Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2022 року

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технології»

на тему:

МЕТОДИ ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «ЕЛЕКТРОННИЙ РОЗКЛАД ФАКУЛЬТЕТУ» НА ПЛАТФОРМІ .NET

Виконав: студент групи МІТм-21

Старчевий А. О. _____
(прізвище ім'я по-батькові) (підпис)

Керівник: асистент кафедри мережевих та інтернет технологій
к.т.н. Махович О. І. _____
(посада, прізвище ім'я по-батькові) (підпис)

Київ 2022

РЕФЕРАТ

Пояснювальна записка: 66 с., 38 рисунків, 3 табл., 1 додаток, 25 джерел.

Об’єкт дослідження: методи проектування та реалізація інформаційної системи «Електронний розклад факультету» на платформі .NET

Предмет дослідження: розробка схеми бази даних бази, реалізація бази даних та розробка веб-додатку «Електронний розклад факультету» на платформі .NET

Мета роботи (проекту): провести теоретичний аналіз методичної літератури по розробці інформаційної системи на платформі .NET. Проаналізувати предметну область «Електронний розклад факультету». Спроектувати ефективну схему бази даних. За допомогою набутих знань та методичного інструментарію розробити веб-додаток «Електронний розклад факультету».

В роботі проведено аналіз методичної літератури по інформаційним системам, системам управління, мовам SQL, C#, технологіям ASP.NET Core та Entity Framework,.

Спроектовано та побудовано базу даних для інформаційної системи «Електронний розклад факультету».

Розроблено веб-додаток, на якому реалізована обробка, зберігання та представлення даних про розклад, викладачів та пари. Також реалізовано розподілення на ролі, такі як адміністратор, викладач та студент.

Результати здійснених у дипломному проекті досліджень можуть бути використані як основа розробки додатку або бази даних для розкладу факультету або деякої учбової установи.

Ключові слова: SQL, .NET, СУБД, MICROSOFT SQL SERVER, ASP.NET CORE, MVC, TRANSACT-SQL, ENTITY FRAMEWORK, RAZOR, МОДЕЛІ, КОНТРОЛЕРИ

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

SQL - Structured Query Language

TCL - Transaction Control Language

DDL - Data Definition Language

DML - Data Manipulation Language

DCL - Data Control Language

DQL - Data Query Language

СУБД – Система управління базами даних

EDM - Entity Data Model

MVC – Model-View-Controller

T-SQL – Transact-SQL

HTML – HyperText Markup Language

РСУБД - Реляційна система управління базами даних

ACID - Atomicity, consistency, isolation, durability

НФ - Нормальна форма

MVCC - Multiversion Concurrency Control

ORM - Object-Relational Mapping

ВСТУП

Стрімкий розвиток засобів обчислювальної техніки, поширення дистанційного навчання, розширення спектру технічних засобів, а також значне збільшення кількості інформації змушують світ навколо себе змінюватися. Сучасні системи управління базами даних забезпечують точний та повний аналіз, обробку, зберігання та отримання інформації. Встає гостра потреба автоматизації та впровадження інформаційних технологій в освіту.

Розпочнемо з визначення поняття “Автоматизація”. Зазвичай під автоматизацією розуміють використання інформаційних технологій та технічних засобів для повного або часткового звільнення людини від безпосередньо процесів отримання, зберігання або обробки інформації. Переваги автоматизації та впровадження інформаційних технологій на сьогодні очевидні. Це і підвищення якості та зниження помилок при роботі з інформацією, це і економія часу та грошей.

Одним з основних елементів автоматизації в освіті має бути можливість швидко та зручно переглядати, зберігати та обробляти інформацію про розклад. На жаль, на даний момент ця технологія є недостатньо вивченою. Більша кількість університетів не мають зручного та функціонального додатку з розкладом. Часом, це призводить до плутанини через те, що при освітньому процесі часто відбуваються заміни або відміни пар, викладачів, аудиторій та іншого. А зручний та багатофункціональний додаток дозволить уникнути великій кількості неприємностей, пов’язаним з цим.

Головною частиною електронного розкладу факультету являється гарно спроектована база даних, яка дозволить автоматизувати та спростити процеси, що відбуваються під час навчання. Більшість систем управління базами даних тримаються на єдиному комплексі правил та стандартів. Це дозволяє на прикладі однієї СУБД розглянути всі процеси та прийоми, які необхідні при роботі з ними.

Основні компанії, які працюють в сфері розробки систем управління: Oracle, Microsoft або PostgreSQL.

Практичний результатом даної роботи являється спроектована схема база даних для розкладу у закладі освіти, а також реалізація даної схеми у СУБД. Також серед результатів буде прототип додатку “Електронний розклад факультету”. Базу даних та прототип додатку можна буде використовувати як основу для сайту з розкладом для будь-якого закладу освіти.

1 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Реляційні бази даних

Одним з найпопулярніших типів баз даних є реляційні бази даних. Вони використовуються для зберігання, обробки та виведення інформації. Основою реляційної моделі являється візуальний табличний спосіб зберігання даних. Інформація, що яка знаходиться у реляційній моделі є записом з ідентифікатором. Також є дуже зручний зв’язок між таблицями з даними[1].

Реляційна модель передбачає логічні структури даних: таблиці, строки, процедури, функції та індекси. Логічна структура відрізняється від фізичної структури зберігання. Це розділення дозволяє адміністраторам керувати фізичною системою зберігання без зміни даних у логічній структурі. Щоб забезпечити точність і доступність даних, у реляційних базах повинні дотримуватися певних правил.

1.1.1 ACID

Транзакції реляційних баз даних визначаються чотирма основними властивостями: атомарність, узгодженість, ізольованість та довговічність, які зазвичай позначаються абревіатурою ACID (атомарність, узгодженість, ізольованість, довговічність). ACID - це набір властивостей, що гарантують надійну роботу баз даних[2].

Атомарність гарантує, що жодна операція не виконується частково, навіть якщо транзакція містить кілька операцій. Усі зміни даних для транзакції виконуватимуться так, ніби це була операція. Тобто буде внесено або всі зміни або жодних.

Узгодженість: система має перебувати в узгодженому стані до і після транзакції

Ізольованість: проміжний стан транзакції невидимий для інших транзакцій. Таким чином, одночасні транзакції здаються серіалізованими.

Довговічність: у будь-якому випадку після відновлення системи результати вже завершених транзакцій будуть збережені.

1.1.2 Нормальні форми

Нормальна форма - властивість відношення в реляційній моделі даних, що характеризує його з погляду надмірності. Нормальна форма являється сукупністю вимог, яким мають задовольняти відношення у РСУБД. Процес перетворення бази даних до виду, що відповідає нормальним формам, називається нормалізацією.

Відношення знаходиться в першій нормальній формі, якщо всі його атрибути є простими, всі домени, що використовуються, повинні містити тільки скалярні значення. Не повинно бути повторень рядків у таблиці[3].

Друга нормальна форма передбачає що кожен стовпець, який не є ключем, повинен залежати від первинного ключа.

Третя нормальна форма передбачає що кожен стовпець, який не є ключем, повинен залежати тільки від первинного ключа.

Четверта нормальна форма застосовується для усунення багатозначних залежностей - тих, де стовпець з первинним ключем має зв'язок один до багатьох зі стовпцем, який не є ключем. Ця нормальна форма усуває некоректні відносини багато до багатьох.

П'ята нормальна форма ділить таблицю на менші задля усунення надмірності даних. Розбиття має йти доти, доки не можна буде відтворити оригінальну таблицю шляхом об'єднання малих таблиць.

База даних знаходиться в шостій нормальній формі тоді і тільки тоді, коли вона задовольняє всім нетривіальним залежностям з'єднання. З визначення випливає, що змінна знаходиться в шостій нормальній формі тоді і тільки тоді, коли вона не приведена, тобто не може бути піддана подальшій декомпозиції без втрат. Кожна база даних, яка перебуває у 6НФ, також перебуває й у 5НФ.

База даних знаходиться в досконалій кон'юнктивній нормальній формі тоді і тільки тоді, коли кожне накладене на неї обмеження є логічним наслідком обмежень доменів та обмежень ключів, накладених на цю змінну відношення.

1.2 Особливості нереляційних баз даних

NoSQL – це ще один з варіантів підходу до реалізації масштабованої бази даних з гнучкою моделлю даних[4]. Однак на відміну від SQL інформація тут зберігається без суворої структури та явного зв'язку між різними даними. Дані тут зберігаються не тільки в табличній, а й в аудіо, відео, графічній, текстовій та будь-якій іншій формі. Широке застосування такі БД отримали комп'ютерних та мобільних програмах. Вони використовуються тоді, коли структурування даних не є в пріоритеті, за те потрібна гнучка база даних з високим рівнем продуктивності та можливістю до масштабування.

Основні переваги нереляційних баз даних:

- Висока гнучкість;
- Ефективність;
- Хороша масштабованість.

Також розглянемо основні недоліки нереляційних баз даних:

- Через відсутність стандартів є велика різниця між різними СУБД. Через це є проблеми швидкого переходу з однієї нереляційної СУБД на іншу;
- При розробці в NoSQL доводиться розробляти власні інструменти для роботи з БД;
- Обмежена ємність вбудованої мови запитів.

1.2.1 Вибір між SQL та NoSQL

Проводячи аналіз систем управління базами даних, необхідно відразу вибрати оптимальний варіант. При розробці будь-якого проекту, розумне планування розробки бази даних дозволить заощадити чимало часу.

В даній роботі існують ознаки того, що для нашого додатку підходять саме реляційні СУБД, серед них:

- Є логічні вимоги до даних, які можна визначити заздалегідь.
- Дуже важлива цілісність даних.
- Потрібна заснована на стандартах технологія, що добре зарекомендувала себе, використовуючи яку можна розраховувати на великий досвід розробників і технічну підтримку.

- Потрібна обробка великої кількості запитів та рутинного аналізу даних. Також в цій роботі потрібна надійна обробка транзакцій.

