

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

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри

О.В. Коваль
(підпис) (ініціали, прізвище)

“ ” 2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напрямку підготовки 121 Інженерія програмного забезпечення

на тему: Система аналізу показників рівня міжнародного співробітництва

Виконав: студент 4 курсу, групи ТВ-61

Петровський Олександр Григорович

(прізвище, ім'я, по батькові)

(підпис)

Керівник старший викладач Дацюк Оксана Антонівна

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Консультант

(назва розділу)

(вчені ступінь та звання, прізвище, ініціали)

(підпис)

Рецензент

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент

(підпис)

Київ – 2020 року

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрямок підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.В. Коваль

(підпис)

” ____ ” _____ 2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Петровському Олександру Григоровичу

(прізвище, ім'я, по батькові)

1. Тема роботи ”Система аналізу показників рівня міжнародного співробітництва”

керівник роботи старший викладач Дацюк Оксана Антонівна

(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ” ____ ” ____ 202__р. № ____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи персональний комп'ютер на операційній системі Windows 10, мова програмування C#, мова програмування TypeScript.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати програмне забезпечення, аналогічне до розроблюваного, виявити його переваги та недоліки, розробити власний програмний продукт та користувацький інтерфейс.

5. Перелік ілюстративного матеріалу користувацькі інтерфейси аналогічних програм, схеми архітектури проекту, схеми бази даних, приклади роботи програми.

6. Публікації: _____

7. Дата видачі завдання ” ____ ” _____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/П	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент

(підпис)

Петровський О.Г.

(прізвище та ініціали,)

Керівник роботи

(підпис)

Дацюк О. А.

(прізвище та ініціали,)

АНОТАЦІЯ

Робота містить 55 сторінок, 24 малюнки, 3 додатки та 12 посилань.

Мета роботи – створити програмне забезпечення для швидкого аналізу показників міжнародного співробітництва у науково-технічній сфері.

У ході роботи було розглянуто Microsoft Excel та SQL Server Management Studio як аналогічні програмні продукти, виявлено їхні переваги та недоліки, розглянуто різні структури багатовимірних баз даних, розроблено програмний продукт для аналізу показників міжнародного співробітництва у науково-технічній сфері.

Ключові слова: multidimensional databases, data warehouses, Web API, Angular, куб, переріз.

ANNOTATION

The work contains 55 pages, 24 pictures, 3 appendices and 12 references.

The purpose of the work is to create software for quick analysis of indicators of international cooperation in science and technology.

In the course of work Microsoft Excel and SQL Server Management Studio were considered as similar software products, their advantages and disadvantages were revealed, various structures of multidimensional databases were considered, a software product for analysis of indicators of international cooperation in science and technology was developed.

Keywords: multidimensional databases, data warehouses, Web API, Angular, cube, cross section.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів	8
Вступ.....	10
1. Задача розробки системи для аналізу показників рівня міжнародного співробітництва у науково-технічній сфері.....	11
1.1 Призначення.....	11
1.2 Підсистеми	12
1.2.1 Сховище даних.....	12
1.2.2 Багатовимірна база даних	12
1.2.3 Web API	12
1.2.4 Користувацький інтерфейс.....	13
1.2.5 Взаємодія підсистем	13
2. Опис існуючих програмних рішень для роботи з багатовимірними базами даних	14
2.1 Структура багатовимірної бази даних.....	14
2.2 Огляд існуючих систем.....	15
2.3 Висновки до розділу.....	17
3. Засоби розробки.....	18
3.1 Завантаження даних у систему	19
3.2 Сховище даних.....	20
3.3 Багатовимірна база даних та OLAP	20
3.4 База даних користувачів	21
3.5 Веб-сервер	23
3.6 Користувацький інтерфейс.....	25
3.7 Бібліотека ADOMD.NET	26
3.8 Середовища розробки	27
3.9 Висновки до розділу.....	28
4. Опис програмної реалізації	30
4.1 Сховище даних.....	30
4.2 Додавання даних до сховища	32
4.3 Багатовимірна база даних	34
4.4 Веб-сервер	36
4.5 Інтерфейс користувача.....	40

4.6 Алгоритм роботи системи	43
4.7 Висновки.....	44
5. Робота користувача з програмою	45
5.1 Запуск системи.....	45
5.2 Інтерфейс користувача.....	45
5.3 Висновки.....	52
Висновки	53
Список використаних джерел	54
Додаток А	56
Специфікація.....	56
Додаток Б.....	58
Текст програмного модулю	58
Додаток В	66
Опис програмного модулю.....	66
Анотація.....	67
Зміст	68
1. Загальні відомості.....	69
1.1 Опис логічної структури.....	69
1.2 Вхідні та вихідні дані	69
1.3 Використані технічні засоби	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MDB – Multidimensional Database
OLAP – Online Analytical Processing
JSON – JavaScript Object Notation
API – Application programming interface
IDE – Integrated Development Environment
SSMS – SQL Server Management Studio
SQL – Structured Query Language
ASP.NET – Active Server Pages для .NET
REST – Representational state transfer
HTTP – Hypertext Transfer Protocol
MDX – Multidimensional Expression
HTML – Hypertext Markup Language
CSS -- Cascading Style Sheets
JS – JavaScript
TS – TypeScript
DW – Data Warehouse
MVS – Microsoft Visual Studio
WS – WebStorm
ORM – Object Relating Mapping
EF – Entity Framework
DAL – Data Access Layer
BLL – Business Login Layer
PL – Presentation Layer
CRUD – Create, Read, Update, Delete
URL – Uniform Resource Locator

SPA – Single Page Application

ETL – Extract, Transform, Load

UI – User Interface

XML – Extensible Markup Language

JWT – JSON Web Token

ВСТУП

Сьогодні більшість вищих навчальних закладів (ВНЗ) намагається співпрацювати з ВНЗ інших країн для отримання нового досвіду у викладанні, обміні студентами та аспірантами, міжнародних розробок, що збільшить популярність ВНЗ за кордоном. Виходячи з цього закладам необхідні системи для обліку інформації про заходи, студентів, викладачів і т.д. під час співробітництва та подальшого її аналізу.

OLAP (з англ. Online Analytical Processing, аналітична обробка у реальному часі) [6] – це технологія, на основі якій побудовано багато програмного забезпечення для бізнес-аналітики. Суть її полягає у тому, що нові результати отримуються за лічені секунди (“online”) за рахунок попередньо вирахованих даних і аналітик не вимушений довго очікувати виконання нового запиту.

Основною концепцією OLAP-системи є багатовимірний OLAP-куб. Виміри у кубу є даними з предметної області; зазвичай ці дані представлені у лінгвістичному вигляді, рідше – у числовому. Прикладами вимірів з різних предметних областей можуть бути: календар (дати, місяці, роки), університети (кафедри, факультети, університети), географія (країна, штат, місто) і т.д.

Інструменти OLAP дозволяють користувачам інтерактивно аналізувати багатовимірні дані з різних точок зору. OLAP складається з трьох основних аналітичних операцій: консолідація (згортання), деталізація та переріз. Консолідація передбачає агрегацію даних, які можуть бути накопичені та обчислені в одному або кількох вимірах. Навпаки, деталізація - це техніка, яка дозволяє користувачам орієнтуватися в деталях. Тобто користувач може переходити по ієрархії до більш детальних рівнів. Переріз - це особливість, завдяки якій користувачі можуть виймати (нарізати) певний набір даних з куба OLAP і переглядати фрагменти з різних точок зору. Ці точки зору іноді називають розмірностями (наприклад, перегляд міжнародних конференцій, проведених лише у певній країні у певний рік).

1. ЗАДАЧА РОЗРОБКИ СИСТЕМИ ДЛЯ АНАЛІЗУ ПОКАЗНИКІВ РІВНЯ МІЖНАРОДНОГО СПІВРОБІТНИЦТВА У НАУКОВО-ТЕХНІЧНІЙ СФЕРІ

Метою роботи є розробка системи для аналізу показників рівня міжнародного співробітництва в науково-технічній сфері. Аналіз даних є важливою частиною у багатьох інформаційних системах. Це допомагає за допомогою цифр явно побачити динаміку розвитку системи, оцінити її результативність або популярність (в залежності від предметної області) та в деяких випадках робити прогнози.

1.1 Призначення

Дане програмне забезпечення призначене для аналітиків вищих навчальних закладів, які беруть участь у міжнародному співробітництві. Аналітик може швидко отримувати дані як заздалегідь визначених показників, так і зробити інший запит. Генерація запитів може відбуватися як у графічному режимі, обираючи куби та виміри, так і за допомогою MDX-запиту.

У зв'язку з вищеописаними вимогами на вхід до системи можуть надходити три види даних: назва визначеного показника, назва кубу та набір вимірів, для яких робиться запит та MDX-запит. Додатково для авторизації можуть знадобитися електронна пошта та пароль користувача від системи.

На виході до користувача завжди повертається таблиця з даними. В силу того, що на двовимірному екрані неможливо відобразити більше вимірів, ніж два, користувач не зможе отримати дані в перерізі більше, ніж з двох вимірів.

1.2 Підсистеми

1.2.1 Сховище даних

Для збереження даних, які потім треба аналізувати використовуються сховище даних [10]. Сховище даних – це реляційна база даних із зв'язками та відношеннями, проте вона має іншу структуру. Так як у звичайній нормалізованій базі даних під час запиту зазвичай потрібні дані з кількох таблиць, то в час їх виконання входить прив'язка записів із однієї таблиці до іншої. Для зменшення виконання запитів до аналітичної бази даних, або її обрахунку у випадку OLAP, сховища даних спеціально денормалізують.

Іншою особливістю структури сховища даних є те, що вона має два типи таблиць: таблиці-факти та таблиці-виміри. Перші зберігають інформацію, зазвичай числову, яка потім буде вираховуватися. Другі – побічну інформацію, по якій потім групуватимуться дані.

1.2.2 Багатовимірна база даних

Багатовимірні бази даних [8] беруть інформацію зі сховища даних. Основою багатовимірної бази даних є гіперкуби. Куби мають виміри, за якими потім робляться перерізи. Далі ці перерізи об'єднують, агрегуючи дані. Найважливішою особливістю багатовимірної БД є те, що всі дані агрегуються під час обробки куба, тобто потім все що робить аналітик – перерізає куб потрібними вимірами та отримує результат.

1.2.3 Web API

Система має мати веб-сервер у вигляді Web API [11], який буде проміжним пластом між базою даних та графічним інтерфейсом користувача. Сервер отримує дані від користувача у вигляді HTTP запитів та посилає відповіді також по протоколу HTTP, робить відповідні запити до бази даних, отримує дані з БД, перетворює дані у потрібний формат та повертає інтерфейсу. Також важливою функцією серверу є аутентифікація та авторизація для захисту даних. Архітектурно сервер не залежить та

нічого не знає про користувацький інтерфейс. Це дає можливість підлаштовувати інтерфейс або мати декілька користувацьких інтерфейсів, якщо цього вимагатиме розширення, не вдаючись до змін на сервері та не припиняючи його роботу.

1.2.4 Користувацький інтерфейс

Система має бути реалізована у вигляді SPA (Single-Page Application). Односторінкові додатки є набагато зручнішими для користувача, адже браузер не перезавантажує сторінку кожен раз під час якоїсь дії користувача. Інтерфейс має давати можливість користувачу виконувати певні запити виходячи з вказаних показників, мати можливість обрати куб, до якого буде виконуватись запит та обрати зі списку вимірів, ієрархій та рівнів об'єкти, за якими робитимуться агрегування кубу.

1.2.5 Взаємодія підсистем

Описані вище підсистеми утворюють вертикальну ієрархію і мають взаємодіяти тільки з однією, на один рівень вищої підсистемою, тобто користувацький інтерфейс взаємодіє лише з Web API, а Web API лише з багатовимірною базою даних. Зменшення залежності підсистем одна від одної є досить важливим, коли є висока вірогідність розширення системи та полегшує підтримку програмного забезпечення.

2. ОПИС ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДЛЯ РОБОТИ З БАГАТОВИМІРНИМИ БАЗАМИ ДАНИХ

2.1 Структура багатовимірної бази даних

Як було сказано у попередньому розділі, основною структурною одиницею багатовимірної БД є гіперкуби. Існують два способи структурувати багатовимірну БД:

1. Створити один куб, реалізувавши у ньому всі виміри та міри.
2. Створити багато кубів для представлення окремих логічних структур.

Системи, що мають невелику кількість мір, або які розробляються під спеціальне програмне забезпечення, що працює лише з одним кубом, розробляють першим способом. Проте у таких систем є кілька важливих недоліків. По-перше, єдиний куб у системі має перетинати всі виміри, проте не всі міри можуть перетинати ці виміри. На практиці це дає користувачу можливість робити перерізи по тим вимірам, які паралельні даній мірі і ніяк не впливають на результат. Тому ефективною системою такою БД буде лише за умови чіткого розуміння як цей куб був створений, що накладає поріг входу для програмного забезпечення. Псевдо перевага такої структури також розвіюється під час розширення системи. Так як усі міри та виміри знаходяться в одному кубі, при великій кількості мір, вони логічно групуються у каталоги для зручності. Пошук потрібної міри у списку каталогів абсолютно аналогічний пошуку кубу у списку кубів, проте маючи окремі куби, користувач зможе оперувати виключно тими вимірами, які доступні для саме цього кубу. Проте якщо ще є засоби для групування мір у каталогах, то для роботи з вимірами аналогічного функціоналу немає, тому крім цього користувач буде вимушений шукати потрібні йому міри у величезному списку, що також не сприятиме покращенню користувацького досвіду.

2.2 Огляд існуючих систем

Сьогодні існують готові рішення для роботи з багатовимірними базами даних. Наприклад, Pivot tables у Microsoft Office Excel можна налаштувати на роботу з багатовимірною базою даних. Користувач вводить дані підключення, отримує список кубів, обирає потрібний та може с ним працювати. Головною перевагою Excel є його зручний користувацький інтерфейс, який є зрозумілим та не вимагає додаткових знань про роботу багатовимірної бази даних (рисунок 2.1).

