовнішні з'єднання, серед яких є бази даних.

Тож при розробці веб-систем чимала кількість розробників стикається з необхідністю налаштування та управління базами даних. Проте не всі вони мають достатню навичок або часу на реалізацію серверної частини, яка відповідає за зв'язок із базою даних. Дана проблема є актуальною для розробників, які створюють невеликі сайти або веб-застосунки, які потребують лише базові CRUD операції з даними. У цьому разі витрати часу на реалізацію веб-серверу стають недоцільним.

Відповідно до цього, створення веб-системи, яка б давала: можливість управляти базами даних через веб-інтерфейс; зручне REST API для роботи з даними, може бути ефективним підходом до вирішення даної проблеми.

Сучасний ринок таких систем не є великим. Проаналізувавши популярні платформи для управління базами даних, можна виділити серед них такі основні риси:

- хмарна база даних;
- масштабування та резервні копії;
- підтримка базових CRUD операцій;
- можливість роботи з даними через SDK, REST API, GraphQL.

До недоліків даних систем можна віднести:

- відсутність інтерфейсу для управління базою даних;
- прив'язка до конкретної бази даних;
- відсутність інтеграції з будь-якою системою через використання SDK чи GraphQL;
- відсутність можливості створювати власні запити до баз даних.

Тож розробка нової веб-системи інструментів розробника реляційної бази

даних є актуальною потребою. Дана система має: забезпечувати високий рівень безпеки доступу до ресурсу за рахунок використання захищених протоколів передачі даних та використання JWT для автентифікації користувачів [1]; надавати: веб-інтерфейс для управління базою даних; REST API для інтеграції з будь-якою системою; можливість створення власних запитів.

Розроблена система матиме наступні функціональні можливості:

1. Створення та управління базами даних, таблицями, полями та даними через веб-інтерфейс.

2. Генерація токенів для доступу до баз даних за допомогою REST API.

3. Автоматичне створення REST API для CRUD операцій. Система автоматично створює API для кожної таблиці.

4. Створення власних запитів для баз даних. Користувач може створювати власні запити до таблиць.

Для роботи з веб-системою необхідно мати доступ до Інтернету та будь-який пристрій з браузером.

Для побудови веб-системи було обрано клієнт-серверну архітектуру [2] (рис. 1). Клієнтська частина розробляється мовою програмування JavaScript та фреймворку Angular 18 [3]. Для розробки серверної частини використовується мова програмування Java та фреймворк Spring Boot [4]. Для збереження основних даних використовується СУБД InterSystems IRIS, для збереження JWT використовується Redis Cache [5].