1.3 SQL

SQL це інструмент для організації, керування, вибору і обробки інформації, що зберігається в базі даних [5]. SQL має багато функціональних можливостей. Розглянемо їх:

1. Вибірка даних. Можливість вибирати частину або всю інформацію з бази даних;
2. Обробка даних. Дозволяє обробляти базу даних. Наприклад, додавати, видаляти або відновлювати дані;
3. Спільне використання. У SQL можна працювати декільком користувачам одночасно, і при цьому не буде порушуватися цілісність бази даних. Всі зміни будуть коректними.
4. Визначення даних. В SQL ви можете визначати структуру і організацію даних, що зберігаються та взаємозв’язки між елементами даних;
5. Управління доступом. В SQL можна певною мірою обмежити можливості користувачів. Наприклад, заборонити йому додаванти і змінювати дані. Це захищає базу даних від несанкціонованого доступу;
6. Цілісність даних. SQL забезпечує цілістність даних у базі, захищаючи її від руйнування через неузгоджені зміни або відмови системи.

SQL складається з п’яти частин:

DML (Data Manipulation Language). Мова керування даними. Дані в реляційних базах даних керуються за допомогою DML команд. Це команди

INSERT, UPDATE, DELETE та MERGE. Команда INSERT може додати один рядок в одну таблицю або багато рядків у багато таблиць. Приклад коду:

INSERT INTO table [(column [,column…])] VALUES (value [,value…]);

Команда UPDATE використовується при необхідності зміни рядків, які вже існують – які, наприклад, створені за допомогою команди INSERT. Команда UPDATE також може впливати або один рядок або набір рядків. Вибірка даних, що оновлюється за допомогою UPDATE, визначається умовою WHERE. Синтаксис UPDATE:

UPDATE table SET column=value [column=value…] [WHERE condition];

Додані в таблицю рядки можна видалити за допомогою команди DELETE. Приклад:

DELETE FROM table [WHERE condition];

Іноді виникає ситуація, коли необхідно взяти набір даних та інтегрувати в існуючу таблицю. Це можна зробити за допомогою команди MERGE. MERGE допомагає працювати з наборами даних, для кожного рядка джерела намагається знайти вже існуючий рядок у таблиці. Якщо MERGE не знаходить збіг – рядок буде додано; якщо ж рядок знайдено, то він буде оновлений. Приклад коду:

```
merge into employees e
using Table1 n on (e.id = n.id)
when matched then
update set e.field1=n.field1
when not matched then
insert (id,field2,field3)
values (n.id,n.field2,n.field3);
```

DDL (Data Definition Language). DDL – це група операторів визначення даних. За допомогою операторів, які входять до цієї групи, можна визначити структуру бази даних та працювати з об'єктами цієї бази. До цієї групи входять такі оператори:

Оператор CREATE використовується для створення нової таблиці в базі даних. Синтаксис:

CREATE TABLE Table1 (column1 datatype, column2 datatype, column3 datatype, ...);

ALTER використовується для додавання, видалення або зміни стовпців у існуючій таблиці.

ALTER TABLE Table1 ADD columnName datatype;

DROP TABLE видаляє існуючу таблицю використовуючи конструкцію

DROP TABLE Table1;

Data Control Language (DCL) – група операторів, яка використовується для визначення прав доступу до даних. Це оператори для керування дозволами, за допомогою яких можна дозволяти або забороняти виконання деяких операцій над об'єктами бази даних. Основні команди цієї групи операторів це GRANT та REVOKE. Перша команда надає користувачам права доступу до бази даних за допомогою

GRANT privileges_names ON object TO user;

Друга - команда скасовує привілеї доступу користувача, надані за допомогою команди GRANT і має синтаксис

REVOKE privileges_names ON object TO user;

Transaction Control Language. Команди TCL використовуються для керування транзакціями в базі даних. Вони використовуються для керування змінами, внесеними операторами DML. Це також дозволяє групувати команди в логічні транзакції.

Приклади команд TCL:

Команда COMMIT використовується для збереження та застосування будь-якої транзакції в базу даних.

Команда ROLLBACK відновлює базу даних до останнього зафіксованого стану. Також ця команда використовується з командою SAVEPOINT для переходу до конкретної точки збереження.

Команда SAVEPOINT використовується для тимчасового збереження транзакції, і якщо вам потрібна можливість повернутися до цієї точки у разі потреби.

DQL(Data Query Language) використовуються для виконання запитів до даних. DQL включає оператор SELECT. Ця команда дозволяє отримати дані з бази даних для виконання операцій з ними. Коли ми використовуємо SELECT результат компілюється в деяку тимчасову таблицю, яка відображається або, наприклад, отримується програмою.

Оператор SELECT складається з кількох розділів:

SELECT визначає список стовпців(існуючих або обчислюваних), що повертаються, їх імена, обмеження:

FROM вказує, з якої таблиці проводити вибір даних.;

WHERE визначає обмеження на рядки;

GROUP BY поєднує ряди, що мають однакову властивість;

HAVING вибирає з груп означених оператором GROUP BY;

ORDER BY задає критерії сортування рядків;

Розглянемо на рисунку 1.1 структуру SQL команд.

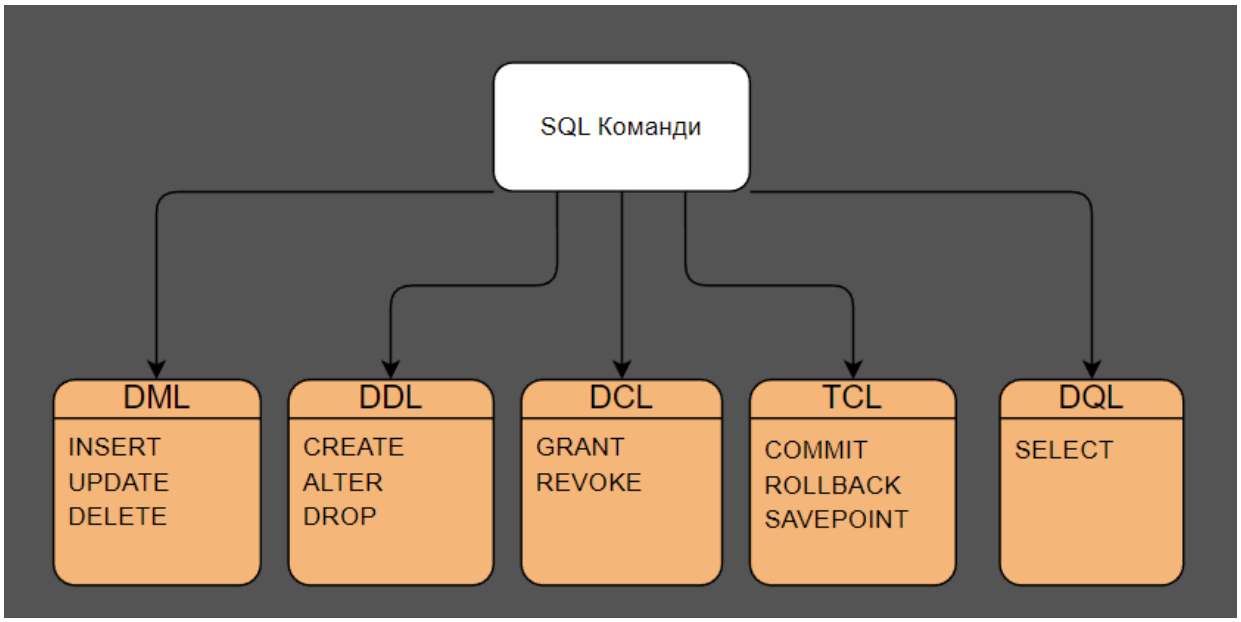

Рисунок 1.1 - Команди SQL

1.3.1 Багатотабличні запити, оператор SQL JOIN

JOIN - це команда в SQL, яка використовується при роботі з базами даних. Вона об'єднує дані із двох різних таблиць у базі. Основною метою використання команди є отримання потрібної підмножини даних. Команда JOIN в SQL є однією з найважливіших. Зазвичай JOIN використовується у блоках SELECT, які вибирають з бази таблиці, рядки або стовпці, що відповідають потрібним критеріям.

Загальна структура багатотабличного запиту:

```
SELECT field1, field2
FROM table1
[INNER] | [[LEFT | RIGHT | FULL][OUTER]] JOIN table2
```

ON умова_з’єднання

[[INNER] | [[LEFT | RIGHT | FULL][OUTER]] JOIN table3

ON умова_з'єднання]

INNER JOIN. За замовчуванням JOIN виконується як INNER JOIN, тобто як внутрішнє з'єднання таблиць. Внутрішнє з'єднання - з'єднання двох таблиць при якому утворюються всі можливі комбінації рядків обох таблиць, тобто кожен рядок з першої таблиці з'єднується з кожним рядком другої таблиці. На рисунку 1.2 можна побачити ілюстрацію INNER JOIN.

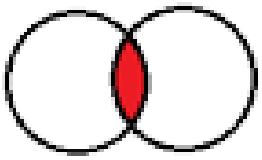

Рисунок 1.2 - INNER JOIN

Другий поширений варіант – OUTER JOIN, зовнішнє з'єднання. Якщо INNER JOIN можна порівняти з бінарним «і», то OUTER — варіації бінарного «або». Такий JOIN повертає не тільки перетин між двома таблицями, а й окремі елементи, які належать лише одному з множин.

Left Join. Повертає перетин множин та всі елементи з лівої таблиці. Right Join працює аналогічно, але замість лівої таблиці права. Join Full повертає обидві таблиці, які об'єднує в одну. Ілюстрацію цих трьох варіацій OUTER JOIN можна побачити на рисунку 1.3.