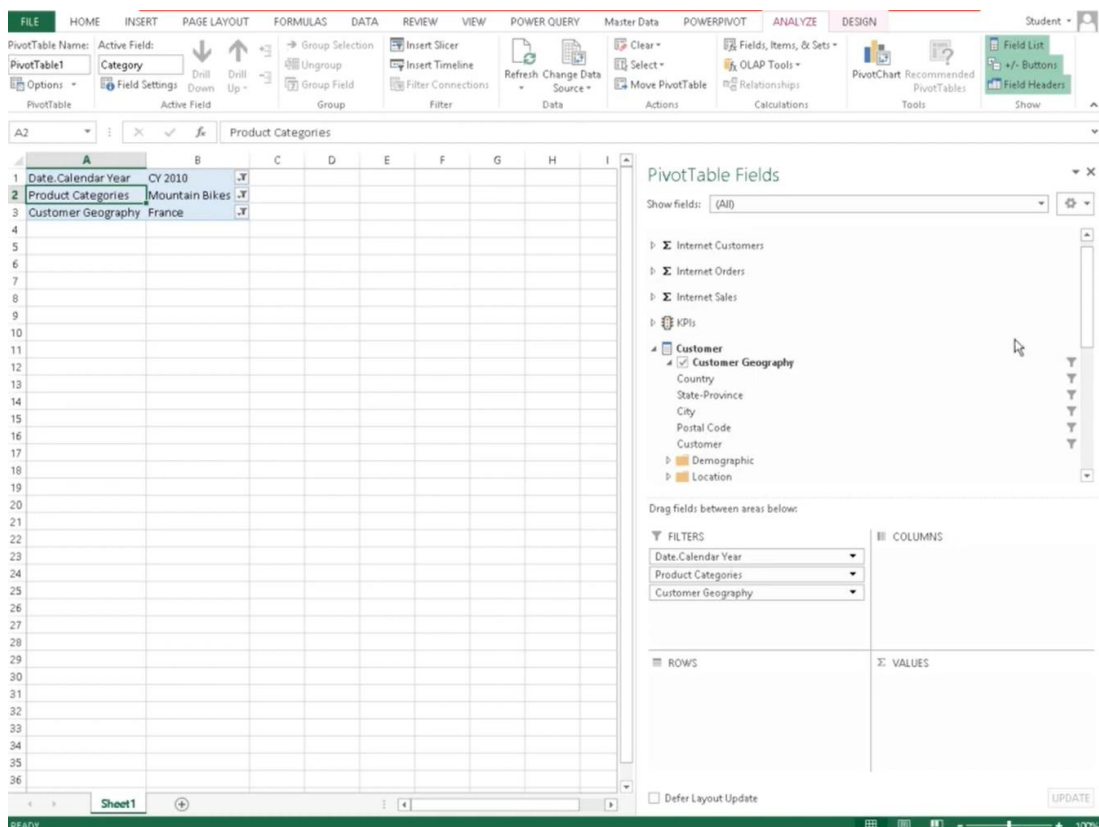


Рисунок 2.1 – Інтерфейс програми Excel

Проте це програмне забезпечення має ряд недоліків. По-перше, немає захисту даних: інша людина з доступом до файлу може отримати дані. По-друге, вибір кубу пропонується на етапі підключення до серверу, тому на його зміну піде багато дій, що не є зручним під час роботи в системі з багатьма кубами.

Іншим програмним забезпеченням від компанії Microsoft є SQL Server Management Studio (SSMS) [5]. Хоча дана програма має набагато ширші можливості для роботи з базами даних, в тому числі і з багатовимірними, вона більше призначена для розробки. Для роботи з багатовимірною базою даних треба підключитися до Analysis Services серверу ввівши користувача та пароль. Це покращує безпеку системи та дає адміністратору широкі можливості по керуванню користувачами, їхніми ролями та накладання обмежень або надання доступу до певних даних. З іншої сторони SSMS має не такий зручний графічний інтерфейс, порівняно з Excel, проте вона має більше можливостей (рисунок 2.2). Наприклад, додавання фільтрів до запитів по різним ієрархіям чи рівням, що дає можливість виконувати більш конкретні запити. SSMS дозволяє також перейти текстового редактору, щоб побачити який запит буде виконаний після того, як він був створений у графічному редакторі. Більше того, можна одразу перейти до написання власного MDX-запиту.

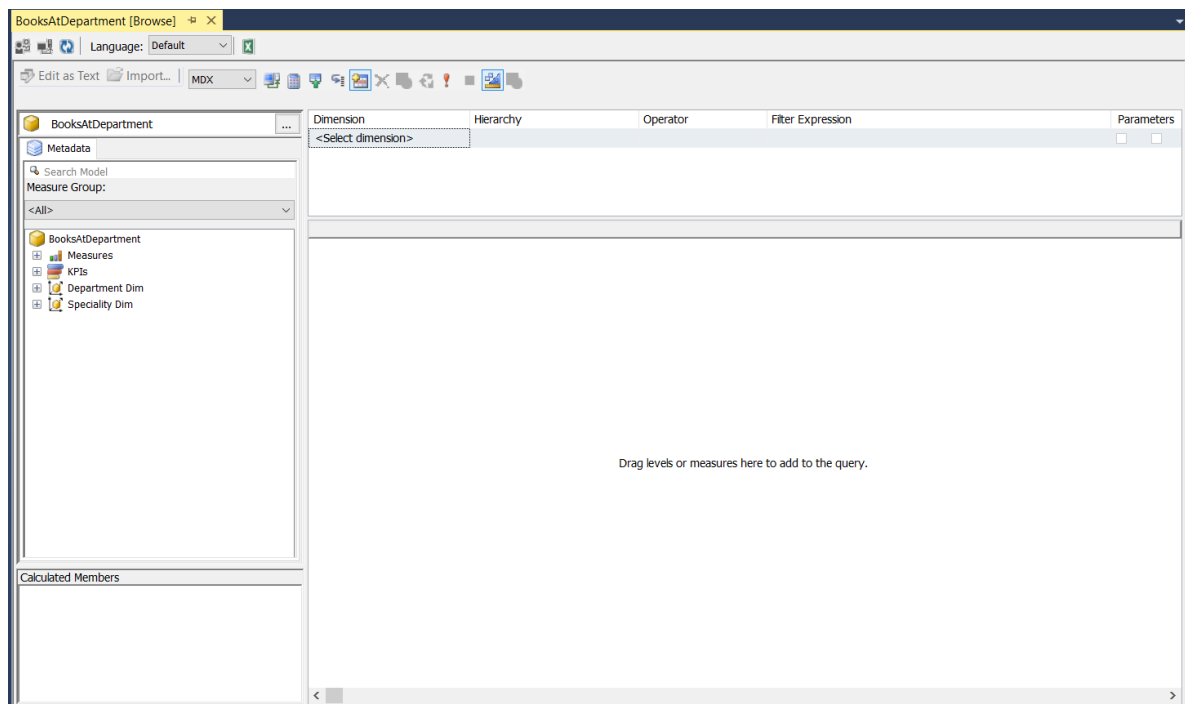


Рисунок 2.2 – Інтерфейс програми SSMS

Розроблена система вибирає в себе переваги вищеповисаних програмних продуктів та опускає недоліки, так як вона розроблена виключно для роботи з OLAP. Система дає можливість отримувати заздалегідь визначені показники Створений

візуальний конструктор для швидкої побудови запитів та реалізована можливість виконання MDX-запитів. Система має авторизацію, що дозволяє отримувати доступ до даних лише деяким користувачам, що дозволить встановлювати додаток на робочих комп'ютерах. Результати виконання запитів можна зберігати у вигляді файлу Excel та будувати лінійні графіки за цими результатами. Швидкий вибір кубів дозволить швидко та зручно працювати з обома структурами багатовимірної бази даних.

2.3 Висновки до розділу

У даному розділі було розглянуто два основні типи структуризації багатовимірної бази даних: з одним кубом та багатьма кубами. Встановлено їхні переваги та недоліки один перед одним.

В якості існуючих програмних продуктів, аналогічних до розробленого під час роботи, були розглянуті два програмних продукта від Microsoft. Microsoft Office Excel та SQL Server Management Studio. Наведено приклади інтерфейсу користувача в обох програмах та принципах роботи з багатовимірними базами даних. Основними перевагами першого є зручний інтерфейс користувача та побудова графіків, а перевагами другого – зручність у роботі з багатьма кубами та легкість виконання MDX-запитів. Описано, який функціонал ввібрала в себе розроблювана система.

3. ЗАСОБИ РОЗРОБКИ

Найважливішими аспектами під час створення програми були здатність до розширення та легкість у підтримці. Тому ідея була у розбитті монолітної архітектури додатку на рівні. В результаті з додатку можна виділити наступні логічно-виокремлені рівні:

- сховище даних,
- багатовимірна база даних,
- реляційна база даних з інформацією про користувачів,
- веб-сервер,
- користувацький інтерфейс.

Розбиття компонентів на частини полегшує як розробку, так і підтримку програмного забезпечення. Так як у монолітній архітектурі всі модулі знаходяться в одному місці і тісно пов'язані один з одним, то при внесенні змін до одного модуля, треба буде змінювати і залежні від цього модуля модулі. Схема взаємодії рівнів зображена на рисунку 3.1.

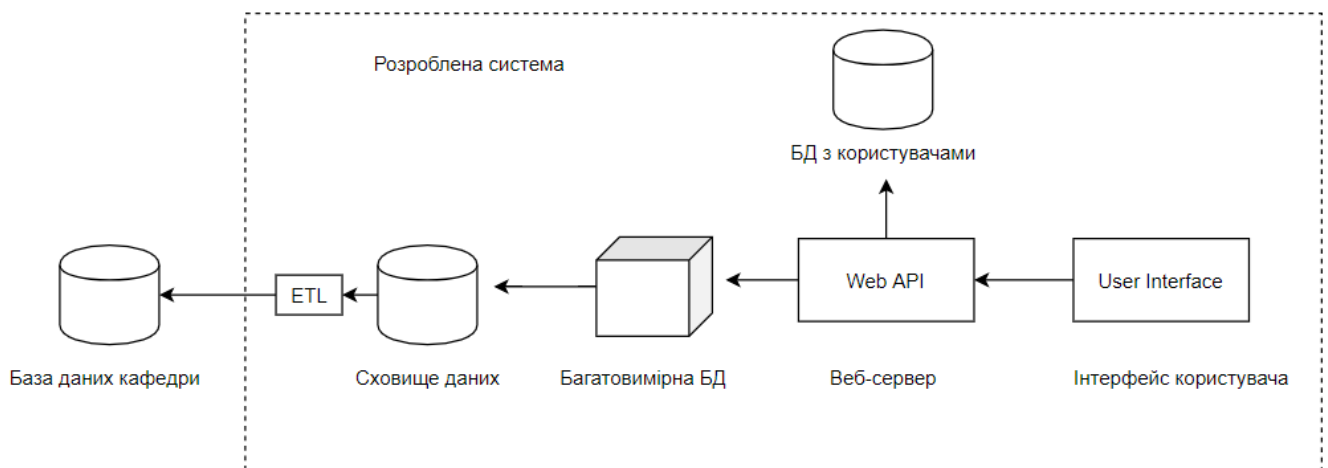


Рисунок 3.1 – Взаємодія рівнів додатку

Як можна бачити з рисунку крім веб-серверу усі модулю мають не більше 1 залежності. Дана архітектура також дозволяє розділити розробку додатку між різними командами.

3.1 Завантаження даних у систему

Дані, що зберігаються у сховищі, завантажуються з бази даних кафедри. Процес наповнення сховища називається ETL (з англ. extract, transform, load). Поняття ETL виникло в результаті появи безлічі корпоративних інформаційних систем, які необхідно інтегрувати один з одним з метою уніфікації та аналізу збережених в них даних. Реляційна модель представлення даних, що підходить для потреб транзакційних систем, виявилася неефективною для комплексної обробки і аналізу інформації. Пошук уніфікованого рішення привів до розвитку сховищ і вітрин даних – самостійних систем зберігання консолідованої інформації у вигляді вимірювань і показників, що вважається оптимальним для формування аналітичних запитів.

Прикладне призначення ETL полягає в тому, щоб організувати таку структуру даних за допомогою інтеграції різних інформаційних систем. Незалежно від особливостей побудови і функціонування ETL-система повинна забезпечувати виконання трьох основних етапів процесу ETL-процесу:

- вилучення даних з одного або декількох джерел і підготовка їх до перетворення (завантаження в проміжну область, перевірка даних на відповідність специфікаціям і можливість подальшого завантаження в СД)
- трансформація даних – перетворення форматів, агрегація і очищення
- завантаження даних – запис перетворених даних, включаючи інформацію про структуру їх подання (метадані) в необхідну систему зберігання (СД) або вітрину даних.

Таким чином, ETL-процес являє собою переміщення інформації (поток даних) від джерела до одержувача. Вимоги до організації потоку даних описує аналітик.

Тому ETL – це не тільки процес перенесення даних з однієї програми до іншої, а й інструмент підготовки даних до аналізу.

3.2 Сховище даних

Сховище даних (з англ. Data Warehouse), також відоме як корпоративне сховище даних (EDW), є системою, що використовується для звітності та аналізу даних, і вважається основною складовою бізнес-аналітики. DW – це центральні сховища даних з одного або декількох різних джерел. Вони зберігають поточні та історичні дані в одному місці, які використовуються для створення аналітичних звітів для працівників по всьому підприємству.

Для розробки був використаний SQL Server Data Engine, розроблений компанією Microsoft. SQL Server – це система управління реляційними базами даних. Як сервер бази даних, це програмний продукт з основною функцією зберігання та отримання даних, як цього вимагають інші програмні програми - який може працювати або на тому ж комп'ютері, або на іншому комп'ютері в мережі (включаючи Інтернет).

3.3 Багатовимірна база даних та OLAP

Онлайн-аналітична обробка (OLAP) – це багатовимірне, багатокористувацьке обчислювальне середовище клієнт-сервер для користувачів, яким потрібно аналізувати дані підприємства. Фінансові управління використовують OLAP для таких додатків, як бюджетування, калькуляція на основі діяльності (асигнування), аналіз фінансових результатів та фінансове моделювання. Відділи продажів використовують OLAP для аналізу та прогнозування продажів. Відділи маркетингу використовують OLAP для аналізу маркетингових досліджень, прогнозування продажів, аналізу акцій, аналізу клієнтів та сегментації ринку/клієнтів. Типові виробничі програми OLAP включають планування виробництва та аналіз дефектів.

Важливою для всіх цих програм є можливість надати менеджерам інформацію, необхідну їм для прийняття ефективних рішень щодо стратегічних напрямків організації. Успішна програма OLAP надає необхідну інформацію; тобто вона надає інформацію "вчасно" для ефективного прийняття рішень.