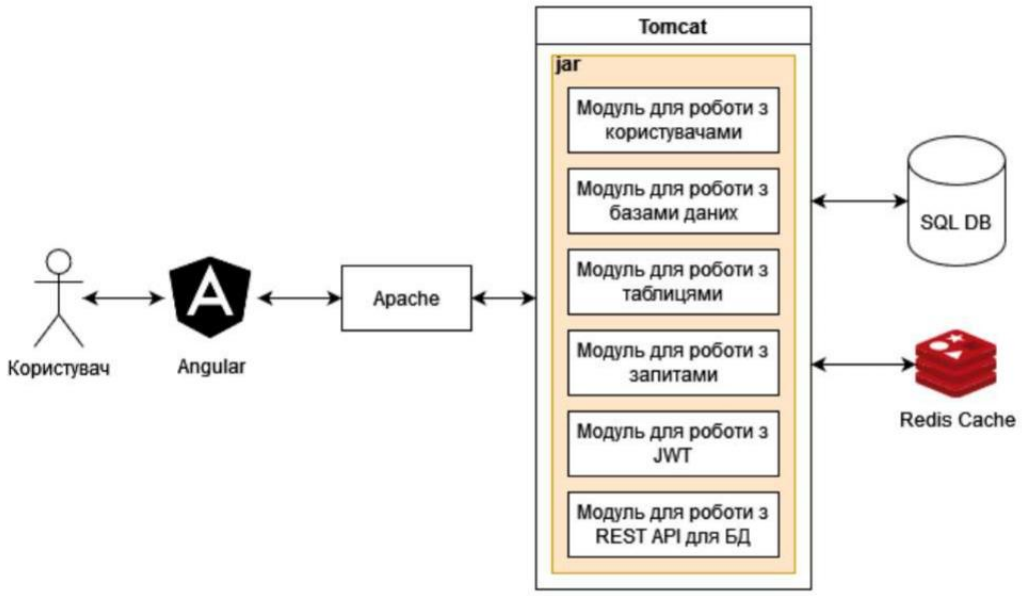

Рис. 1. **Архітектура системи**

Запропонована система «Інструменти розробника реляційної бази даних» дозволить швидко та надійно управляти базами даних та даними.

Список використаних джерел:

1. Spilca L. Spring Security in Action. 1st ed. Manning Publications Co. LLC. 2020. 560 p.
2. Посвістак В. С., Демківська Т. І. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення. Інформаційні технології в науці, виробництві та підприємництві: зб. наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. Київ: Освіта України; ФОП Маслаков. 2020. С. 78-81.
3. Fain. Y. Angular Development with TypeScript. 2nd ed. Manning Publications Co. LLC. 2018. 560 p.
4. Walls C. Spring Boot in Action. 1st ed. Manning Publications Co. LLC. 2016. 264 p.
5. Carlson J. L. Redis in Action. 1st ed. Manning Publications Co. LLC. 2013. 320 p.

ДОДАТОК Б

ІНСТРУМЕНТИ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Апробація наукових досліджень

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТР-31мп_24

Аркушів 6

Київ – 2024

ЗМІСТ

UNIVERSUM
№ 14 / 2024

Міщенко Андрій Андрійович
здобувач магістерського освітнього ступеня інституту атомної та теплової енергетики
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Україна

Науковий керівник: Михайлова Ірина Юріївна
канд. техн. наук, доцент кафедри цифрових технологій в енергетиці
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Україна

ІНСТРУМЕНТИ РОЗРОБНИКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

АНОТАЦІЯ. **Популяризація Інтернет технологій призвела до збільшення кількості різного виду веб-сайтів та веб-систем. Разом з цим для забезпечення динамічності веб-сервісів з'явилася потреба в швидкому створенні та управлінні базами даних. Проте існує проблема, що не всі розробники мають достатню кількість часу та досвіду для виконання цих завдань: розгортання окремої серверної частини та створення бази даних. Особливо це притаманно невеликим проєктам, які мають потребу лише в базових CRUD операціях і не потребують занадто складної обробки даних. У статті досліджуються популярні веб-системи, що покликані вирішувати дану проблему: надають користувачу можливість створювати бази даних, надаючи при цьому API для роботи з ними для заміни серверної частини. Аналізуються можливості таких систем, розглядаються їх переваги та недоліки. Також у статті розглядається доцільність створення нового програмного рішення, яке б виправило існуючі недоліки популярних рішень та дозволило користувачам спростити процес створення та керування базами даних.**

КЛЮЧОВІ СЛОВА: ***реляційна бази даних, інструменти, серверна частина, веб-система.***

З кожним роком Інтернет стає все дедалі популярнішим. Це призводить до збільшення кількості нових користувачів; створенню нових блогів; відкриттю сайтів чи магазинів різноманітними бізнесами. Популяризацію Інтернет технологій підтверджують останні дослідження, які показують, що кількість веб-сервісів росте з кожним роком, а їх використання наздоганяє використання звичайних застосунків [1].

Разом з цим з'являється попит на створення веб-сайтів як для бізнесу, так і для звичайних користувачів. Проте це завдання не є простим, оскільки для забезпечення динамічності на таких ресурсах потрібне використання веб-сервісів, які взаємодіють з базами даних.

Тож під час розробки таких веб-систем розробники стикаються з проблемами створення та керування базами даних. Не всі вони здатні вирішити дані проблеми по різних причинах: відсутність достатнього рівня знань та досвіду; відсутність часу на реалізацію окремої серверної частини, яка б взаємодіяла з базами даних.