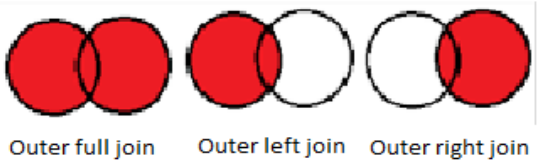

Рисунок 1.3 - OUTER JOIN

OUTER JOIN з NULL. По суті це той же самий Outer Join, але з додатковим параметром, який прибирає з результатів пошуку перетин категорій. Являється протилежністю INNER JOIN. На рисунку 1.4 можна побачити принцип роботи OUTER JOIN з NULL.

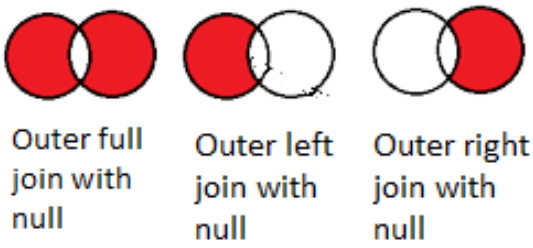

Рисунок 1.4 - OUTER JOIN з NULL

CROSS JOIN повертає декартовий добуток — він збирає всі можливі пари з обох таблиць. CROSS JOIN, на відміну від інших JOIN команд не вимагає вказівки додаткової інформації. Являється аналогом звичайного звернення до двох таблиць, але відрізняється від нього тим, що вони поєднуються в одну таблицю. Ілюстрацію роботи CROSS JOIN можна побачити на рисунку 1.5.

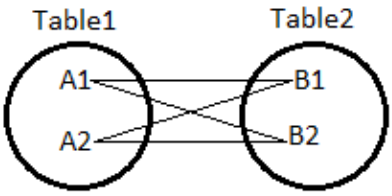

Рисунок 1.5 - CROSS JOIN

SELF JOIN - об'єднання всередині однієї таблиці. Використовується для ситуацій, в яких різні поля однієї таблиці мають однакові значення. SELF JOIN може бути внутрішнім або зовнішнім.

1.3.2 Процедури та тригери. Індексація

Іноді трапляється так, що деякий набір операцій потрібно виконати у конкретній послідовності. Наприклад при купівлі товару в магазині спочатку потрібно перевірити чи є товар в наявності. Тобто це процес, який охоплює декілька дій. В цьому випадку в SQL є можливість інкапсулювати ці дії в один об'єкт - процедуру[4]. Серед основних переваг процедур: безпека, продуктивність та спрощення коду. Для створення процедури використовується команда CREATE PROCEDURE. Приклад синтаксису:

```
CREATE PROCEDURE ProcedureName AS
BEGIN SELECT Field1, Field2, Field3
FROM Table1 END;
```

Для визову створеної процедури використовується код:

```
EXEC ProcedureName;
```

Тригери являються спеціальним типом процедури. Але у відмінності від звичайної процедури вона викликається автоматично при виконанні певної умови. Наприклад при додаванні, зміні або видаленні даних. Формальне визначення тригера:

```
CREATE TRIGGER TriggerName
ON Table1
{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]
AS sql_вираз
```

Існує два типи тригерів:

1. AFTER - виконується після виконання певної дії з таблицею.
2. INSTEAD OF - виконується замість виконання певної дії з таблицею.

Тригер можна видалити за допомогою команди DROP, відключити командой DISABLE та включити командою ENABLE.

Індекси – це спеціальні таблиці, які використовуються пошуковим рушієм бази даних для прискорення отримання інформації з таблиці. Для додавання індексу, нам необхідно використати команду CREATE INDEX. Це дасть можливість нам вказати ім’я , визначити тип індексу та назву таблиці. Існує два типи індексів: це кластерний та некластерний індекс.

1.4 Системи управління базами даних

База даних – це організований набір даних, які зазвичай зберігаються та мають електронний доступ з комп’ютерної системи[6]. Система управління базами даних – це програмне забезпечення, яке взаємодіє з кінцевими користувачами, програмами та самою базою даних для збору та аналізу даних [7].

Будь-яка сучасна система управління базами даних складається з кількох основних частин:

1. Ядро СУБД.
2. Процесор мови бази даних.
3. Підсистема підтримки часу виконання
4. Сервісних програм (зовнішніх утиліт)

СУБД має велику кількість функціональних можливостей. Розглянемо головні з них на рисунку 1.6.

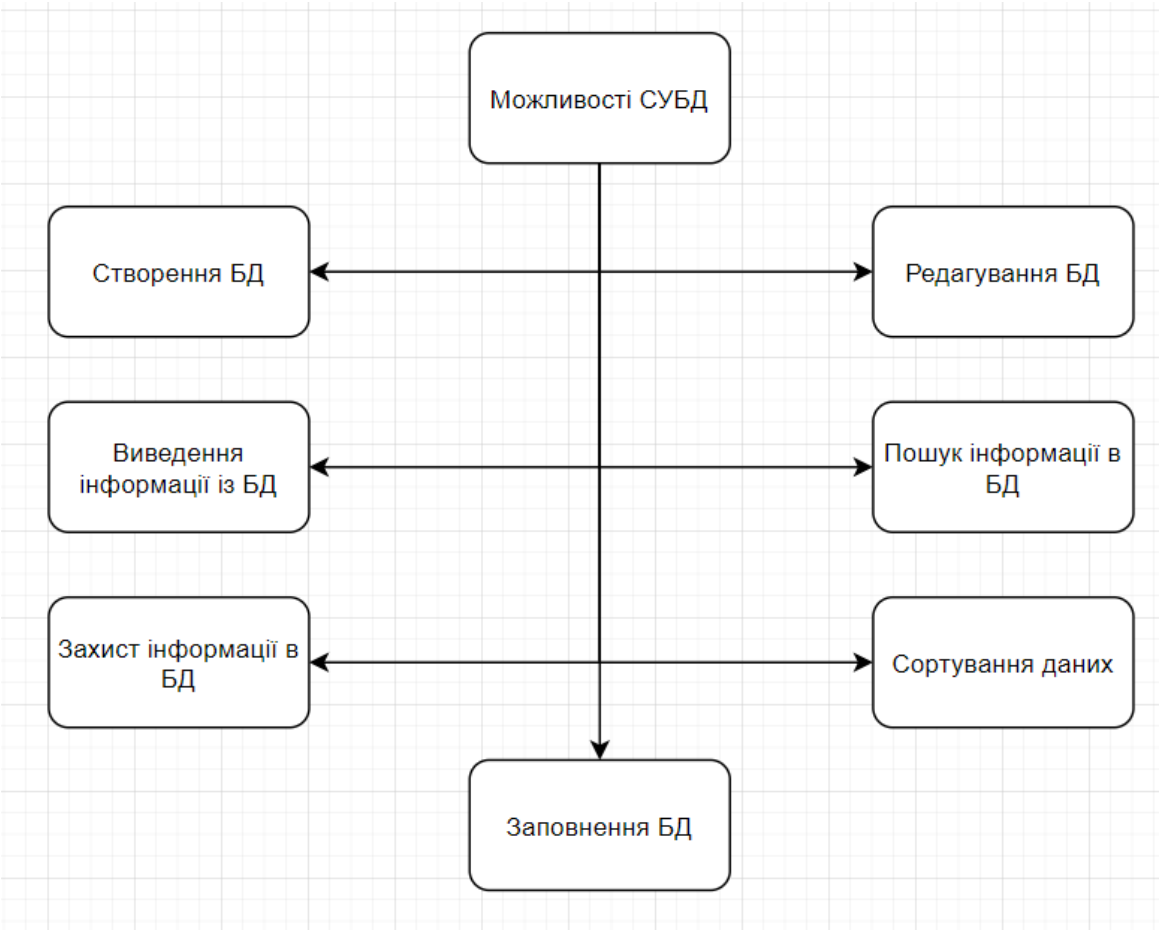

Рисунок 1.6 - Функціональні можливості СУБД

1.4.1 Вибір СУБД

1.4.1.1 Oracle Database

Oracle Database – це об’єктно-реляційна система управління базами даних, створена компанією Oracle [8]. Ідея створення належить Ларрі Еллісон

Розглянемо основні переваги Oracle Database:

1. Підтримується велика кількість користувачів, які, зокрема, можуть працювати одночасно;

2. Різні типи даних не суперечать між собою;
3. Є підтримка транзакцій;
4. Oracle Database є доволі відмовостойкою.
5. Можна легко переносити бази даних із однієї операційної системи в іншу без майже будь-яких змін.

Серед недоліків Oracle Database є такі:

1. Занадто висока вартість, особливо для невеликих компаній.
2. Невелика можливість масштабування;

1.4.1.2 PostgreSQL

PostgreSQL – це система управління реляційними базами даних із відкритим кодом. PostgreSQL підтримується співтовариством та поширюється за допомогою ліцензії. Серед головних переваг автори PostgreSQL виділяють відкритий код та масштабованість.[9]

Розглянемо основні переваги PostgreSQL. В даній СУБД є підтримка тригерів. Також інтегровані транзакції. Також в PostgreSQL є збережені процедури та автоматично оновлювані подання. Цілісність даних підтримується в PostgreSQL за допомогою MVCC (Multiversion Concurrency Control). Коли в базі даних працює декілька людей, кожен з них бачить деякий знімок даних, назалежно від поточного стану СУБД. В такій ситуації це захищає базу даних від неузгодженості, наприклад, коли декілька транзакцій внесуть різні зміни до одних і тих самих таблиць. В свою чергу це знижує рівень конфліктів і забезпечує більшу продуктивність. Також серед переваг треба виділити можливість створення модифікацій через відкритість коду. В даній СУБД підтримуються

індекси. Серед них хеш-індекси, R-дерево, B-дерево, GIST та GIN. Також тут є можливість спадкування від інших таблиць.

PostgreSQL має багато призначень. Він може обробляти наприклад невеликі бази даних на персональних комп’ютерах, але і дуже зручний при обробці дуже великих баз даних на веб-серверах при великій кількості користувачів.

Головною операційною системою для PostgreSQL є macOS, але вона також доступна і для Linux та Windows.