Надання такої інформації вимагає більш ніж базового рівня детальних даних. Інформація, що знаходиться вчасно, – це обчислені дані, які, як правило, відображають складні взаємозв'язки і часто розраховуються під час руху. Аналіз та моделювання складних взаємозв'язків практичні лише в тому випадку, якщо час реакції постійно короткий. Крім того, оскільки характер відносин даних може бути невідомий заздалегідь, модель даних повинна бути гнучкою. По-справжньому гнучка модель даних гарантує, що системи OLAP можуть відповідати на зміни ділових потреб, необхідних для ефективного прийняття рішень.

OLAP характеризується порівняно низьким обсягом транзакцій. Запити часто досить складні і передбачають агрегацію. Для систем OLAP час реакції – це міра ефективності. Бази даних OLAP зберігають агреговані дані у багатовимірних схемах (як правило зіркові схеми). Підхід OLAP використовується для аналізу багатовимірних даних з різних джерел та перспектив.

У системі багатовимірна база даних реалізована завдяки SQL Server Analysis Service. Це онлайн-аналітична обробка (OLAP) та інструмент пошуку даних у Microsoft SQL Server. SSAS використовується організацією як інструмент для аналізу та осмислення. Служба аналізу включає групу можливостей OLAP та обміну даними та має два вигляди – багатовимірний та табличний.

3.4 База даних користувачів

База даних користувачів – звичайна реляційна база даних, що зберігає інформацію про користувачів та їхні ролі. Створена Entity Framework Core та ASP.NET Core Identity [3] на SQL Server Data Engine.

Entity Framework Core (EF) – це легка, розширювана, відкрита версія та міжплатформна версія популярної технології доступу до даних Entity Framework.

EF Core може слугувати об'єктно-реляційним відображенням (ORM), дозволяючи розробникам .NET працювати з базою даних за допомогою .NET-об'єктів і виключати потребу в більшості кодів доступу до даних, який вони зазвичай потребують для запису. EF Core підтримує багато двигунів баз даних.

За допомогою EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, який представляє сеанс із базою даних, що дозволяє запитувати та зберігати дані.

Ви можете генерувати модель з існуючої бази даних, вручну кодувати моделі, щоб вона відповідала вашій базі даних, або використовувати EF Migrations для створення бази даних з вашої моделі, а потім розвивати її, коли ваша модель змінюється з часом.

ASP.NET Core Identity – це API, який підтримує функцію входу в інтерфейс користувача (UI), керує користувачами, паролями, даними профілю, ролями, претензіями, маркерами, підтвердженням електронної пошти тощо.

Користувачі можуть створити обліковий запис із інформацією про вхід, що зберігається в Identity, або вони можуть скористатися зовнішнім постачальником послуг. Підтримувані зовнішні постачальники послуг входу включають Facebook, Google, обліковий запис Microsoft та Twitter. Вихідний код Identity доступний на GitHub.

Ідентифікація, як правило, налаштована за допомогою бази даних SQL Server для зберігання імен користувачів, паролів та даних профілю. Альтернативно, може використовуватися інший стійкий магазин, наприклад, сховище таблиці Azure.

База даних, створена ASP.NET Core Identity має наступні таблиці:

- `AspNetUser` – зберігає інформацію про користувачів (ідентифікатор, ім'я, хеш паролю, електронну пошту і т.д.).
- `AspNetRole` – зберігає інформацію про ролі користувачів у системі (ідентифікатор та назву).
- `AspNetUserClaim` – відповідає таблиці зв'язків між користувачами та об'єктами claims.
- `AspNetUserToken` – зберігає інформацію про токени користувача.

- `AspNetUserLogin` – зберігає зв'язки користувачів з логінами через зовнішні сервіси
- `AspNetRoleClaim` – відповідає таблиці зв'язків між ролями та об'єктами claims
- `AspNetUserRole` – таблиця, що зберігає співставлення користувачів та ролей

Усі таблиці можуть бути розширені наслідуванням відповідних класів та додавання їх до контексту Identity.

Також ASP.NET Core Identity дозволяє легко контролювати різні обмеження, що стосуються входу в систему. По-перше, вимоги до паролю нових користувачів, такі як:

- Обов'язкова цифра
- Обов'язкова маленька літера
- Обов'язкова велика літера
- Обов'язковий неалфавітний символ
- Мінімальна довжина паролю
- Кількість унікальних символів

По-друге, можна встановити додаткові обмеження для користувачів. Наприклад, унікальна адреса електронної пошти, щоб користувач не міг зареєструвати у системі більше одного акаунту на поштову скриньку. Також можна самостійно визначити перелік символів, з яких має складатиметься ім'я користувача.

3.5 Веб-сервер

У якості веб-серверу використовувався ASP.NET Core Web API [2].

ASP.NET Core – це безкоштовний фреймворк із відкритим кодом, наступник ASP.NET, розроблений компанією Microsoft та спільнотою. ASP.NET – це модульний фреймворк, який працює як на повному .NET Framework на операційній системі

Windows, так і на крос-платформеному .NET Core. Однак ASP.NET Core версії 3 працює лише на .NET Core, втративши підтримку .NET Framework.

Фреймворк представляє собою повне переписування, об'єднує більш раннє відділення ASP.NET MVC та ASP.NET Web API в єдину модель програмування.

Не дивлячись на те, що це новий фреймворк, який побудований на новому веб-стеку, він володіє високим ступенем сумісності концепцій з ASP.NET. Додатки ASP.NET Core підтримують паралельне управління версіями, при якому різні додатки, працюючі на одному комп'ютері можуть орієнтуватися на різні версії ASP.NET Core, що було неможливо з попередніми версіями ASP.NET.

Web-API представляє можливість побудувати додатки ASP.NET, які спеціально закладаються для роботи в стилі REST (Representation State Transfer або "передача стану представлення"). REST-архітектура пропонує застосовувати наступні методи або типові запити HTTP для взаємодії з серверами:

- GET
- POST
- PUT
- DELETE

Зазвичай REST-стиль дуже зручний під час створення всякого роду Single Page Application, які не рідко використовують спеціальні JavaScript-фреймворки типу Angular, React або Vue.js. По суті Web API представляє собою веб-службу, до якої можуть звертатися інші додатки. Причому ці додатки можуть представляти будь-яку технологію та платформу – це можуть бути веб-додатки, мобільні або десктопні клієнти.

Веб-API сервера – це програмний інтерфейс, що складається з однієї або декількох публічно доступних кінцевих точок до визначеної системи повідомлень запит-відповідь, як правило, виражається в JSON або XML, яка піддається впливу через Інтернет – найчастіше за допомогою HTTP-основі веб-серверу.

Користувач може отримати доступ до ресурсів на сервері за допомогою уніфікованого ідентифікатору ресурсу (з англ. Uniform Resource Identifier або URI).

Для REST-серверу різні типи запитів від користувача відповідають різним CRUD операціям:

- CREATE (POST)
- READ (GET)
- UPDATE (PUT)
- DELETE (DELETE)

3.6 Користувацький інтерфейс

Інтерфейс користувача у системі представлений як односторінковий додаток (Single Page Application), створений за допомогою фреймворку Angular 9 [9]. Angular – один з найпопулярніших JavaScript фреймворків, який використовується сьогодні. Має підтримку величезної компанії (Google) та широке використання у корпоративному світі.

Angular був створений Google і спочатку був випущений у 2010 році як AngularJS – версія 1.x. Початковий випуск версії Angular 2.x відбувся 14 вересня 2016 року. Хоча початковий випуск попередньої альфа- та бета-версії існував у 2015 році, осінь 2016 року – час, коли Angular побачив реальну тягу у використанні.

Друга основна редакція спочатку називалася AngularJS 2 або 2.0, але згодом була ребрендована лише на "Angular" для випуску 2.0 та вище. Версія 5.x у бета-версії була опублікована саме в той час, коли це було написано. Якщо ви відчуваєте, що чогось не вистачає, це версія 3, тому що це було пропущено. Поки Angular був на версії 2.0, маршрутизатор Angular вже був у версії 3, тож щоб синхронізувати все, вони перейшли з 2.x у 4.x.

Angular має вже вбудовану реалізацію патерну MVC. Архітектура Model-View-Controller не тільки надає значення фреймворку під час створення додатка на стороні клієнта, але й встановлює основу для інших функцій, таких як прив'язка даних та сфери застосування.

За допомогою архітектури MVC можна ізолювати логіку програми від рівня інтерфейсу та підтримувати розділення проблем. Контролер отримує всі запити на

додаток і працює з моделлю, щоб підготувати будь-які дані, необхідні для перегляду. Перегляд використовує дані, підготовлені контролером, і відображає остаточну презентабельну відповідь.

Фреймворк Angular 9 використовує TypeScript – це мова програмування з відкритим вихідним кодом, розроблений і підтримуваний Microsoft. Це строгий синтаксичний розширений набір JavaScript, який додає в мову необов'язкову статичну типізацію. TypeScript призначений для розробки великих програм і транскompілюється в JavaScript. Оскільки TypeScript є розширеним набором JavaScript, існуючі програми JavaScript також є допустимими програмами TypeScript.

TypeScript може використовуватися для розробки додатків JavaScript як для виконання на стороні клієнта, так і на стороні сервера (як з Node.js або Deno). Є кілька варіантів, доступних для транскompіляції. Можна використовувати або перевірку за замовчуванням TypeScript, або можна викликати компілятор Babel для перетворення TypeScript в JavaScript.

3.7 Бібліотека ADOMD.NET

Для зв'язку серверу, реалізованому на платформі ASP.NET Core із багатовимірною базою даних використовувалася офіційна бібліотека від Microsoft ADOMD.NET [1]. Вона являє собою розширення для базової технології доступу до даних ADO.NET.

ADO.NET (ActiveX Data Object для .NET) – технологія, що надає доступ і керування даними, що зберігаються в базі даних або інших джерелах (Microsoft SQL Server, Microsoft Access, Microsoft Excel, Microsoft Outlook, Microsoft Exchange, Oracle, OLE DB, ODBC , XML, текстові файли), заснованих на платформі .NET Framework і входить до складу .NET Framework 2.0, являє собою набір бібліотек. На відміну від технології ADO, яка була в основному призначена для тісно пов'язаних клієнт-серверних систем, ADO.NET більше націлена на автономну роботу за допомогою об'єктів DataSet. Об'єкти DataSet представляють локальні копії взаємопов'язаних таблиць даних, кожна з яких містить набір рядків і стовпців. Об'єкти

DataSet дозволяють викликаючій збірці (на зразок веб-сторінки або програми, що виконується на настільному комп'ютері) отримувати доступ до вмісту DataSet, змінювати його, не вимагаючи підключення до джерела даних, і відправляти назад блоки змінених даних для обробки за допомогою відповідного адаптера даних. Але, мабуть, саме фундаментальна відмінність між класичною ADO і ADO.NET полягає в тому, що ADO.NET є керованою кодовою бібліотекою, і, отже, підпорядковується тим же правилам, що і будь-яка керована бібліотека. Типи, складові ADO.NET, використовують протокол управління пам'яттю CLR, належать до тієї ж системи типів (класи, інтерфейси, перерахування, структури і делегати), і доступ до них можливий за допомогою будь-якої мови .NET.

MD у назві буквально означає Multi Dimensional, для чого і використовується дана бібліотека. Вона включає у себе класи, які представляють об'єкти багатовимірних баз:

- Cube
- Dimension
- Hierarchy
- Level
- Member

Також бібліотека включає класи, аналогічні класам ADO.NET для створення підключення, створення команд та їх виконання.

3.8 Середовища розробки

Для створення веб-серверу та багатовимірної бази даних з OLAP використовувалася IDE Microsoft Visual Studio 2019 Community [7]. Visual Studio – це не лише одна з найстаріших IDE навколо, яка була запущена в кінці 90-х, але також є однією з найпопулярніших. І з поважних причин: це зрілий, потужний і універсальний IDE, який використовується по всьому світу для створення веб-сайтів, програмного забезпечення для настільних ПК, мобільних додатків та ігор на ряді популярних мов

програмування, таких як C # і C ++. Іншим дуже важливим фактором є постійна підтримка від Microsoft, який оновлює її та додає нові функції.

Для роботи із реляційними базами даних використовувалася Microsoft SQL Server Management Studio (SSMS) – це інтегрована середовище для управління інфраструктурою SQL Server. Він надає користувацький інтерфейс та групу інструментів із багатими редакторами сценаріїв, які взаємодіють із SQL Server.

SSMS надає інструменти для налаштування, управління та адміністрування екземплярів Microsoft SQL Server, а також об'єднує цілий ряд інструментів графічного та візуального дизайну та багатих редакторів сценаріїв для спрощення роботи з SQL Server. Комбіновані функції SSMS надходять від Enterprise Manager, аналізатора запитів та диспетчера аналізу, а також функції, що містяться в попередніх випусках SQL Server. Він підтримує більшість адміністративних завдань SQL Server та підтримує єдине інтегроване середовище для управління та створення авторів бази даних SQL Server.

Для розробки інтерфейсу користувача та роботі з Angular використовувалася IDE від компанії JetBrains – WebStorm. WebStorm використовується для роботи з веб-додатками. Крім HTML, CSS та JS, вона підтримує роботу з TypeScript, який використовується фреймворком Angular. Іншими перевагами є підтримка великої кількості плагінів, які можуть розширити функціонал та вбудований термінал, який теж є корисним під час роботи, адже Angular\CLI використовується для швидкого створення структурних елементів проекту – компонентів та сервісів.

3.9 Висновки до розділу

У даному розділі було наведено перелік інструментів, з допомогою яких створювалася система аналізу показників рівня міжнародного співробітництва у науково-технічній сфері. Розроблений додаток був розділений на п'ять різних модулів: сховище даних, багатовимірна БД, веб-сервер, БД користувачів та клієнтський інтерфейс. Збереження даних та робота з ними реалізована через SQL Server. Data Engine для сховища даних та БД користувачів та Analysis Services для

багатовимірної БД. Веб-сервер був реалізований на ASP.NET Core як Web API. Клієнтський інтерфейс створений за допомогою фреймворку Angular. У якості СКБД була обрана SSMS, IDE для розробки серверу на інтерфейсу користувача MVS 2019 та WebStorm відповідно. Для роботи з багатовимірною базою даних на сервері використовувалася бібліотека ADOMD.NET.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Реалізація даної системи складалася з декількох етапів:

1. Розробити структуру сховища даних виходячи із отриманих показників.
2. Описати алгоритм наповнення сховища із бази даних кафедри.
3. Розробити куби для багатовимірної бази даних.
4. Розробити веб-сервер для реалізації потрібного функціоналу.
5. Створити інтерфейс користувача.