Найбільш поширеною дана проблема є актуальною серед розробників, які створюють невеликі проєкти, наприклад, до таких веб-сервісів можна віднести блоги, особисті та навчальні проєкти, прості CRM системи та багато іншого, що потребує лише базових CRUD операцій з даними. У цьому випадку витрачати час на створення та розгортання серверної частини є недоцільним.

Тому для вирішення даних проблем слід використовувати спеціальні веб-системи, що надають можливість створювати та управляти базами даних, а також надають API для роботи з даними у власних системах.

На сьогодні існує не велика кількість популярних рішень, які б насправді надавали потрібний функціонал. До них можна віднести: Firebase, Supabase, Hasura, Strapi. Ці системи об'єднує одна мета: полегшити процес створення різного типу баз даних та надати зручне API для роботи з інформацією в сховищах даних. Проте кожна система не є універсальною, покликана вирішувати різні проблеми зберігання даних, та має як ряд переваг, так і недоліків. Тож розглянемо кожну з них більш детальніше.

Firebase – це універсальна платформа для розробки як веб-, так і мобільних застосунків, що підтримується компанією Google. Вона надає підтримку всіх необхідних інструментів для створення серверної частини: бази даних, хостинг, автентифікацію, аналітику та багато іншого [2].

До переваг даної платформи можна віднести:

- база даних знаходиться в хмарі;
- підтримка резервних копій та масштабованості;
- проста інтеграція з зовнішніми системами через Google SDK та REST API.

Єдиним головним недоліком цієї платформи є відсутність підтримки реляційних баз даних, з якого випливають інші:

- дані тут зберігаються в колекціях у вигляді документів, через що зв'язки між даними потрібно забезпечувати в ручну;
- відсутня підтримка SQL запитів, особливо з JOIN та WHERE.

Supabase – це нова платформа, яка має відкритий вихідний код. Має підтримку одної реляційної бази даних PostgreSQL, що дозволяє використовувати її для проєктів, які потребують підтримки SQL запитів та

реляційних моделей даних [3].

Перевагами даної платформи є:

- підтримка реляційної бази даних PostgreSQL;
- зручний веб-інтерфейс для керуваннями базами даних, таблицями, стовпцями та даними.
- можливість створювати зовнішні ключі через інтерфейс;
- підтримка тригерів, процедури;
- підтримка REST API для роботи з базовими CRUD операціями;
- вбудований механізм перевірки доступу до бази даних.

Головними недоліками ж платформи є:

- усі таблиці за замовчуванням створюються з головним ключем з назвою id, що обмежує певним чином свободу користувача;
- зв'язування таблиць відбувається через головний ключ id, та не підтримує використання інших унікальних індексів для цього;
- відсутність можливості додати обмеження унікальності на стовпчик;
- відсутність можливості створювати та зберігати власні SQL запити для виклику їх через REST API.

Hasura – це інструмент з відкритим вихідним кодом, що дозволяє автоматично створювати GraphQL API для існуючих баз даних як реляційних PostgreSQL, так і нереляційних MongoDB [4].

Перевагами платформи є:

- автоматична генерація GraphQL API для існуючих баз даних, що дозволяє виконувати базові CRUD та складні SQL запити за допомогою створеного прикладного інтерфейсу;
- підтримка зовнішніх зв'язків між таблицями та можливість роботи з ними через GraphQL без обмежень;
- інтеграція з зовнішніми системами за допомогою GraphQL API.
- можливість гнучкого налаштування правил доступ до бази даних різним користувачам;
- підтримка тригерів.

Недоліками платформи є:

- відсутність користувацького веб-інтерфейсу для створення та управліннями базами даних, таблицями та іншим, передбачається, що користувач сам має необхідні навички роботи з SQL для управління ними;
- відсутність підтримки REST API;
- створення GraphQL вимагає встановлення спеціального програмного забезпечення на власний сервіс.

Strapi – це платформа для управління контентом, що має підтримку API для роботи з даними. Підтримує роботу з PostgreSQL, MySQL, SQLite, MariaDB. Для роботи з нею необхідно завантажувати та розгортати на

власному сервері [5].