1.4.1.3 Microsoft SQL Server

Microsoft SQL Server – це система управління реляційними базами даних, розроблена корпорацією Майкрософт [10]. MS SQL Server велику кількість різних видань, які доповнюють один одного. Кожне з видань спрямовані на свою нішу клієнтів. Від невеликих додатків для персональних клієнтів та декількох користувачів до величезний додатків, які можуть працювати на веб-серверах. Розглянемо основні переваги цієї СУБД:

1. Зручна інтеграції з іншими продуктами Microsoft. Такими як Visual Studio, Windows та Microsoft Access;
2. Можливість створювати великі бази даних;
3. Легке масштабування за допомогою простої інтеграції між різними виданнями. Це дозволяє дуже просто збільшувати свої бази даних та переносити їх від персональних комп’ютерів до веб-серверів;
4. Велика кількість автоматизованих рутинних задач;
5. Можливість створювати профілі;
6. Вбудований зручний пошук.

Серед недоліків Microsoft SQL Server прийдеться виділити необхідність великої кількості ресурсів для роботи бази даних. Також великі видання можуть бути достатньо дорогими.

Але як можна побачити недоліки, в даній роботі вони є не дуже критичними. А через зручну інтеграцію в Visual Studio, Windows та великий рівень автоматизації було прийнято рішення використовувати цю СУБД в даній роботі.

Якщо було прийнято рішення робити дану роботу на базі Microsoft SQL Server, важливо буде розповісти більш докладно про цю СУБД та її архітектуру, а також про Transact-SQL.

1.5 Архітектура Microsoft SQL Server

Для того, щоб повернути нам результат у вигляді даних, SQL виконує досить багато різних операцій. Розглянемо, що саме відбувається з моменту, коли ми натиснули кнопку «Виконати» і надіслали SQL запит, до того моменту, як ми побачили дані. Всередині Microsoft SQL Server працює механізм, робота якого забезпечується кількома “рушіями”, кожен з яких відповідає за певну частину роботи[11].

Обробка запитів SQL у Microsoft SQL Server виконується рушієм, який має назву Relational Engine. Relational Engine включає в себе декілька етапів обробки запиту SQL. Виділимо 3 основні етапи.

Перший називається Query Parsing. Складається з двох основних частин. Перша - це парсинг. В цій частині виконуються такі дії:

1. Читання та розбір тексту запиту SQL;
2. Генерування хешу за текстом;

3. Перевірка кешу планів та пошук відповідного плану в кеші. Тобто перевірка, чи був цей план вже сформований. Якщо так, то можна взяти план з кешу;
4. Синтаксичний аналіз.

Друга частина роботи Query Parsing - це Algebrizer. На даному етапі перевіряється, чи існують об'єкти бази даних, стовпці у таблицях, які вказуються у запиті.

Наступним етапом є Query Optimization. Основною функцією оптимізатору запитів є побудова плану його виконання. Цей процес оптимізації включає кілька фаз, зокрема:

1. Simplification. На даному етапі Query Optimization спрощує дерево запиту.
2. Trivial Plan Optimization. Виконується пошук тривіального плану.
3. Full Optimization: Search 0. На даному етапі оптимізатор намагається знайти найпростіший та найкращий, витративши мінімум часу.
4. Full Optimization: Search 1. Тут Query Optimization додає деякі правила перетворення та можливі перестановки варіантів з'єднання даних.
5. Full Optimization: Search 2. На даному етапі у будь-якому разі буде знайдено той чи інший план виконання запиту.

Останнім етапом обробки запиту за допомогою Relational Engine є Query Execution. В результаті етапу Query Optimization в нас є готовий план виконання запиту. Query Execution призначений для того, щоб цей план реалізувати.

Підсумувати всю вище викладену інформацію, можна, розглянувши рисунок 1.7.

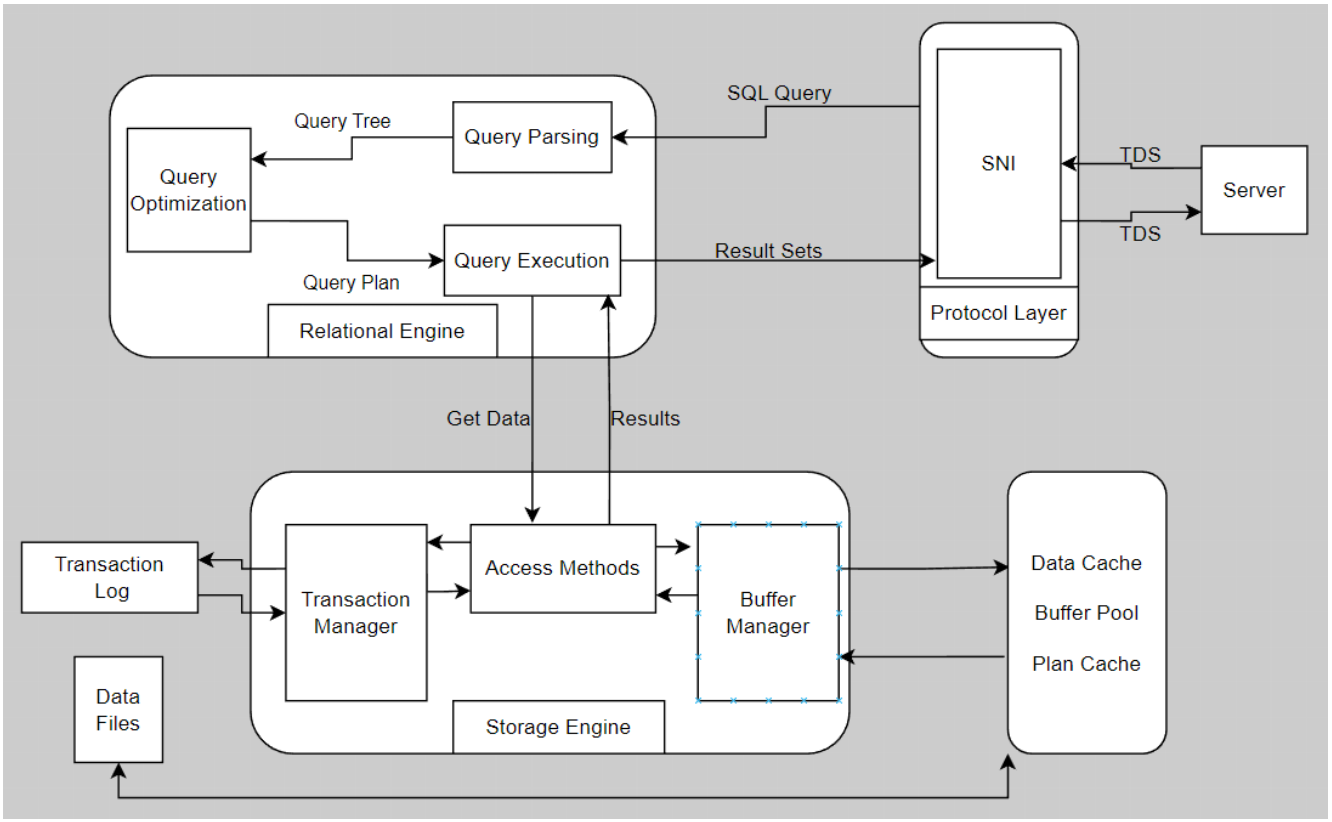

Рисунок 1.7 - Візуалізація архітектури MS SQL Server

1.6 Transact-SQL

Transact-SQL — це набір розширень до мови SQL, які використовує СУБД Microsoft SQL Server[12]. Серед цього набору є обробка помилок, винятків та рядків, оголошені змінні, контроль транзакцій. Синтаксис T-SQL відрізняється від інших мов SQL, але має таку саму функціональність та видає схожі результати з іншими мовами.

Розглянемо основні особливості Transact-SQL. Зручна аутентифікація за допомогою Microsoft Windows. T-SQL включає в себе інструкцію BULK INSERT. Це дозволяє користувачам імпортувати файл у таблицю бази даних або

переглядати у форматі, який визначає користувач. Також серед переваг є можливість підтримки різних функцій для обробки рядків. Серед таких функцій:

1. Агрегатні функції. Ці функції працюють з набором значень та повертають одне значення.
2. Функція ранжирування: повертає значення ранжирування для кожного рядка розділення.
3. Функції набору рядків. Ця функція повертає об’єкт.
4. Скалярні функції. Працюють з одним значенням та повертають одне значення.

1.6.1 Типи даних в Transact-SQL

Розглянемо основні типи даних, що використовує Transact-SQL. Розіб’ємо їх на наступні групи:

1. Числові типи даних.
2. Типи даних, які представляють дату та час. Приклади формату даних - рррр-мм-дд - 2022-12-11 дд/мм/рррр - 11.12.2022
3. Символьні типи даних(VARCHAR, CHAR)
4. Юнікод(NVARCHAR, NCHAR). Відрізняється від стандартних символьних типів даних, тим що NVARCHAR може зберігати будь-які дані Unicode. Стовпець varchar обмежений 8-бітною кодовою сторінкою.
5. Бінарні типи даних.
6. Інші типи даних. Виділимо найбільш популярні з них:
    - UNIQUEIDENTIFIER. Унікальний ідентифікатор GUID. Займає 16 байт;
    - GEOGRAPHY. Зберігає географічні дані, такі як широта та довгота;

- GEOMETRY. Зберігає координати місцезнаходження на площині;
- TABLE. Подає визначення таблиці.

На рисунку 1.8 можна розглянути всі найпопулярніші типи даних у Transact-SQL.

Рисунок 1.8 - Основні типи даних у T-SQL

1.7 ASP.NET Core та .NET

ASP.NET Core являється технологію для створення веб-додатків на платформі .NET, яку створила та розвиває компанія Microsoft. Мови програмування, які використовуються у ASP.NET Core: C# та Visual Basic.NET, J#[13]. На основі цієї платформи можна створювати сайти різної складності і тематики.