4.1 Сховище даних

Сховище даних реалізоване на SQL Server Data Engine. У зв'язку з призначенням сховища даних, дані в ній не нормалізовані. Іншою важливою відмінністю є те, що таблиці поділені на два типи: таблиці-виміри, які зберігають деталізовану інформацію (найчастіше ці дані у строковому вигляді), та таблиці-факти, в який зберігається інформація, яка потім буде агрегована (найчастіше числова).

Іншою особливістю сховища даних є його структура, яку ще часто називають зіркою або star (рисунок 4.1).

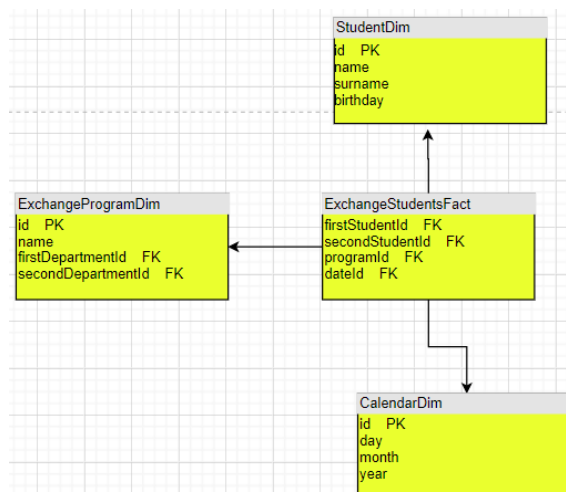


Рисунок 4.1 – Приклад структури зірка

Суть її полягає у тому, що у центрі так званої зірки знаходиться таблиця-факт, а від неї відходять таблиці-виміри, які розширюють та потім слугуватимуть для групування даних.

Ускладнивши структуру сховища, коли одні виміри залежать від інших, виходить структуру “сніжинка”. Приклад структури сніжинки можна побачити на рисунку 4.2.

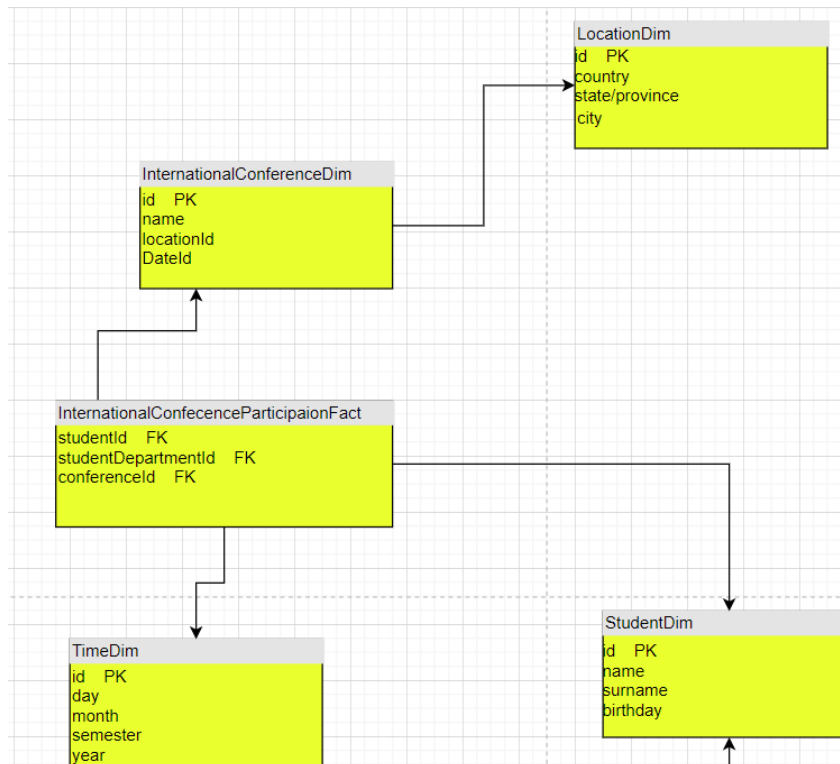


Рисунок 4.2 – Приклад структури сніжинки

Аналогічним чином реалізовані інші факти та виміри. Факт залежать від багатьох вимірів, а від виміру залежить декілька фактів, що не дає можливість візуально представити лише частини системи, адже кількість зв’язків у всій системі досить висока.

Прикладом денормалізації таблиць може виступити таблиця, яка представляє календарний вимір, який зберігає дати (рисунок 4.3). Очевидно, що усі ці дані можна отримати лише з одного поля, яке відображає дату у звичайному для SQL форматі – Date. Проте, щоб зменшити час виконання запиту та додати можливість групування з подальшою агрегацією (схлопування куба) багатовимірній базі потрібні поля з

однаковими значеннями: роки, місяці і т.д. Тому ми маємо зберігати однотипну інформацію, розплачуючись за швидкодію OLAP.

Для тестування та відлажування системи до таблиць були додані кілька тестових записів (рисунок 4.3) у кожену таблицю.

	Id	Date	Month	Semester	Year
1	20192	NULL	NULL	2	2019
2	20201	NULL	NULL	1	2020
3	20202	NULL	NULL	2	2020
4	19900315	1990-03-15	March	2	1990
5	19920628	1992-06-28	June	NULL	1992
6	19940203	1994-02-04	February	2	1994
7	20000203	2000-02-03	February	2	2000
8	20180321	2018-03-21	March	2	2018
9	20180901	2018-09-01	September	1	2018
10	20181001	2018-10-01	October	1	2018
11	20200901	2020-09-01	September	1	2020

Рисунок 4.3 – Структура таблиці CalendarDim з тестовими даними

Подібним чином заповнені інші таблиці у сховищі даних для тестування працездатності багатовимірної БД, побудованій поверх сховища.

4.2 Додавання даних до сховища

Процес додавання даних до сховища називається ETL (extract, transform, load) та складається з трьох етапів: вилучення даних із зовнішнього джерела, перетворювання їх до потрібного вигляду та завантаження даних у сховище. По факту, ETL виконує роль адаптера між базою даних кафедри та сховищем даних. Варто відмітити, що дані у сховищі дуже рідко видаляються або змінюються. У більшості випадків під час процесу наповнення даними знаходяться лише нові дані, та додаються у сховище. На рисунку 4.4 зображено зміну концептуальної моделі даних під час їхнього додавання у сховище.

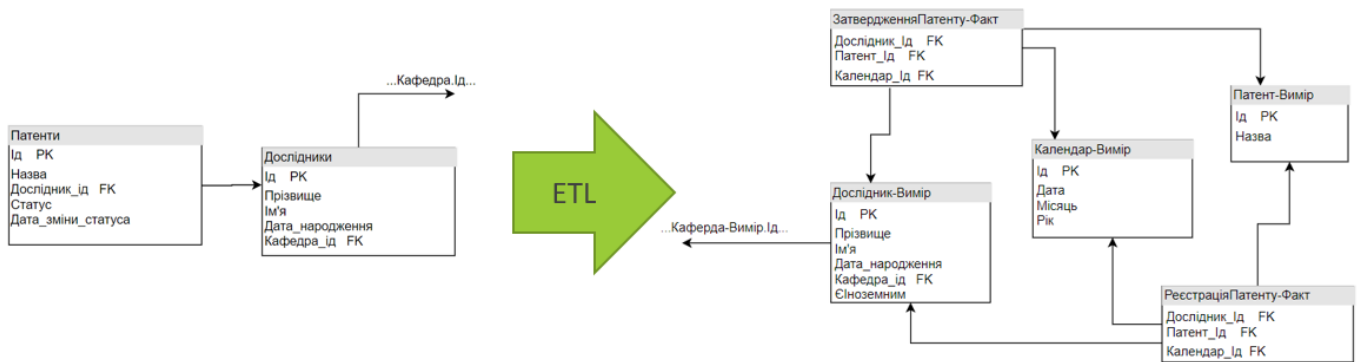


Рисунок 4.4 – Зміна концептуальної моделі під час ETL

Так як процес додавання даних у сховище при великій кількості даних досить тривалий та вимагає подальшого перерахунку мір у кубах, що також займає велику кількість часу, цей процес виконують раз в певний проміжок часу, зазвичай перед тим, коли необхідні актуальні дані. Тому повний процес додавання даних в систему включає крім роботи з базою даних кафедри та сховищем даних ще й роботу з багатовимірною базою даних.

Алгоритм наповнення сховища даними (рисунок 4.5) реалізований у вигляді набору SQL-виразів, які дістають дані із відповідних таблиць бази даних, строка підключення якої подається на вхід від користувача, та приводиться до потрібного вигляду для сховища даних. Після додавання усіх даних аналітик може перерахувати куби.

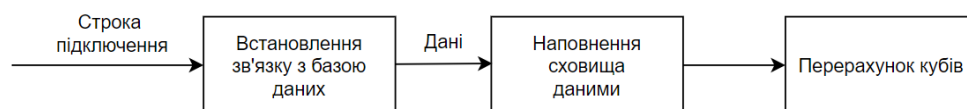


Рисунок 4.5 – Алгоритм додавання даних у систему

Після перерахунку кубів система надаватиме можливість аналізувати найактуальніші дані. Алгоритм передбачає, що база даних, що є джерелом даних, матиме визначену концептуальну модель.

4.3 Багатовимірна база даних

Багатовимірна база даних використовує як джерело даних вищеописане сховище даних. Реалізована вона на SQL Server Analysis Service та створена через MS Visual Studio проект типу Analysis Services Multidimensional and Data Mining Project. Під час створення багатовимірної БД першим чином вказується джерело даних та так звані Data Source View, де зберігаються метадані про структуру сховища даних, які використовуються під час створення кубів або вимірів.

Як структура багатовимірної бази даних була використана схема з багатьма кубами, так як вибір куба для користувача із списку кубів не відрізнятиметься від вибору каталогу для пошуку потрібної міри, але це забезпечить користувачу безпеку від безрезультатного перерізу по виміру, який немає перетину з потрібною мірою. Під час створення кубу завдяки метаданим вказуються перелік таблиць із сховища даних, які утворюють відповідну зірку або сніжинку. Після цього куб залишається прив'язаним до цих таблиць і додання нових даних вимагатиме лише перерахунок куба.

Приклад створеного кубу зображено на рисунку 4.6.

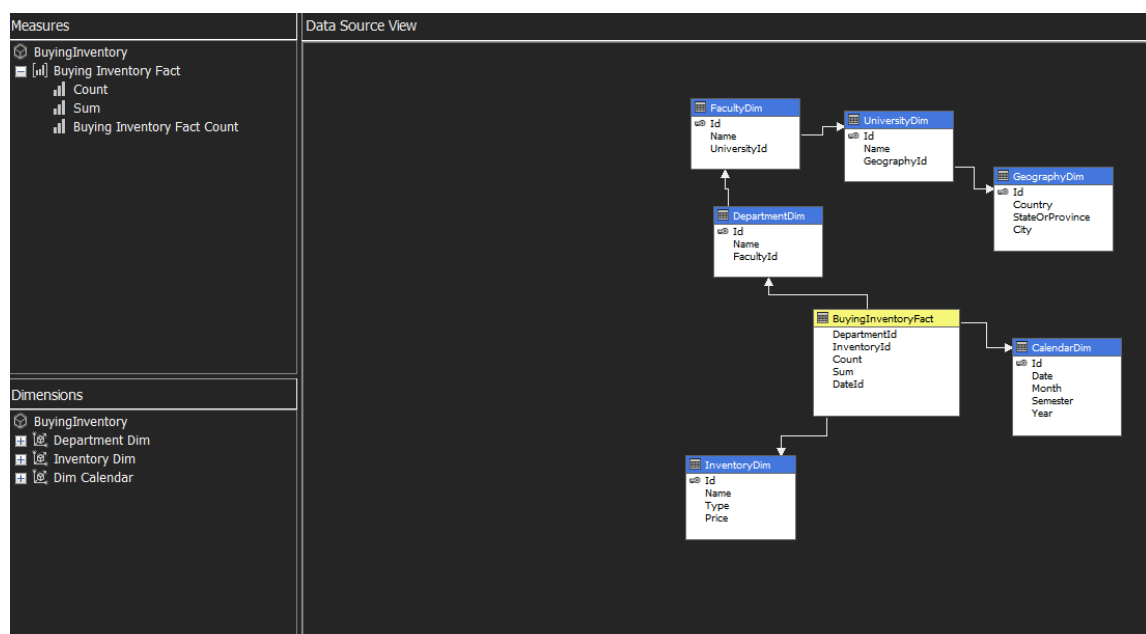


Рисунок 4.6 – Створений куб у багатовимірній базі даних

Як видно з рисунку 4.6, куб має 6 вимірів. Список вимірів, які куб перетинає самостійно знаходяться у віконці Dimensions. Проте як сказано вище, можна отримати доступ до інших вимірів, які перетинаються з його рідними. По вимірам можна робити перерізи або схлопування куба.

Список мір, які представляє куб знаходяться у віконці Measures. Можна побачити, що доступні кількість власне закупівель, суми закупівель та кількість закуплених товарів.

Функції агрегації, які будуть виконуватись над мірами під час схлопування куба (наприклад, сума або кількість записів), обираються під час створення та обрахунку кубу, що накладає певні обмеження на користувача.

Аналогічним чином реалізовані інші куби та виміри у багатовимірній базі даних.

Доступ до даних у багатовимірній базі даних реалізується завдяки мові MDX (з англ. Multidimensional Expression). Приклад MDX-запиту з результатом зображено на рисунку 4.7.

```

SELECT
    NON EMPTY
    [Inventory Dim].[Type].[Type]
    ON COLUMNS,
    NON EMPTY
    [Department Dim].[Department].[Department]
    ON ROWS
FROM
    [BuyingInventory]
WHERE
    [Measures].[Sum];
  
```

	Book	Computer
AES	1100	(null)
APEPS	2000	(null)
ASOIU	(null)	4000
AUTS	(null)	8000
English	(null)	9000
ITF	600	(null)
KTK	(null)	2100

Рисунок 4.7 – Запит до описаного вище кубу з результатом

Аналогічними запитами розраховуються статичні показники, важливі для даної предметної області. Ці запити відправляються з серверу, який гарантує безпеку завдяки контролю авторизації та аутентифікації користувачів та виконує проміжну логіку.

4.4 Веб-сервер

Веб-сервер реалізований на платформі ASP.NET Core.