Перевагами є:

- зручний інтерфейс для створення та налаштування структури таблиць;
- підтримка REST та GraphQL API;
- підтримка реляційних зв'язків;
- підтримка користувацьких плагінів;
- можливість адаптувати панель адміністратора під власні потреби.

До недоліків можна віднести:

- необхідність розгортання платформи на власному сервері;
- розроблений мовою програмування Node.js, тому вимагає базового її розуміння для налаштування та розгортання;
- відсутність інструментів для створення та виконання специфічних запитів;
- хоча плагіни і є перевагою та надають розширений функціонал, проте вони можуть стати і проблемою, оскільки деякі функції завжди вимагають встановлювати додаткові плагіни та налаштовувати їх.

Проаналізувавши переваги та недоліки чотирьох платформ для вирішення проблеми створення та управління базами даних з підтримкою API для взаємодії з даними, можна побачити навіть популярні рішення не є ідеальними та не мають усіх ключових та потрібних можливостей для кінцевих користувачів. Так, наприклад, Supabase та Hasura є одними з найкращих у свій ніші, проте перший не надає повної гнучкості користувачу в створенні таблиць, а другий не має веб-інтерфейсу для управління базами даних та потребує встановлення програмного забезпечення на сервер для генерації API. Усе це в кінцевому результаті може змусити користувача прилаштовуватися під це, або ж власноруч почати створювати серверну частину з необхідним функціоналом.

Тому є актуальним завдання створення інструментів розробника реляційної бази даних, які б у свою чергу вирішували базову проблему з створення та налаштування баз даних з підтримкою адаптивного та універсального API. А також надавали б користувачу зручний інтерфейс для роботи з базами даних та не потребували б від клієнта встановлювати собі на сервер програм чи власноруч розгортати платформу. Розроблювана система має забезпечити хмарне рішення надаючи можливості для роботи з нею через зручні веб-інтерфейс та API.

API для роботи з даними має бути розроблене з використанням REST архітектури, оскільки вона є найбільш поширеною, усі розробники добре орієнтуються в ній, тому інтеграція з користувацькими системами не викличе ніяких проблем.

Для забезпечення захисту даних необхідно використати стандарт створення токенів – JSON Web Token. Оскільки легко інтегрується з REST API, та дозволить встановлювати системі чи має той чи інший користувач доступ до запитуваних даних.

Інструменти розробника реляційної бази даних матимуть забезпечить кінцевого користувача наступним функціоналом:

1. Створення та керування базами даних, а також їх таблицями, стовпцями через зручний інтерфейс.

2. Автоматичну генерацію REST API для взаємодії з даними в таблицях у базах даних.

3. Забезпечення перевірки прав доступу до виконання запитів для взаємодії з даними через REST API за допомогою JSON Web Token, які є унікальними для кожної бази даних.

4. Можливість створювати, зберігати та виконувати власні SQL запити для баз даних через REST API.

Таким чином інструменти розробника реляційної бази даних забезпечать користувача всім необхідним функціоналом для керування базою даних за допомогою зручного інтерфейсу через браузер, а також REST API з JWT автентифікацією для безпечної взаємодії з даними в таблицях, що в кінцевому результаті дозволять швидко та надійно виконувати базові операції з сховищами даних без витрати великої кількості часу.

Список використаних джерел:

[1] 2022 App vs. Website Trend Report. AMPLITUDE LAB : веб-сайт. URL: https://amplitude.com/guides/2022-app-vs-website-report (дата звернення: 03.11.2024).
[2] Firebase Realtime Database. Firebase Documentation: веб-сайт. URL: https://firebase.google.com/docs/database (дата звернення: 03.11.2024).
[3] Supabase Documentation. SUPA : веб-сайт. URL: https://supabase.com/docs (дата звернення 03.11.2024).
[4] Hasura DDN Documentation. Hasura Docs: веб-сайт. URL https://hasura.io/docs/3.0/index/ (дата звернення: 03.11.2024).
[5] Strapi 5 Documentation. Strapi Docs : веб-сайт. URL: https://docs.strapi.io/ (дата звернення 03.11.2024).