Розглянемо основні перевати технології ASP.NET:

1. Інтегроване зручне юніт-тестування;
2. Єдине рішення для створення як веб-інтерфейсу, так і веб-API;

3. Відкритий код;
4. Рушій Razor значно спрощує роботу;
5. Висока продуктивність.

ASP.NET Core є доволі популярною платформою. Є велика кількість популярних сайтів, які написані на даній платформі. Серед цих сайтів можна виділити такі:

1. сайт компанії Microsoft;
2. сервіс для реєстрації доменних імен GoDaddy;
3. StackOverflow - це найбільший онлайн форум;
4. сайт Dell та інші.

На рисунку 1.9 можна розглянути статистику використання різних фреймворків на 2022 рік[14].

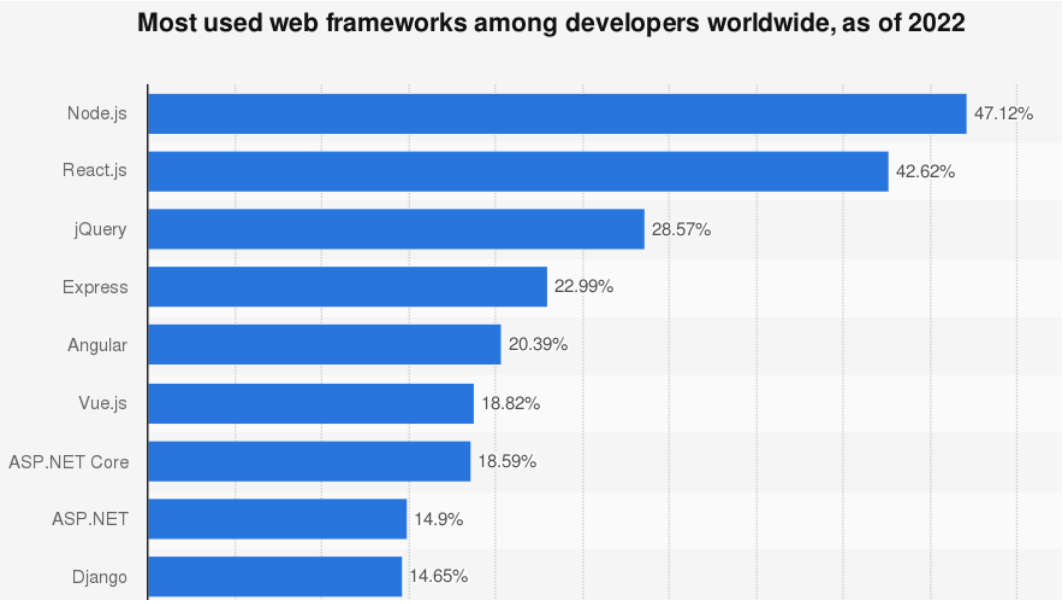

Рисунок 1.9 – Статистика використання різних фреймворків

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЕЛЕКТРОННОГО РОЗКЛАДУ ФАКУЛЬТЕТУ ТА ПРОЕКТУВАННЯ СХЕМИ БАЗИ ДАНИХ

2.1 Аналіз предметної області

Першим етапом розробки будь-якого проекту в якій-якій сфері являється аналіз предметної області цієї сфери. Спочатку проаналізуємо вже існуючі дані та впорядкуємо їх. Електронний розклад факультету буде включати велику кількість елементів. Необхідно правильно їх впорядкувати та проаналізувати. Після чого побудувати ефективну схему бази даних. Розглянемо множину задач для обробки даних:

- Оновлення даних
- Видалення даних
- Додавання даних
- Вибірка даних

Наступним етапом виділимо множину інформаційних елементів бази даних. Множина інформаційних елементів бази даних буде включати 17 елементів: “Група”, “Кафедра”, “Освітня програма”, “Курс”, “День тижня”, “Парність тижня”, “Номер аудиторії”, “Номер пари”, “Пара”, “Корпус”, “Підгрупа”, “Форма навчання”, “Тип заняття”, “Викладач”, “Аудиторні години”, “Неаудиторні години”, “Коментар”.

2.1.1 Проектування схеми бази даних

Для початку розподілимо множину інформаційних елементів, визначених вище між різними таблицями та виділимо властивості об'єктів:

1. Кафедра - базовий підрозділ закладу вищої освіти:
    - назва підрозділу.
2. День тижня.
3. Парність тижня. При створенні розкладу факультету є можливість розділити розклад на парні та непарні тижні.
4. Підгрупа. Пара може бути створена для першої, другої підгрупи або всієї групи.
5. Викладач - людина, людина, яка викладає деяку дисципліну. Властивості:
    - ФІО;
    - Вчене звання;
    - Кафедра.
6. Назва освітньої програми - комплекс освітніх компонентів.
7. Корпус. Властивості:
    - Назва корпусу;
    - Адреса;
    - Телефон корпусу.
8. Номер курсу.
9. Номер пари. Включає в себе такі властивості:
    - Номер пари
    - Час початку пари
    - Час закінчення пари
10. Номер аудиторії.
11. Тип заняття. Серед можливих значень:

- Лекція – усний виклад матеріалу з якоїсь теми, розділу навчальної дисципліни, з ціллю засвоєння теоретичного матеріалу.
- Практичне заняття – заняття, на якому розглядаються та вирішуються конкретні завдання для контролю за успішністю учнів.
- Лабораторна робота - заняття, на якому студенти на практиці здобувають навички, вчаться роботі з приладами та інше.

12. Форма навчання. Денна або заочна.
13. Навчальна група. Властивості об’єкта:
    - Назва групи;
    - Назва освітньої програми;
    - Кафедра;
    - Номер курсу.
14. Пара - урок у навчальному закладі. Виділимо властивості об’єкта “Пара”:
    - Назва пари;
    - Тип заняття;
    - Кафедра;
    - Форма навчання;
    - Корпус;
    - Викладач;
    - Планова кількість аудиторних годин;
    - Планова кількість неаудиторних годин.
15. Розклад. Основний об’єкт, який буде в себе включати такі властивості:
    - Група;
    - День тижня;
    - Парність тижня;
    - Номер аудиторії;
    - Пара;
    - Підгрупа;

- Коментар, для додавання додаткової інформації, яка не є обов’язковою.

На основі виділених об’єктів та їх властивостей, візуалізуємо три основні таблиці нашої бази даних. Об’єкт “Пара” на рисунку 2.1, об’єкт “Група” на рисунку 2.2 та “Розклад” – рисунок 2.3. Розглянемо ці три рисунки нижче.

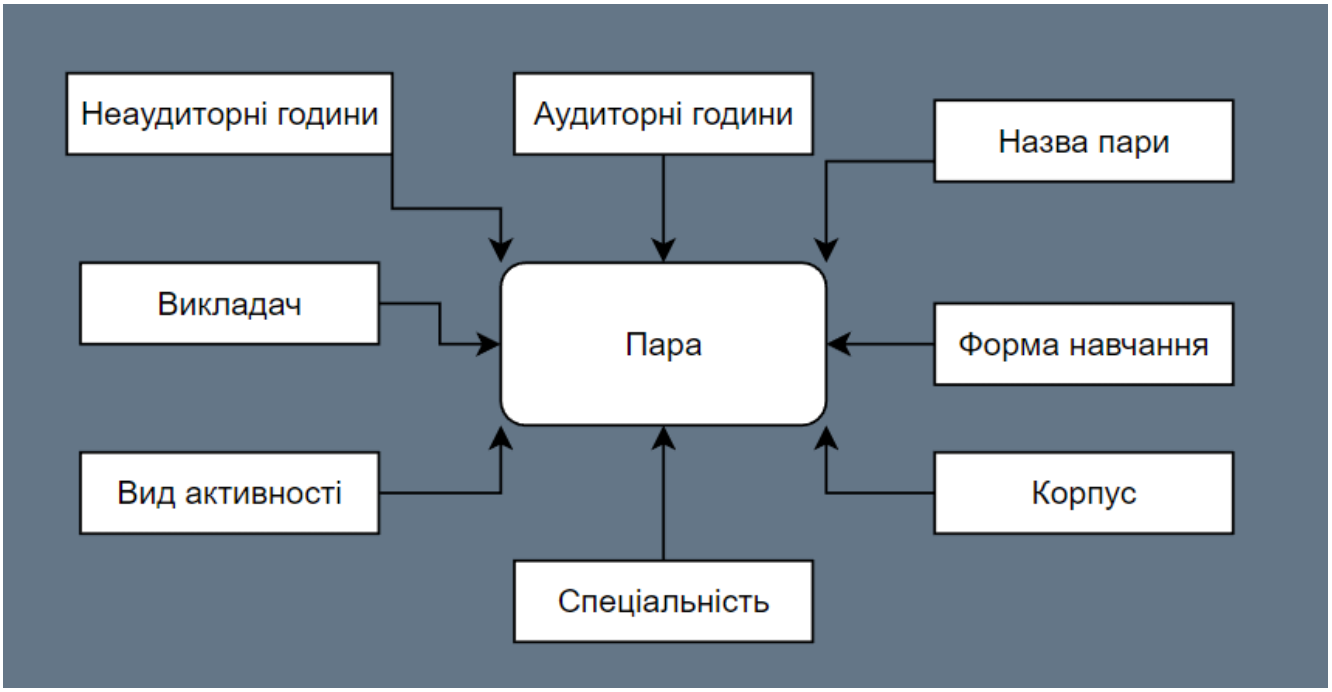

Рисунок 2.1 – Таблиця “Пара”