Сервер має трьохрівневу архітектуру, яка дозволяє розмежувати логіку та відповідальність модулів, зменшити між ними залежність.

Найнижчий рівень – рівень доступу к даним (Data Access Layer, DAL). Цей рівень представляє собою бібліотеку класів, які реалізують фундаментальні операції для роботи з даними.

Всі ці операції загорнуті у клас `CubeDataGateway`, що є аналогом патерну `Table Data Gateway`. Суть створення цього класу поглядала у створенні абстракції між клієнтами, що будуть використовувати даний функціонал та власне доступом до даних (відкриття за закриття підключення до бази даних, генерація запиту та отримання результатів). В результаті клієнтський код взаємодіє лише через інтерфейс, знаючи які дії може виконати та який результат отримає. Такий підхід полегшує тестування програми та подальшу підтримку, адже додавання нового функціоналу у клієнтському коді не потребуватиме зміни у рівні доступу до даних.

На цьому рівні працює `Entity Framework Core`. Він у парі з `ASP.NET Core Identity` відповідають за генерацію таблиць у базі даних, що відповідають за вхід до системи: `Users`, `Roles`, `UserRoles`, `UserClaims`, `RoleClaims`, `UserLogins`, `UserTokens`. За що відповідають кожна з цих таблиць описано у попередньому розділі. Створення таблиць відбувається за технологією `Entity Framework` у `Code First`. Вона передбачає те, що таблиці створюються на основі описаних моделей, які зареєстровані в контексті. Реалізація EF разом з `ASP.NET Core Identity` відбувається завдяки наслідуванню контексту від `IdentityDbContext`, який сам є наслідником `DbContext`,

необхідному EF. Вищеописані бібліотеки відподівають за роботу з користувачами у системі. Для роботи з кубами на цьому ж рівні, в середині CubeDataGateway, використовувалася бібліотека ADOMD.NET. Вона є розширенням ADO.NET та передбачає роботу з багатовимірними базами даних, а не реляційними. Вона дає можливість створювати підключення до серверу бази даних, створювати запити, виконувати та отримувати результати.

CubeDataGateway реалізує інтерфейс ICubeDataGateway, який використовується Dependency Injection (DI) для додавання об'єкту туди, де він потрібен. У ASP.NET Core механізм DI встановлений та працює у щойно створеному додатку. Дана функція дуже зручна, але перекладає створення об'єктів на фреймворк та зменшує залежності між об'єктами. CubeDataGateway має методи для виконання статичних запитів, отримання назв всіх кубів, отримання детальної інформації про конкретний куб (назву, міри, виміри, ієрархії, рівні), виконання користувацького запиту.

Наступним рівнем є рівень бізнес логіки (Business Logic Layer, BLL). Іноді такий рівень називають Services Layer. На цьому рівні реалізуються сервіси, які інкапсують основну логіку додатку, використовуючи рівень доступу до даних. CubeService реалізує ICubeService, що також дозволяє використовувати сервіс у клієнтському коді через механізм Dependency Injection. Так як бібліотека ADOMD.NET крім звичайних даних дістає також велику кількість метаданих на цьому рівні реалізований мапер, який із результату запиту вибирає лише ті дані, які потрібно переслати користувачу. Результат запиту, готовий до повернення приводиться до типу QueryResultDTO, який має інформацію для коректного відображення результату користувачу. Дана операція також зменшує навантаження на мережу, через яку ці дані транслюватимуться. Рівень бізнес логіки також є бібліотекою класів і нічого не знає про клієнтський код.

Найвищим рівнем на сервері є презентаційний рівень (Presentation Layer, PL), на якому і розташована Web API. Цей рівень є головним. По-перше, тут знаходить конфігураційний файл, що зберігає критичну для роботи серверу інформацію, таку як строки підключення як до бази даних з користувачами, так і до багатовимірної бази

даних, та ключ, завдяки якому шифруються та дешифруються токени користувачів. По-друге, на цьому рівні розвернутий механізм DI, який створює об'єкти класів усього серверу. Таким чином усі попередньо описані рівні серверу пов'язуються в логічне ціле. По-третє, вся взаємодія серверу як окремої незалежної підсистеми відбувається тільки через Web API. Клієнти, що звертаються до сервера, нічого не знають про існування та реалізацію рівнів бізнес логіки та доступу до даних. Діаграма класів веб-серверу зображена на рисунку 4.8.

Також на цьому рівні працює ASP.NET Core Identity. Тому тут відбувається її підключення, встановлення етапу перевірки користувачів у конвеєр обробки запитів. Крім цього, тут відбувається налаштування додаткових опцій для Identity, таких як обмеження на пароль або обмеження на унікальність електронної адреси.

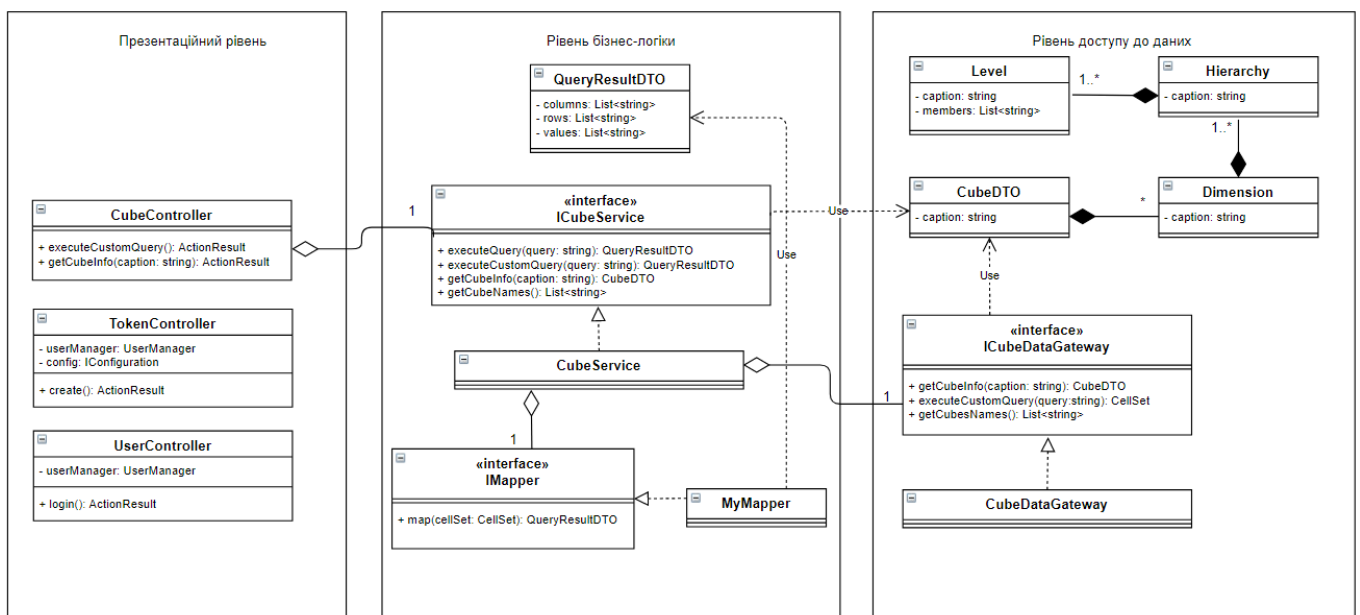


Рисунок 4.8 – Діаграма класів веб-серверу

Для опрацювання помилок, які могли виникнути на сервері передбачено або непередбачено у конвеєр обробки запитів вбудований проміжний шар для обробки виключних ситуацій. Таке рішення дозволяє не робити перевірки на правильний потік роботи, адже цей шар спрацює для будь-якого контроллера та кінцевої точки. Іншою перевагою такого підходу, крім прибирання дублювання коду є те, що можна

відправити користувачу зрозумілу інформацію про проблему, причину її виникнення та залогувати виникнення помилки.

Web API має власний стандартний маршрутизатор. Щоб потрапити на відповідний контролер, потрібно послати запит на `api/[contoller]`. `api/` на початку шляху показує, що запит буде до Web API. Таке іменування шляху відбувається за замовчуванням, його можна змінити, але так прийнято іменувати, щоб не плутати запити до Web API та ASP.NET Core MVC, які можуть бути в одному проєкті. Замість параметру `controller` підставиться назва контролеру з великої літери та додасться `Controller`. Наприклад, після запиту на `api/cube/` фреймворк шукатиме контроллер `CubeController`. У контролері за кожний запит відповідає метод. За тип запиту та додатковий шлях відповідають атрибути `Http`. Таким, чином, у тому самому контролері `CubeController`, метод, описаний з атрибутом `HttpGet("{id}")` викликатиметься за запити `api/cube/{id}`. Параметр `id` буде параметром методу.

Web API, отримавши запит від користувача на контролер, викликає потрібний метод сервісу з рівня бізнес логіки, який в свою чергу працює з рівнем доступу до даних. Важливо додати, що рівень презентації не залежить напряду від рівня доступу до даних.

На рівні презентації також реалізовані генерація JWT (JSON Web Token) [4], який є основою аутентифікації та авторизації користувачів. Генерація відбувається завдяки пакету `System.IdentityModel.Tokens.Jwt`. Токен складається з трьох частин:

1. Заголовок (Header) – зберігає інформацію про тип шифрування, яким був зашифрований токен та тип токenu.
2. Набір полей (Payload) – зберігає набір полей ключ-значення, проте форматом JWT існують декілька зарезервованих імен, таких як `iss`, `aud`, `exp` і т.д.
3. Сигратуру (Signature) – генерується методами шифрування.

Перші дві частини шифруються алгоритмом `Base64Url`. Для генерації сигнатури беруться перші дві зашифровані частини, секретний ключ, та шифрується алгоритмом, вказаним у заголовку.

У згенерованому токени частини відокремлюються крапкою. Під час генерації токени до набору полів даються наступні значення:

- Ідентифікатор користувача
- Ім'я користувача
- Час, починаючи з якого валідний токен
- Час, до якого валідний токен
- Ролі, до яких належить користувач

Таким чином, шахрай не зможе дешифрувати токен, адже не матиме ключового слова, яке зберігається у конфігураційному файлі на сервері `appsettings.json`. Контролери, які надають доступ до даних, мають захист від неавторизованих користувачів у вигляді атрибуту `Authorize` (рисунок 4.9).

```
[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "User")]
1 reference
public class CubesController : ControllerBase
{
```

Рисунок 4.9 – Захист від неавторизованих користувачів

Отримавши запит на такий контроллер, фреймворк намагається отримати токен з хедеру запиту і дешифрувати його. Якщо токен відсутній або невалідний, користувач отримує відповідь с кодом 401 та не отримує дані.

4.5 Інтерфейс користувача

Інтерфейс користувача реалізований за допомогою фреймворку Angular 9. Головним структурним елементом у Angular є компонент, який складається з 3 файлів: файл-темплейт (зберігає HTML розмітку), файл із стилями (CSS) та файл із класом компоненту, який реалізує його логіку. В розробленому додатку реалізовані наступні компоненти:

- `app-component` (компонент за замовчуванням)

- `indexes-component` (відображає список показників)
- `index-component` (відображає показник у вигляді таблиці)
- `navigation-component` (відображає навігаційне меню)
- `login-component` (відображає сторінку входу)
- `create-query` (візуальний конструктор запиту)
- `text-query` (текстовий конструктор запиту)

`index-component` реалізований як дочірній компонент для компонентів, що відображають візуальний конструктор та текстовий запит. Він має поле, що відповідає за зберігання MDX-запиту у текстовій формі та обробник події зміни цього поля. Дане поле є унаслідкованим від батьківського компонента. Різні батьки в свою чергу мають свої поля для зберігання запиту, які пов'язані з наслідником. Працюючи у власному просторі, батьківський компонент може змінювати своє поле із запитом, що автоматично змінить поле із запитом у наслідника. Далі спрацьовує оброблювач і викликає метод серверу для отримання даних із результатами запиту. Компонент `text-query` має просту форму і кнопку `Execute`. Натиснення кнопки заповнює значення поля із запитом і спричиняє описані вище дії. Схожим чином влаштований інший предок – компонент `create-query`. Натиснення кнопки `Execute` змінює значення поля із запитом. Головна різниця в тому, що для текстового запиту, сам запит вже готовий та написаний користувачем, а у графічному редакторі його треба попередньо побудувати. За це відповідає метод `buildQuery()`. У шаблоні цього компонента присутня таблиця, яка відображає структуру майбутнього запиту та структура виділеного кубу. Користувач може додавати елементи кубу (міри, ієрархії, рівні і т.д.) у структуру запиту. Далі білдер підставляє дані з цією таблиці у шаблон запиту, генерує запит та змінює поле із запитом. Подальші дії виконуються аналогічно до попереднього прикладу.

Для переходу між компонентами та збереженням історії у браузері реалізований роутер, який відображає в залежності від URL потрібний компонент.

Для взаємодії з сервером через конфігураційний файл фреймворку налаштований проксі, який по шаблону адреси перенаправляє запит на сервер додатку. Це має свої переваги. По-перше, адреса серверу не знаходиться в коді, а

винесена в окремий файл. Це зменшую дублювання в коді, що знижує шанс на помилку та у разі зміни адреси серверу, змінити її доведеться лише в одному місці.

Для збереження токена на стороні клієнту використовується `localStorage`. У разі успішного логіну токен зберігається на стороні клієнту, що є досить зручним, адже не потребує від користувача постійного вводу своїх даних. З точки зору безпеки може здатися, що це рішення є недосконалим, проте під час генерації токена на сервері у нього записується дата, до котрої він валідний. Тому через деякий час користувач все ж матиме оновити токен, чого не зможе зробити шахрай.

Реалізований перехоплювач запитів на стороні клієнту. Перехоплювач реалізований стандартними засобами Angular – імплементацією інтерфейсу `HttpInterceptor`. Перехоплювач обробляє кожен запит, який посилається з клієнту та до кожного запиту додає токен, якщо він записаний у `localStorage`. Перехоплювач також перевіряє усі відповіді від сервера. У разі отримання відповіді із кодом 401, яка говорить про те, що користувач неавторизований, він перенаправляє користувача на сторінку входу. Таким чином, роботу з авторизацією можна зобразити схемою, зображеною на рисунку 4.10.