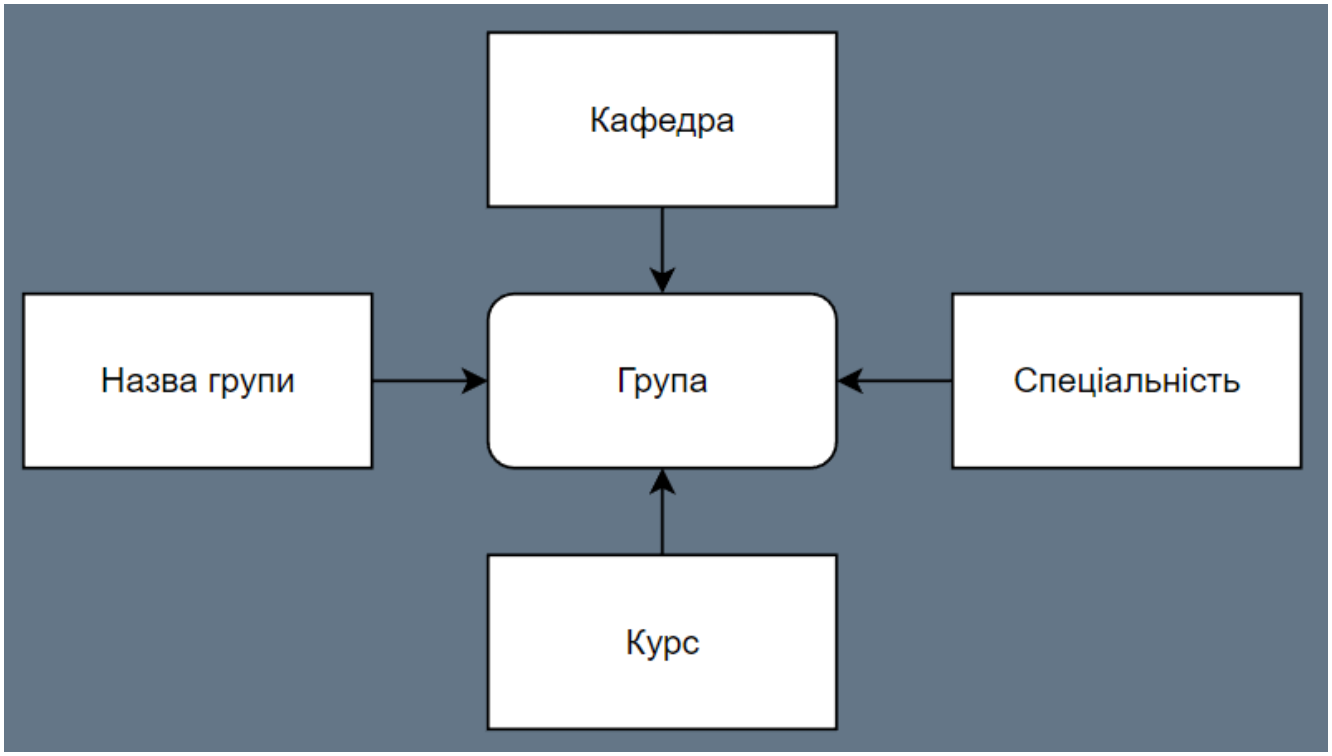

Рисунок 2.2 – Таблиця “Група”

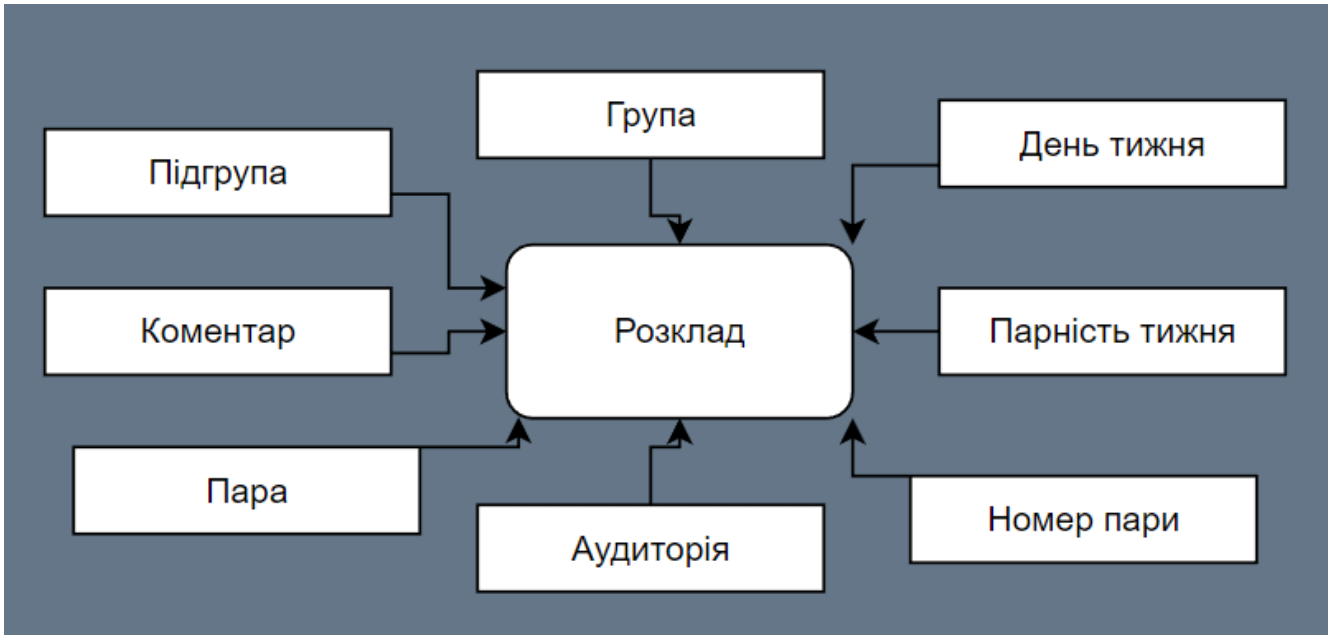

Рисунок 2.3 – Таблиця “Розклад”

Отже, схема бази даних цієї дипломної роботи буде складатися з 15 таблиць, які ми назвемо: Program, DayOfWeek, Parity, Subgroup, Teacher, AcademicBuilding, Department, Course, PairNumber, Audience, FormOfStudy, Activities, TimeTable, Classes, Pair. В кожній схемі буде змінна з назвою Id, яка буде визначати первинний ключ. Візуалізуємо фінальну схему нашої бази даних на рисунку 2.4.

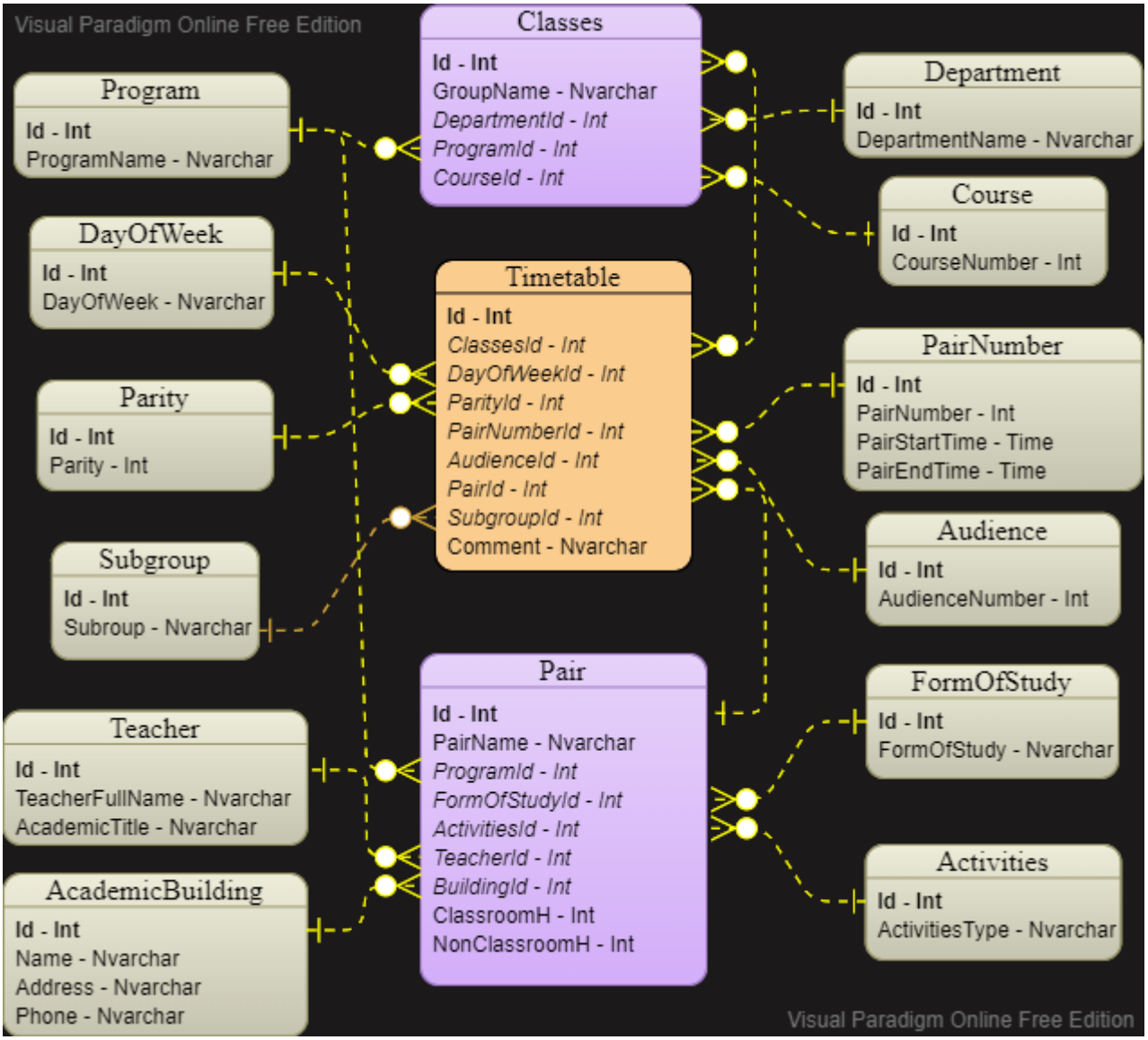

Рисунок 2.4 – Схема БД

В даній схемі ми користувалися також зовнішніми ключами. В таблиці Pair: ProgramId, FormOfStudyId, ActivitiesId, TeacherId, BuildingId. В таблиці Classes:

DepartmentId, ProgramId, CourseId. В таблиці TimeTable: ClassesId, DayOfWeekId, ParityId, PairNumberId, AudienceId, PairId, SubgroupId.

Всі зовнішні ключи будуть зв’язані з первинними відповідних таблиць зв’язками один до багатьох. Про різновиди відношень та різницю між ними між таблицями буде сказано нижче.

Запропонований в цьому підрозділі підхід дозволяє впорядкувати множину інформаційних елементів за рівнями ієрархії. Це дозволяє виділити основну таблицю та таблиці, які займають проміжне положення.

2.1.2 Приклад створення однієї з таблиць у СУБД

Розглянемо принцип створення таблиці в базі даних в Transact-SQL на прикладі створення таблиці Pair із даної роботи. Спочатку створюється база даних за допомогою коду:

```
Create Database {Назва бази даних} Go
```

Далі створюємо таблицю в базі даних. Це виконується за допомогою коду:

```
CREATE TABLE Pair(
        Id int Primary Key,
        PairName nvarchar(50) NOT NULL,
        formOfStudyId int NOT NULL,
        programsId int NULL,
        teacherId int NOT NULL,
        activityId int NOT NULL,
        ClassroomH int NOT NULL,
        NonClassroomH int NOT NULL,

) GO
```

Наповнюємо нашу таблицю Pair деякими даними

```
INSERT Pair
(PairName, formOfStudyId, programsId, teacherId, activityId, ClassroomH, NonClassroomH)
VALUES
(‘Вища математика’, 2, 1, 1, 1, 30, 40),
('Література', 1, 1, 12, 2, 10, 25),
('Англійська', 2, 3, 3, 3, 15, 35),
GO
```

Розглянемо результати роботи коду вище на рисунку 2.5.

Results Messages

	ФІО	Назва пари	Форма	Вид роботи	ProgramName	Неаудиторні год.	К-сть ауд. годин
1	Махович О. І.	Вища математика	Заочна	Лекція	МІТ	40	30
2	Дахно Н. Б.	Література	Денна	Практика	МІТ	25	10
3	Кравченко Ю. В.	Англійська	Заочна	Лабораторна	ІПЗ	35	15

Рисунок 2.5 – Результат створення таблиці в T-SQL

Результат був отриманий за допомогою оператору SELECT і даного коду:

```
SELECT Teacher.FullName as 'ФІО', PairName as 'Назва пари',
FormOfStudy.FormOfStudy as 'Форма', Activity.ActivityType as 'Вид роботи',
Programs.ProgramName, Pair.NonClassroomH as 'Неаудиторні год.', Pair.ClassroomH as 'К-сть ауд. годин'
FROM Pair
LEFT JOIN Teacher on Teacher.Id = Pair.TeacherId
LEFT JOIN Programs on Programs.Id = Pair.programsId
LEFT JOIN FormOfStudy on FormOfStudy.Id = Pair.FormOfStudyId
LEFT JOIN Activity on Activity.Id = Pair.activityId
```

2.1.3 Проектування схеми бази даних для користувачів

Передбачається, що даний додаток будуть відвідувати три категорії користувачів: адміністратор, викладач та студент.

Серед повноважень адміністратора мають бути можливості для додавання, редагування та видалення даних про заняття, викладачів, групи та пари.

Основною можливістю для студентів та викладачів має бути перегляд розкладу у зручній для них формі.

На основі цих вимог спроектуємо схему бази даних, яка буде зберігати користувачів. Також створимо розподілення на ролі. В таблиці “Ролі” додамо 3 рядка: student, teacher та admin. Таблиці будуть з’єднані зв’язком один до багатьох, оскільки багато користувачів можуть мати одну роль.

В свою чергу таблиця бази даних «користувачі» буде включати в себе: ім’я, нікнейм, пошту, пароль та роль. Розглянемо результат проектування схеми бази даних користувачів на рисунку 2.6.

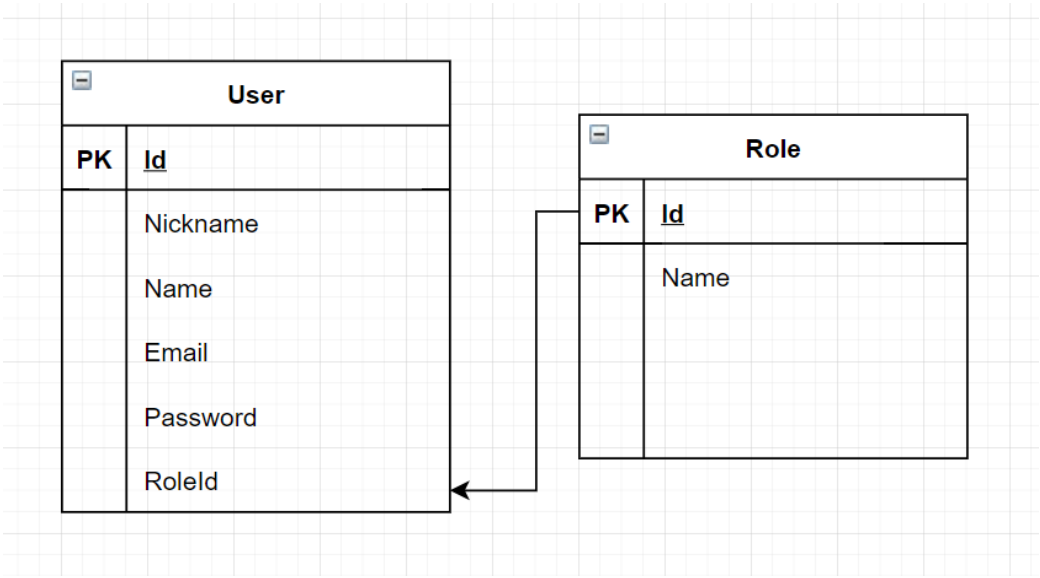

Рисунок 2.6 – Схема бази даних користувачів

Запропонована схема є дуже зручною, оскільки дає можливість легко масштабувати ролі та елементи користувачів.

2.2 Архітектура та моделі розробки ASP.NET Core. Модель MVC

Оскільки в минулому розділі було вирішено використовувати технологію ASP.NET Core, то буде доречно розглянути її архітектуру та модель розробки.

Розглянемо архітектуру ASP.NET Core. На верхньому рівні ASP.NET розташовуються моделі для взаємодії з користувачем. Серед них моделі для побудови користувацького інтерфейсу та обробки введення інформації. Серед основних це - SPA(Single Page Application), MVC, Razor Pages та Blazor. Крім цього, на цьому рівні розташовуються послуги, такі як HTTP API, бібліотеки SignalR або сервісів GRPC.

Ці технології допомагають взаємодіяти із ASP.NET Core. В свою чергу сам ASP.NET Core представлений насамперед middleware – програмним забезпеченням, яке використовується для обробки запиту. Технології вищого рівня також можуть взаємодіяти з розширеннями та додатками. Вони не є частиною ASP.NET Core, але може додаватися для аутентифікації, конфігурації та т. ін.

На нижньому рівні ASP.NET Core розташовуються серверні елементи. Серед них такі: IIS, Kestrel, бібліотеки HTTP.sys.

Архітектуру платформи ASP.NET Core можна виразити на рисунку 2.7[15].

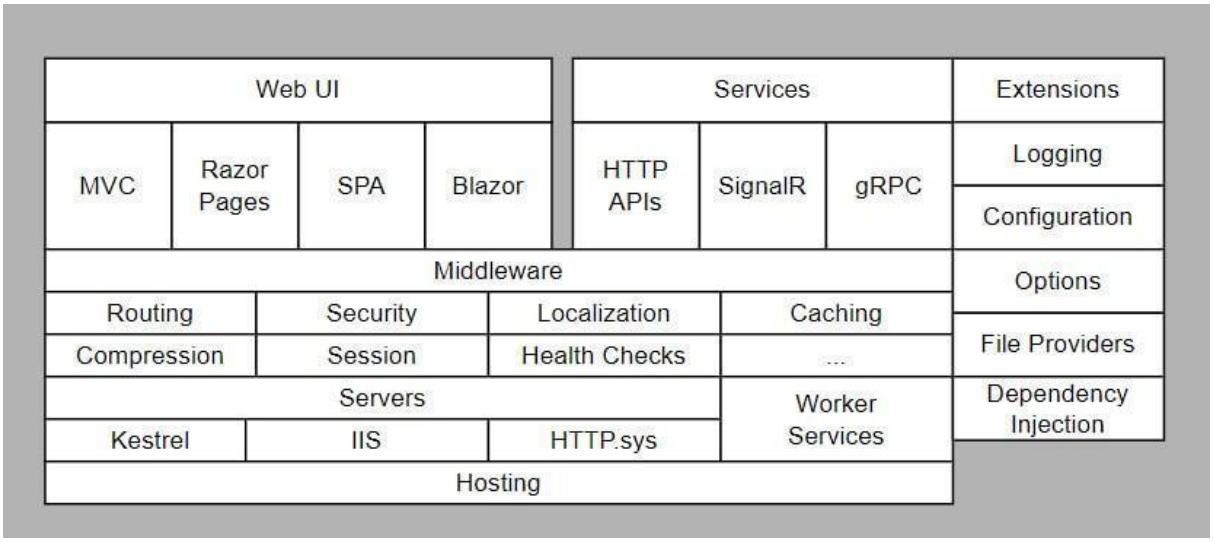

Рисунок 2.7 - Архітектура ASP.NET Core

2.2.1 Схема Model-View-Controller

Model-View-Controller - це найпопулярніша схема розробки додатку в ASP.NET Core. ASP.NET Core MVC будує програму навколо трьох основних компонентів: це Model (модель), View (вид) та Controller (контролер). Кожний компонент в MVC відповідає за певні дії. Клієнт виконує певну дію, посилаючи запит на контролер. Контролер трактує дію клієнта, та, за необхідності, відправляє запит на модель для зчитування або зміни інформації там. Вид же являється кодом Razor, визначає візуальну складову. Розглянемо принцип роботи MVC на рисунку 2.8.