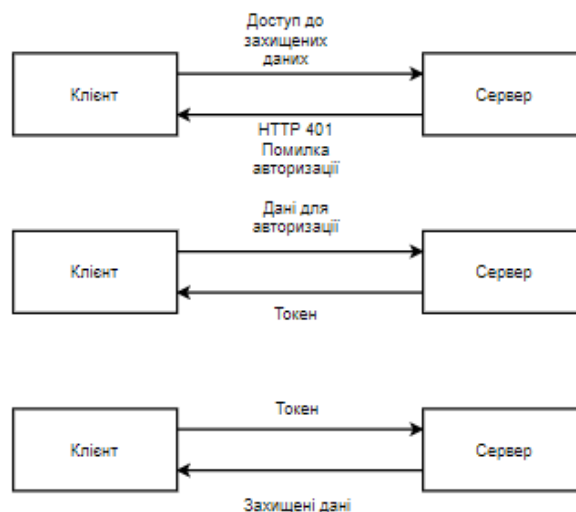


Рисунок 4.10 – Схема доступу до захищених даних

Текст будь-якої іншої помилки з серверу, наприклад, невірно побудований запит, помилка під час виконання запиту, перехід за неіснуючою адресою на сервері, буде відображений у впливаючому діалоговому вікні на сторінці у браузері.

4.6 Алгоритм роботи системи

Алгоритм роботи аналітика з системою можна представити у вигляді діаграми послідовностей, зображену на рисунку 4.10.

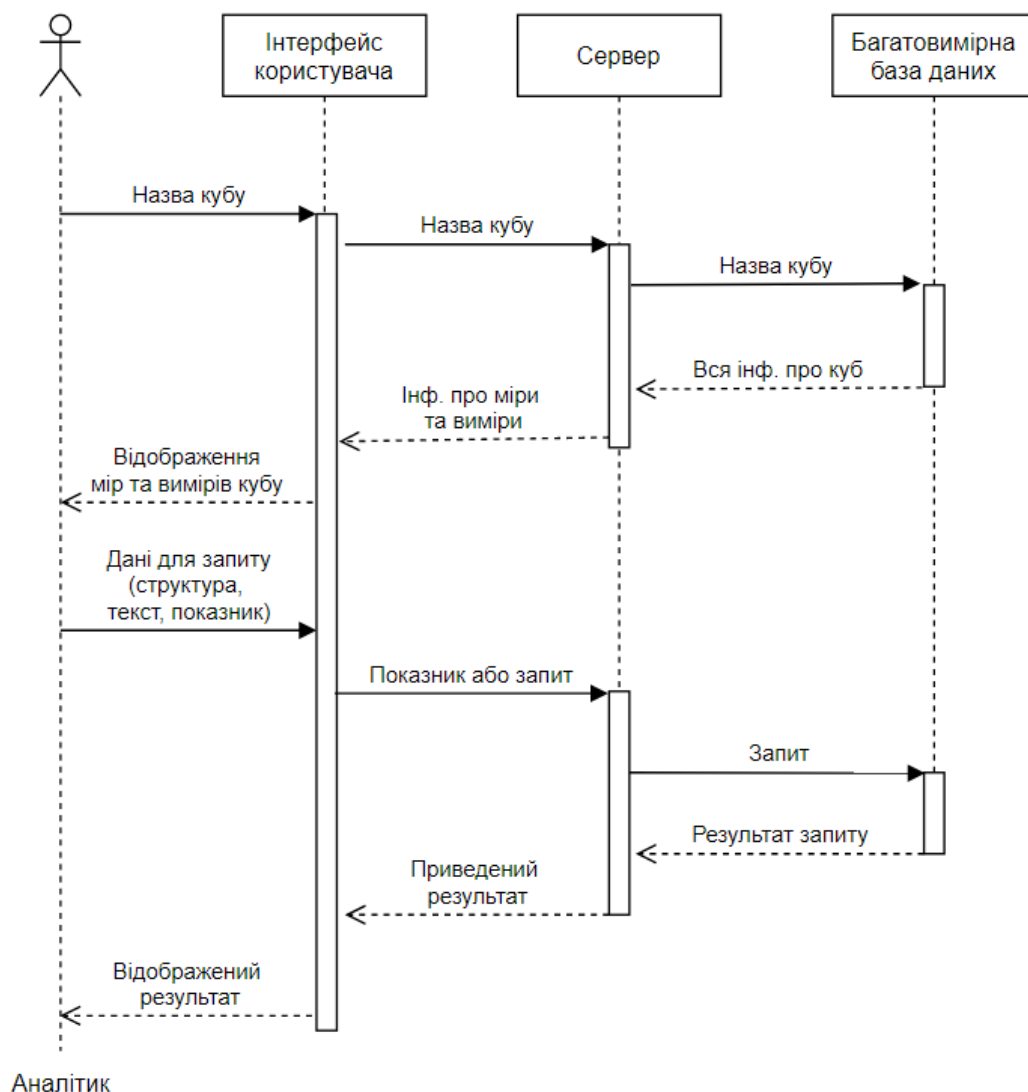


Рисунок 4.10 – Діаграма послідовностей роботи аналітика з системою

Як видно з діаграми, спочатку користувач обирає куб із списку кубів, після чого Інтерфейс користувача посилає запит на сервер, який з бази даних отримує всю інформацію про даний куб. Проте так як користувачу для запитів потрібні лише міри та виміри, він повертає лише цю інформацію. Інтерфейс користувача відображає отримані дані. Далі аналітик будує свій власний запит у конструкторі (або MDX-запит), інтерфейс з конструктора будує MDX-запит, який після цього посилається на сервер. Сервер виконує запит до багатовимірної БД. Отриманий результат він приводить до потрібного для відображення вигляду та повертає інтерфейсу, який в свою чергу відображає його користувачу.

У разі спроби аналітика отримати показник, його назва посилається з інтерфейсу користувача на сервер, а вже на сервері він перетворюється в MDX-запит та виконується, решта алгоритму аналогічна описаному вище.

4.7 Висновки

В даному розділі було описано програмну реалізацію системи аналізу показників міжнародного співробітництва у науково-технічній сфері. Сховище даних реалізовано на SQL Server Data Engine, має 2 типи таблиць: таблиці-факти та таблиці-виміри. Багатовимірна база даних створена на SQL Server Analysis Services, побудована над сховищем даних, має структуру сніжинки. Таблиці факти представляють куби, в яких мірами є поля таблиці. Таблиці-виміри представляють виміри, у яких розташовані куби. Сервер має трьохрівневу архітектуру: рівень доступу до даних, рівень бізнес логіки та презентаційний рівень; реалізує авторизацію. Інтерфейс користувача реалізований на Angular у якості SPA.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ

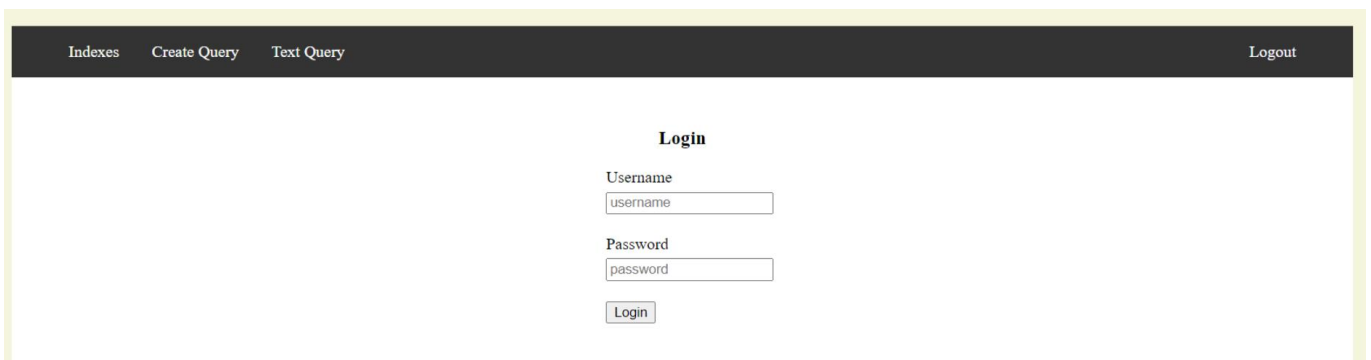
5.1 Запуск системи

Для запуску розробленої програми необхідно:

1. Запустити SQL Server Configuration Manager та запустити SQL Server зі сховищем даних та базою даних користувачів.
2. Там же запустити SQL Server Analysis Services.
3. Запустити сервер через MS Visual Studio.
4. Запустити клієнтський додаток через WebStorm або командою `ng serve` з кореня проекту.

5.2 Інтерфейс користувача

При завантаженні клієнтської частини користувачу доступне навігаційне меню. За замовчуванням користувача з домену пересилає на компонент із показниками, але якщо користувач неавторизований, то з серверу повернеться відповідь 401, через що перехоплювач перенаправить користувача на сторінку входу в систему (рисунок 5.1).



The screenshot shows a web application interface. At the top, there is a dark navigation bar with the following links: 'Indexes', 'Create Query', 'Text Query', and 'Logout'. The main body of the page is white and features a 'Login' section. This section includes a title 'Login', followed by a 'Username' label and an input field containing the text 'username'. Below this is a 'Password' label and an input field containing the text 'password'. At the bottom of the login section is a 'Login' button.

Рисунок 5.1 – Сторінка входу в систему

Потрапити на цю сторінку можна також натиснувши кнопку Login у правій частині навігаційного меню або перейти за адресою <http://localhost:4200/login>.

Після вводу імені користувача та паролю та натиснувши кнопку Login під формами вводу, у разі успішного входу користувач перенаправиться на сторінку з показниками (рисунк 5.2).

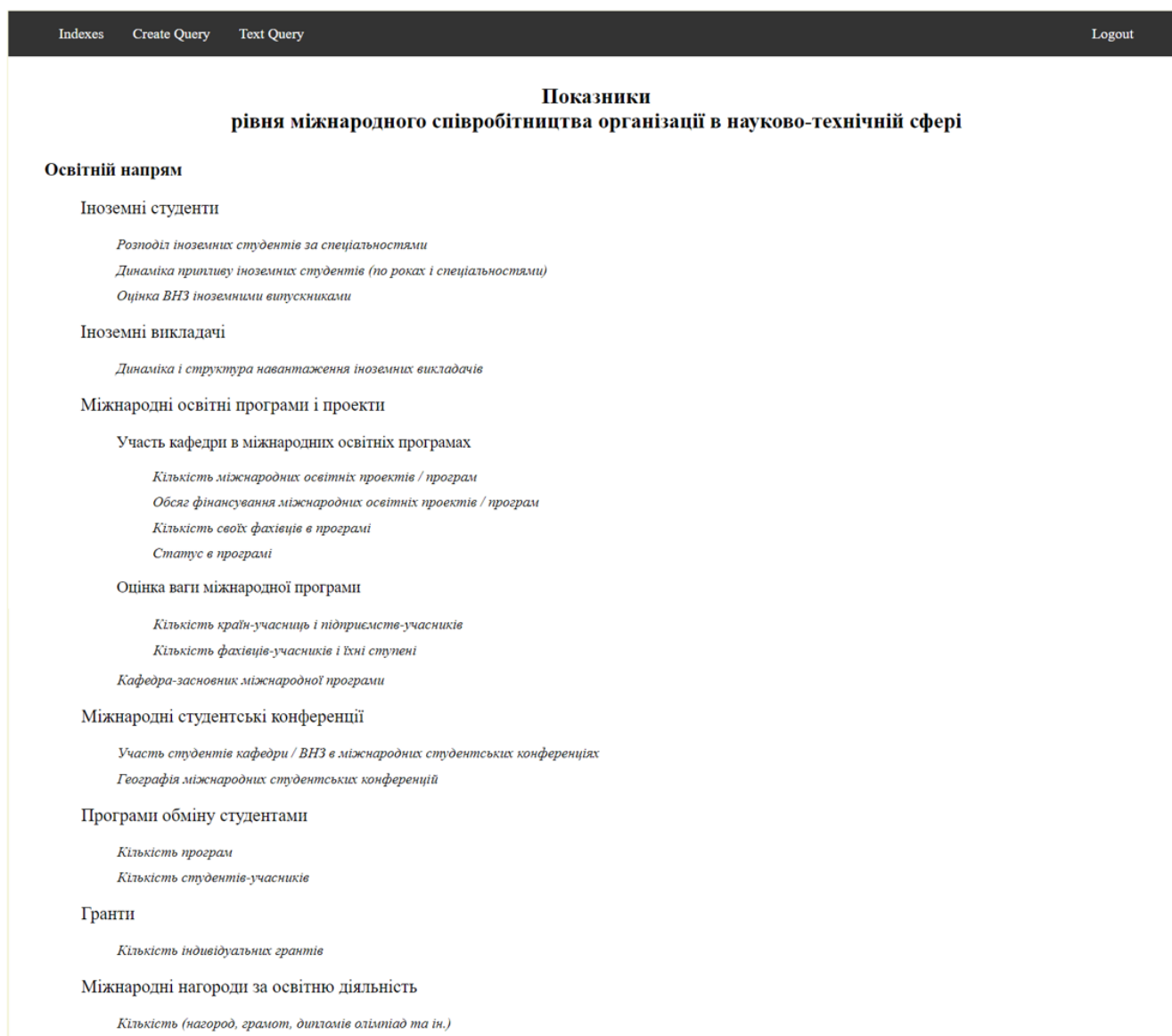


Рисунок 5.2 – Сторінка з показниками

Як можна помітити на рисунку 5.2 після авторизації кнопка Login замінилася кнопкою Logout. Натиснувши на неї, користувач вийде з системи, і ця кнопка кнопка заміниться назад кнопкою Login. Дані статичні показники є посиланнями.

Натиснувши на один з них, користувач отримує цей показник у вигляді таблиці (рисунок 5.3).

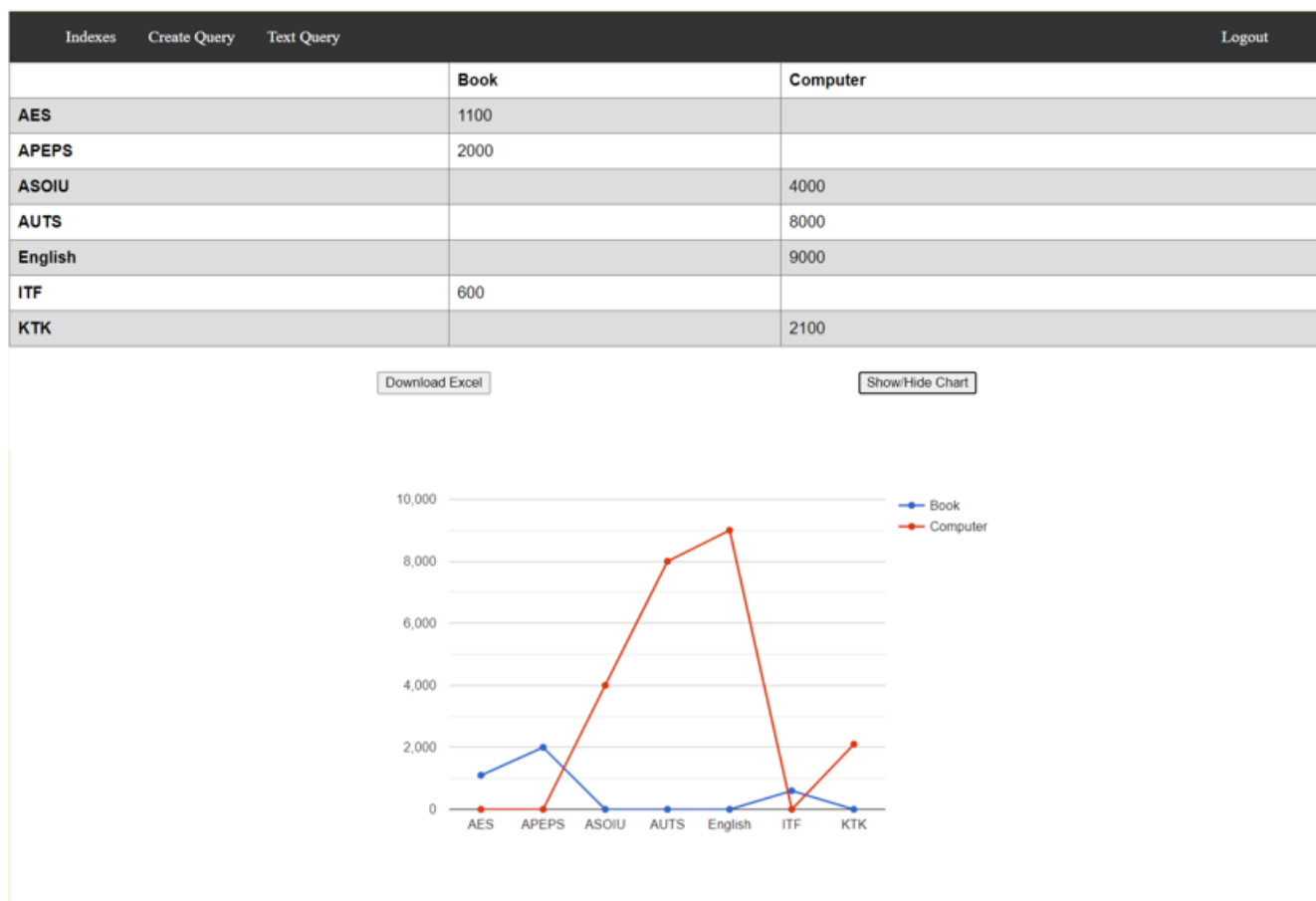


Рисунок 5.3 – Суми закупівель інвентарю за кафедрами та типом інвентарю

Як видно на рисунку 5.3 під таблицю є дві кнопки: завантажити цей результат у вигляді файлу Excel та показати/сховати графік. За замовчуванням графік відсутній.

Важливо відмітити, що отримати ці дані з серверу може тільки авторизований користувач. Якщо неавторизований користувач спробує перейти за посиланням, його поверне на сторінку входу.

Натиснувши в навігаційному меню на Create Query або перейшовши за адресою <http://localhost:4200/createQuery>, користувач потрапить на сторінку конструювання запиту (рисунок 5.4).

[Indexes](#)
[Create Query](#)
[Text Query](#)
[Logout](#)

Cube:

BooksAtDepartment ▼

Dimensions:

- Measures
 - Book Count
- Department Dim
 - City
 - (All)
 - City
 - Country
 - (All)
 - Country
 - Department
 - (All)
 - Department
 - Faculty
 - (All)
 - Faculty
 - GeographyHierarchy
 - (All)
 - Country
 - State Or Province
 - City

Query Structure:

Columns	
Rows	
Measures	

Clear table

Execute

Рисунок 5.4 – Сторінка конструювання запиту

Візуально цей компонент можна поділити на дві частини. У лівій знаходиться випадаючий список з усіма доступними у системі кубами. Обравши потрібний куб, підвантажуються усі його виміри, ієрархії та рівні ієрархій, які утворюють деревовидну структуру. У правій частині знаходяться таблиця, яка показує що буде виводитись на колонках, рядках, та яка міра вираховуватиметься під час запиту. Під таблицею є кнопка швидкого її очищення **Clear table** та кнопка виконання запиту **Execute**. Так як на двовимірному екрані неможливо відобразити більше ніж два виміри (перетин рядків і стовпчиків), виконання обмежене поверненням результату не більше ніж у двох вимірах.

Подвійний клік на рівень ієрархії заносить його у структуру запиту, про що буде видно у таблиці. Так як SQL Server Analysis Services не дозволяє виводити вимір, якщо попередній вільний (тобто не можна вивести значення на рядки, якщо вільні колонки), то рівні додаються в початок. Подвійний клік на рядок у таблиці, яка відображає структуру запиту видалить даний рівень, при цьому змістить усі наступні рівні назад (при заповнених колонках та рядках, видалення колонок поверне старі рядки у нові колонки).

Натиснувши кнопку Execute виконується запит. Результат відобразиться у вигляді таблицьки нижче (рисунок 5.5).

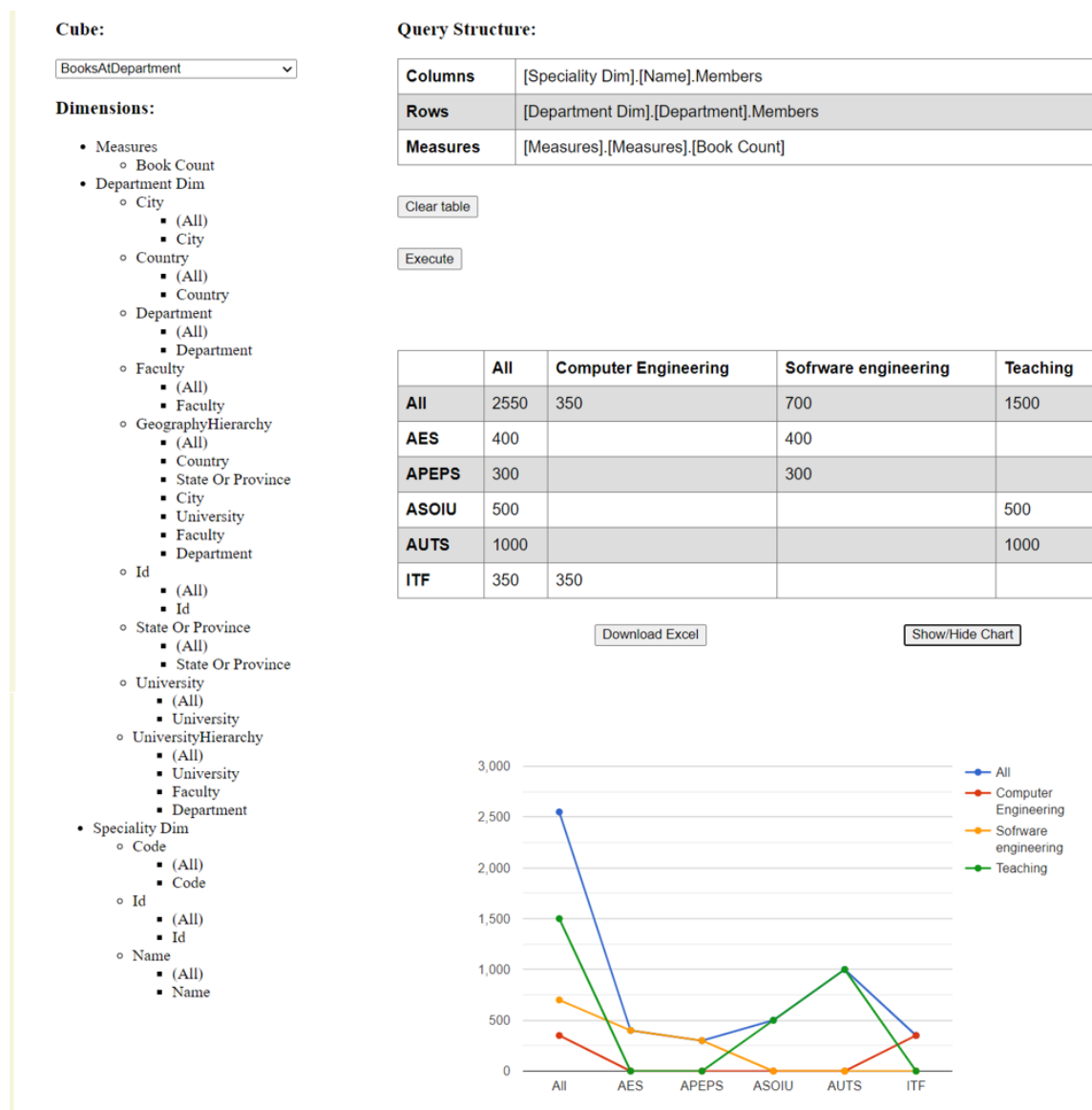


Рисунок 5.5 – Виконаний запит

Даний запит виводить сумарну кількість книжок, розраховану за кафедрами та спеціальностями. Вибір рівня (All) агрегує дані по всім рівням, а вибір ієрархії для запиту, поверне, як сумарне (All) значення, так і окремо по кожному рівню.

Варто відмітити, що для кожного кубу існує міра за замовчуванням, і показаний вище запит виконався і без явного вказання міри, адже вона тут одна. Проте не варто

покладатися на міри за замовчуванням, адже про неї знать тільки люди, що створювали даний куб. Тому рекомендовано завжди явно вказувати міру.

Якщо конструктор не може дати вам потрібний запит, можна перейти на сторінку текстового запиту (рисунок 5.6) натиснувши на кнопку Text Query.

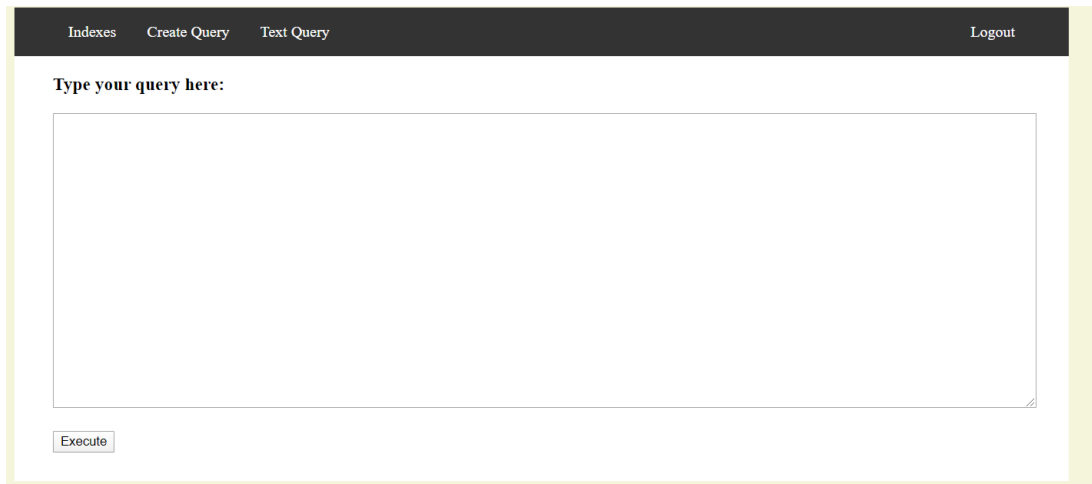


Рисунок 5.6 – Сторінка текстового запиту

На сторінці знаходиться текстове поле, куди користувач вводить запит на кнопка Execute, яка виконує запит (рисунок 5.7).

Type your query here:

```
SELECT
    NON EMPTY
    [Inventory Dim].[Type].[Type]
        ON COLUMNS,
    NON EMPTY
    [Department Dim].[Department].[Department]
        ON ROWS
FROM
    [BuyingInventory]
WHERE
    [Measures].[Sum];
```

Execute

	Book	Computer
AES	1100	
APEPS	2000	
ASOIU		4000
AUTS		8000
English		9000
ITF	600	
KTK		2100

Рисунок 5.7 – Виконаний запит

Таким чином, користувач може створити власний запит та виконати його. Спроба виконати запит неавторизованим користувачем поверне його на сторінку входу.

Для додавання даних до системи користувач має натиснути кнопку Add Data з навігаційного меню. Після цього відкриється сторінка з формою для введення строки підключення до бази даних та кнопок для додавання даних з бази даних та перерахунку кубів (рисунок 5.8).

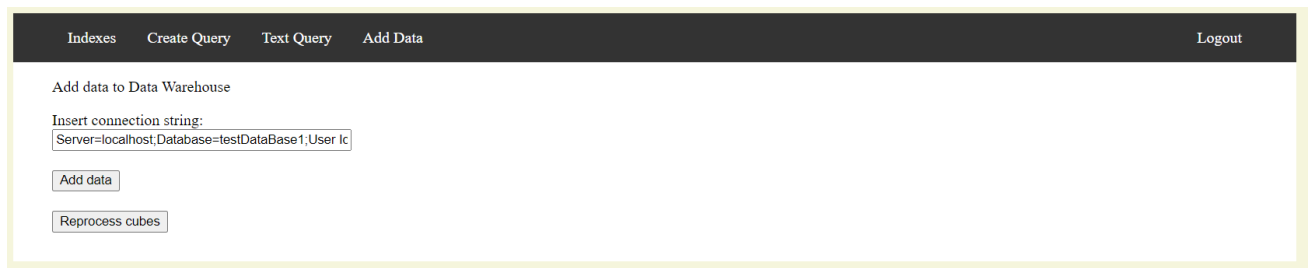


Рисунок 5.8 – Сторінка додавання даних в систему

Якщо користувач помилиться під час написання запиту у текстовому режимі або у конструкторі, він отримає відповідне повідомлення у діалоговому вікні у браузері (рисунки 5.9 і 5.10).

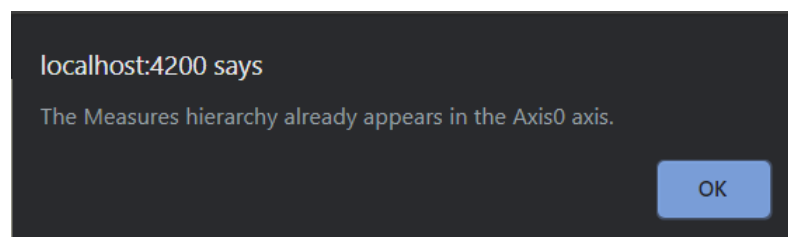


Рисунок 5.9 – Помилка під час виконання запиту

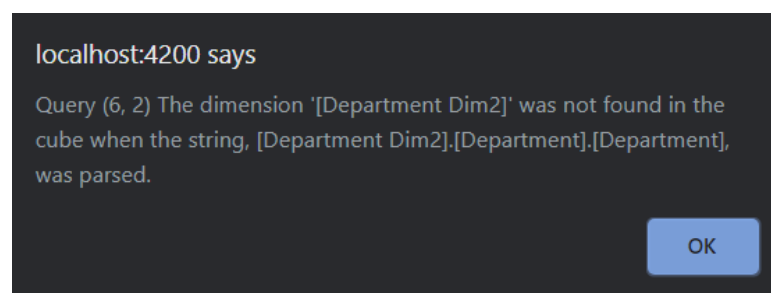


Рисунок 5.10 – Помилка під час виконання запиту

Будь-які інші помилки, які виникли під час роботи програми будуть також донесені до користувача у зрозумілій формі через це діалогове вікно.

5.3 Висновки

У даному розділі було описано необхідні дії для запуску системи та наведено приклади інтерфейсу користувача. Пояснено як користуватися даною системою. Як приклад зроблено вхід у систему, виконано статичний запит, створено власний запит через візуальний конструктор запитів та текстовий MDX-запит. Показано приклади помилок, що можуть виникнути під час роботи.

ВИСНОВКИ

Під час виконання дипломної роботи було розглянуто існуючі рішення для роботи з багатомірними базами даних на основі OLAP – Microsoft Excel та SQL Server Management Studio, виявлено переваги та недоліки даних програм та на основі них сформовані вимоги до створюваного програмного забезпечення.

Спроектовано структуру та створено сховище даних на основі отриманих показників, які треба аналізувати.

Проаналізовано різні підходи до організації структури багатовимірної бази даних: єдиний куб для системи та багато кубів під кожну зірку сховища даних, виявлено їхні переваги і недоліки, обґрунтовано вибір другої структури.

Створено багатовимірну базу даних, яка реалізовує технологію OLAP для швидкого виконання аналітичних запитів.

Розроблено веб-сервер на основі ASP.NET Core Web API із трьохрівневою архітектурою для полегшення розробки та подальшої підтримки.

Реалізовано систему авторизації ASP.NET Core Identity, яка захищає доступ до даних і дозволяє встановлювати програму навіть на офісних комп'ютерах.

Розроблено інтерфейс користувача у вигляді односторінкового додатку за допомогою Angular 9 для покращення досвіду користувача.

В результаті роботи створено програмний продукт, який, використовуючи OLAP, дозволяє виконувати роботу по аналізу показників рівня міжнародного співробітництва у науково-технічній сфері. Розроблений продукт є незалежним від операційної системи, адже являється Web-додатком. Програма дозволяє швидко отримувати визначені предметною областю показники, будувати нові показники у режимі конструктора або напряму виконувати MDX-запити.

Вхідні дані: назва встановленого показника, запит, створений у конструкторі, або MDX-запит.

Вихідні дані: показник у вигляді таблиці, файл Excel з результатом запиту, лінійний графік результату запиту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ADOMD.NET [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/analysis-services/adomd/developing-with-adomd-net?view=asallproducts-allversions>.
2. Create web APIs with ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1>.
3. Introduction to Identity on ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1>.
4. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction/>.
5. Microsoft SQL Server Management Studio (SSMS) [Електронний ресурс] – Режим доступу до ресурсу: <https://searchsqlserver.techtarget.com/definition/Microsoft-SQL-Server-Management-Studio-SSMS>.
6. Online analytical processing [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Online_analytical_processing.
7. The best new features in Visual Studio 2019 [Електронний ресурс] – Режим доступу до ресурсу: <https://betanews.com/2019/04/02/best-new-features-visual-studio-2019/>.
8. Understanding Multidimensional Databases [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/cd/E57185_01/EDBAG/understanding_multidim_dbs.html.
9. What are the Advantages and Disadvantages of Angular? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.edureka.co/blog/advantages-and-disadvantages-of-angular/>.

10. What is Data Warehouse? Types, Definition & Example [Электронный ресурс] – Режим доступа до ресурсу: <https://www.guru99.com/data-warehousing.html>.
11. Введение в Web API [Электронный ресурс] – Режим доступа до ресурсу: <https://metanit.com/sharp/aspnet5/23.1.php>.
12. Виртуальный куб — вместо OLAP [Электронный ресурс] – Режим доступа до ресурсу: <https://habr.com/ru/post/452154/>.

Додаток А

Система аналізу показників рівня міжнародного співробітництва

УКР.НТУУ“КПІ”.ТВ6141_20Б

Специфікація

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ“КПІ”.ТВ614 1_20Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ».ТВ614 1_20Б 12-1	Текст програмного модулю	Частина веб-серверу, що відповідає за виконання запитів
УКР.НТУУ«КПІ».ТВ614 1_20Б 12-2	Текст програмного модулю	Скрипт створення сховища даних
УКР.НТУУ«КПІ».ТВ614 1_20Б 12-3	Текст програмного модулю	Наповнення сховища тестовими даними
УКР.НТУУ«КПІ».ТВ614 1_20Б 12-4	Текст програмного модулю	Інтерфейс користувача
УКР.НТУУ«КПІ».ТВ614 1_20Б 12-5	Текст програмного модулю	Показники у вигляді MDX-запитів до багатовимірної БД
УКР.НТУУ«КПІ».ТВ614 1_20Б 13-1	Опис програмного модулю	Опис модулю з веб- сервером

Додаток Б

Система аналізу показників рівня міжнародного співробітництва

УКР.НТУУ «КПІ».ТВ6141_20Б 12-1

Текст програмного модулю

Аркушів 7

```

//Сервіс для реалізації логіки роботи з кубами
namespace BLL.Services
{
    public class CubeService : ICubeService
    {
        //Поля для збереження маперу та доступу до рівня даних
        private IMapper _mapper;
        private ICubeDataGateway _cubeDataGateway;

        //Конструктор з механізмом DI
        public CubeService(IMapper mapper, ICubeDataGateway cubeDataGateway)
        {
            _cubeDataGateway = cubeDataGateway;
            _mapper = mapper;
        }

        //Метод для виконання статичного запиту
        public QueryResultDTO ExecuteQuery(string query, object[] parameters)
        {
            MethodInfo method = _cubeDataGateway.GetType().GetMethod(query);

            var cube = _mapper.Map((CellSet)method.Invoke(_cubeDataGateway,
parameters));

            return cube;
        }

        //Метод отримання інформації про куб
        public CubeDTO GetCubeInfo(string cubeName)
        {

```

```

        return _cubeDataGateway.GetCubeInfo(cubeName);
    }

    //Метод отримання назв усіх кубів
    public IEnumerable<string> GetCubeNames()
    {
        return _cubeDataGateway.GetCubeNames();
    }

    //Метод виконання запиту користувача
    public QueryResultDTO ExecuteCustomQuery(string query)
    {
        return _mapper.Map(_cubeDataGateway.ExecuteCustomQuery(query));
    }
}

//Контроллер для обробки запитів
namespace OLAPWebApiCore_01.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Authorize(Roles = "User")]
    public class CubesController : ControllerBase
    {
        //Посилання на сервіс
        private ICubeService _cubeService;

        //Контроллер з механізмом DI

```

```

public CubesController(IMapper mapper, ICubeService cubeService)
{
    _cubeService = cubeService;
}

```

//Метод обробки get запиту для отримання назв існуючих кубів
[HttpGet("")]

```

public IEnumerable<string> GetCubeNames()
{
    return _cubeService.GetCubeNames();
}

```

//Метод обробки get запиту для отримання детальної інформації про куб
[HttpGet("{cubeName}")]

```

public CubeDTO GetCube(string cubeName)
{
    return _cubeService.GetCubeInfo(cubeName);
}

```

//Метод обробки get запиту з виконанням статичного запиту і можливим параметром

[HttpGet("queries/{query}/{param?}")]

public QueryResultDTO ExecuteStaticQueryWithParam(string query, string param)

```

{
    object[] userParams = { param };
    return _cubeService.ExecuteQuery(query, param == null ? null : userParams);
}

```

```
//Метод обробки post запиту з виконанням MDX-запиту користувача
[HttpPost("query")]
public QueryResultDTO ExecuteCustomQuery([FromBody]string query)
{
    return _cubeService.ExecuteCustomQuery(query);
}

}
```

```
//Рівень доступу до даних
namespace DAL.DataGateways
{
```

```
public class CubeDataGateway : ICubeDataGateway
{
    //Строка підключення до серверу багатовимірної БД
    private string _connectionString { get; }

    //Конструктор з механізмом DI
    public CubeDataGateway(string connectionString)
    {
        _connectionString = connectionString;
    }
}
```

```
//Отримання назв усіх кубів
public IEnumerable<string> GetCubeNames()
{
```

```
    using (AdomdConnection connection = new
AdomdConnection(_connectionString))
```

```

    {
        connection.Open();

        //Вибрати всі куби, крім системних
        var result = connection.Cubes.Cast<CubeDef>().Where(x =>
!x.Caption.StartsWith("$")).Select(x => x.Caption).ToList();

        connection.Close();
        return result;
    }
}

//Отримання детальної інформації про куб
public CubeDTO GetCubeInfo(string cubeName)
{
    using (AdomdConnection connection = new
AdomdConnection(_connectionString))
    {
        connection.Open();

        //Знайти куб
        var cubeDef = connection.Cubes.Cast<CubeDef>().Where(x => x.Caption
== cubeName).FirstOrDefault();

        //перевірка на існування кубу
        if (cubeDef == null)
        {
            throw new ArgumentException($"There is no cube with name
{cubeName}");
        }
    }
}

```

```

//Отримання вимірів кубу
var dimensions = new List<Entities.Dimension>();
foreach (var dimension in cubeDef.Dimensions)
{
    //Отримання ієрархій
    var hierarchies = new List<Entities.Hierarchy>();
    foreach (var hierarchy in dimension.Hierarchies)
    {

        var levels =
hierarchy.Levels.Cast<Microsoft.AnalysisServices.AdomdClient.Level>().
        Select(level => new Entities.Level
        {
            Caption = level.Caption,
            Members = level.GetMembers().Cast<Member>().Select(memb
=> memb.Caption).Distinct()
        }).ToList();

        hierarchies.Add(new Entities.Hierarchy { Caption =
hierarchy.Caption, Levels = levels });
    }

    dimensions.Add(new Entities.Dimension { Caption =
dimension.Caption, Hierarchies = hierarchies });
}

var cube = new CubeDTO { Caption = cubeDef.Caption, Dimensions =
dimensions };

```



```

        //Закриття з'єднання з БД
        connection.Close();

        return cube;
    }
}

//Виконання MDX-запиту користувача
public CellSet ExecuteCustomQuery(string query)
{
    using (AdomdConnection connection = new
AdomdConnection(_connectionString))
    {
        connection.Open();

        //Створення команди
        AdomdCommand command = connection.CreateCommand();
        command.CommandText = query;

        var result = command.ExecuteCellSet();
        connection.Close();
        return result;
    }
}

```

Додаток В

Система аналізу показників рівня міжнародного співробітництва

УКР.НТУУ«КПІ».ТВ6141_20Б 13-1

Опис програмного модулю

Аркушів 7

АНОТАЦІЯ

Метою роботи було створення додатку для аналізу показників рівня міжнародного співробітництва у науково-технічній сфері. Програма дає змогу виконувати статичні запити або генерувати нові за допомогою візуального конструктору запитів або текстових MDX-запитів. Додаток має надійний захист від неавторизованих користувачів, що дає змогу встановлювати його навіть на робочих комп'ютерах.

ЗМІСТ

АНОТАЦІЯ	67
ЗМІСТ	68
1. ЗАГАЛЬНІ ВІДОМОСТІ	69
1.1 Опис логічної структури.....	69
1.2 Вхідні та вихідні дані	69
1.3 Використані технічні засоби	70

1. ЗАГАЛЬНІ ВІДОМОСТІ

Вищеописаний програмний модуль створений у MVS 2019 Community Edition у якості Web API на основі ASP.NET Core 3. Призначений для отримання метаданих про куби із багатовимірної бази даних та виконання запитів. У модулі використана бібліотека для роботи з багатовимірними БД ADOMD.NET.

1.1 Опис логічної структури

Даний програмний модуль описує взаємодію між рівнями на веб-сервері. Найвищим рівнем є рівень Web API. Після отримання запиту, фреймворк намагається переправити цей запит на контроллер, який зможе його опрацювати. Якщо даний контроллер буде не знайдено, сервер поверне помилку з кодом 404. Знайшовши контроллер, фреймворк ініціює створення об'єкту контроллеру, але так як він у конструкторі має параметри, спочатку створюються вони. Таким чином об'єкти створюються починаючи з найнижчого рівня. Після створення усіх потрібний об'єктів викликається відповідний метод контроллеру. Він в свою чергу виклає сервіс, щоб виконати певні дії, а сервіс, користуючись рівнем доступу до даних виконує свою роботу та повертає дані на Web API. В кінці, API розпоряджаючись результатом сервісу (в залежності від запиту) генерую відповідь користувачу, який прислав запит та відправляє її. У разі виникнення помилки, вона буде опрацьована ExceptionHandlerMiddleware, тому немає сенсу відловлювати їх у контроллері.

1.2 Вхідні та вихідні дані

Вхідними даними до програмного модулю є HTTP запит, присланий з клієнтського інтерфейсу. Вихідними даними є HTTP відповідь клієнту.

1.3 Використані технічні засоби

При роботі програмного модулю використовувався комп'ютер на операційній системі Windows 10 з процесором Intel Core i7 та 16Гб оперативної пам'яті. Робота сховища даних була на SQL Server 2019 Data Engine, багатовимірної бази даних – SQL Server 2019 Analysis Services, веб-серверу – IIS. Розроблена програма не залежить від платформи завдяки реалізації на ASP.NET Core 3, та клієнтському інтерфейсу, що працює у будь-якому сучасному браузері.