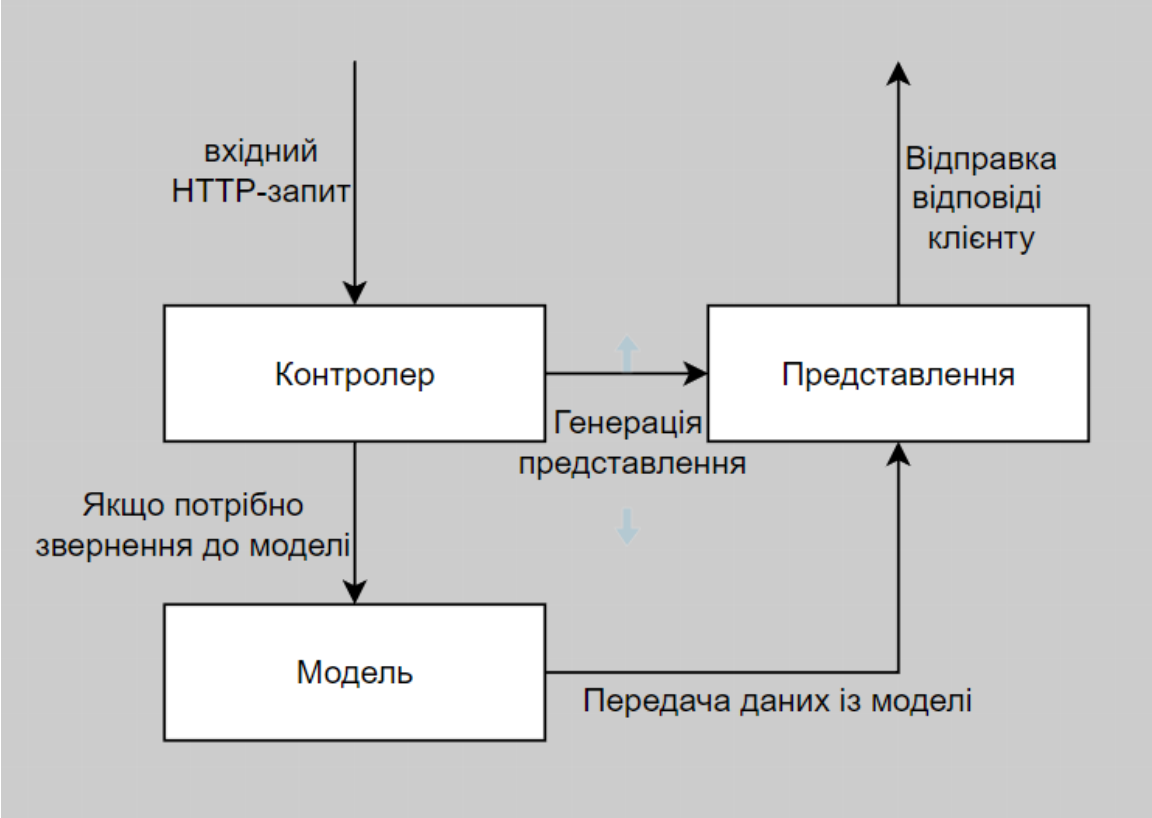

Рисунок 2.8 – Модель Model-View-Controller

У моделі MVC є свої переваги. Серед основних: розробку додатку можна розбити на етапи. Через це тестування коду стає простішим. Серед переваг також можна виділити легкість масштабування. За рахунок цього модель MVC є дуже популярною і використовується в багатьох проектах не тільки на платформі ASP.NET Core, але і в PHP, Java, JavaScript та інших популярних мовах програмування.

2.3 Відношення між таблицями у базі даних

Існує 3 основні види зв'язків між таблицями[16]:

1. Багато до багатьох;
2. Один до багатьох. Цей тип зв’язку ділиться на:

- з обов’язковим зв’язком;
- з необов’язковим зв’язком.

3. Один до одного. Також ділиться на з обов’язковим та обов’язком зв’язками.

Розглянемо докладно кожен з них.

Зв'язок багато до багатьох. Розглянемо приклад. Нам потрібно створити базу даних, яка буде зберігати пари, які може вести один викладач. Викладач може вести декілька пар. Але і одну пару може вести декілька викладачів.

Викладачів представляє таблиця "Teacher" (id, ім'я, наукова ступінь). Для пар створимо таблицю “Pair”(id, назва пари). Отже, ці таблиці пов’язані між собою зв’язком багато до багатьох. Ми вже маємо дві таблиці. Зараз, для того щоб побудувати зв’язок один до багатьох, треба створити ще одну таблицю посередника. Таблиця, яка буде складатися с TeacherId та PairId. Ця таблиця-посередник показує пару та викладача наступним способом в таблиці 2.1:

Таблиця 2.1 - Реалізація схеми багато до багатьох

PairId	TeacherId
1	2
1	3
2	1
1	1

В таблиці видно, що кожному викладачеві протиставлені пари, які він може вести. А кожній парі - викладачі, які її ведуть. Тобто на таблицю можно дивитися з двох сторін.

Розглянемо реалізацію таблиці у вигляді коду на Transact-SQL.

```
create table Teacher (
TeacherId int primary key,
FullName nvarchar(128) not null,
AcademicTitle nvarchar(128) not null
)
create table Pair
(
	PairId int primary key,
	PairName nvarchar(64) not null
)
create table TeacherPair
(
	TeacherId int foreign key references Teacher(TeacherId),
	PairId int foreign key references Pair(PairId),
	primary key(TeacherId, PairId)
)
```

На рисунку 2.9 можна побачити реалізацію даної схеми.

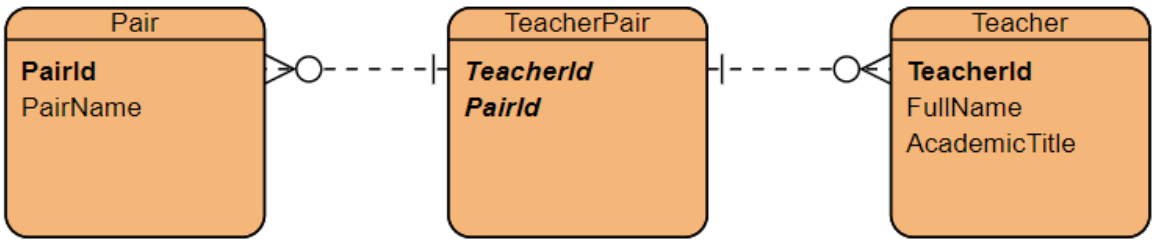

Рисунок 2.9 - Приклад схеми багато до багатьох

Зв’язок один до багатьох. Являється найпопулярнішим зв’язком. Розглянемо дану схему на прикладі бази даних із даної роботи. Нам потрібно створити базу даних, яка буде зберігати користувачів та дані про них. Користувач має: ім’я, нікнейм, пошта, пароль та ролі. В таблиці “Role” зберігаються ролі. При цьому одна роль може бути у багатьох користувачів. Але у одного користувача може бути лише одна роль. Як можна побачити, це зв’язок один до багатьох. Реалізація даної схеми була розглянута у розділі 2.1.3 даної роботи.

Розглянемо код який ми використовували для реалізації даних таблиць у MS SQL Server.

```
create table Users
(
    Id int primary key,
    Email nvarchar(64) not null,
    Password nvarchar(64) not null,
    NickName nvarchar(64) not null,
    Name nvarchar(64) not null,
    RoleId int foreign key references Role(Id)
```

```
)
create table Roles
(
    Id int primary key,
    Name nvarchar(64) not null
)
```

На рисунку 2.10 можна побачити виведення на екран даної таблиці.

Results | Messages

	Id	Email	Password	RoleId	Name	NickName
1	9	imper1uss666@gmail.com	123456	1	Anatoliy	Solution
2	10	user@gmail.com	123456	2	User	User123
3	11	teacher@gmail.com	123456	3	Teacher	Teacher123

Рисунок 2.10 - Результат таблиці “Users”

Зв’язок один до одного. Розглянемо приклад. У деякого університету є база даних розкладу. І у цього університету є тільки одна будівля. Потім у нього з’являється другий корпус. І база даних розкладу треба в кожну пару додати корпус в якому будуть проходити пари. Звичайно, можна просто створити новий стовпець з Id корпусу, в якому буде проходити пара. Але пар дуже багато і це може зайняти дуже велику кількість часу оскільки в кожну пару треба буде вручну вписувати Id корпусу.

Простіше в даній ситуації створити нову таблицю AcademicBuilding. І в цю таблицю додати тільки ті пари, які будуть проходити в новому корпусі. Але по помилці можна додати в цю таблицю одну й ту ж пару декілька разів. Для цього зробимо PairId в таблиці AcademicBuilding унікальним. Це можна зробити за допомогою обмеження unique. Розглянемо на прикладі.

```
create table Pair
(
    PairId int primary key,
    Name nvarchar(128) not null,
    Group nvarchar(128) not null
)

create table AcademicBuilding
(
    Id int primary key,
    PairId int unique foreign key references Pair(PairId)
)
```

В таблиці 2.2 можна побачити результат роботи даного коду.

Таблиця 2.2 - Реалізація схеми один до одного

Id	PairId
1	123
2	654
3	876

Як можна побачити, даний варіант значно простіший за ручне вписування Id будівлі в кожну пару.

2.4 Структура проекту

Однією з найважливіших частин розробки для будь-якого проекту є планування зручної структури. Це дозволяє заощадити ресурси, такі як час в майбутньому. В ASP.NET Core MVC існує два основних підходи до структури проекту. Розглянемо кожен з них.

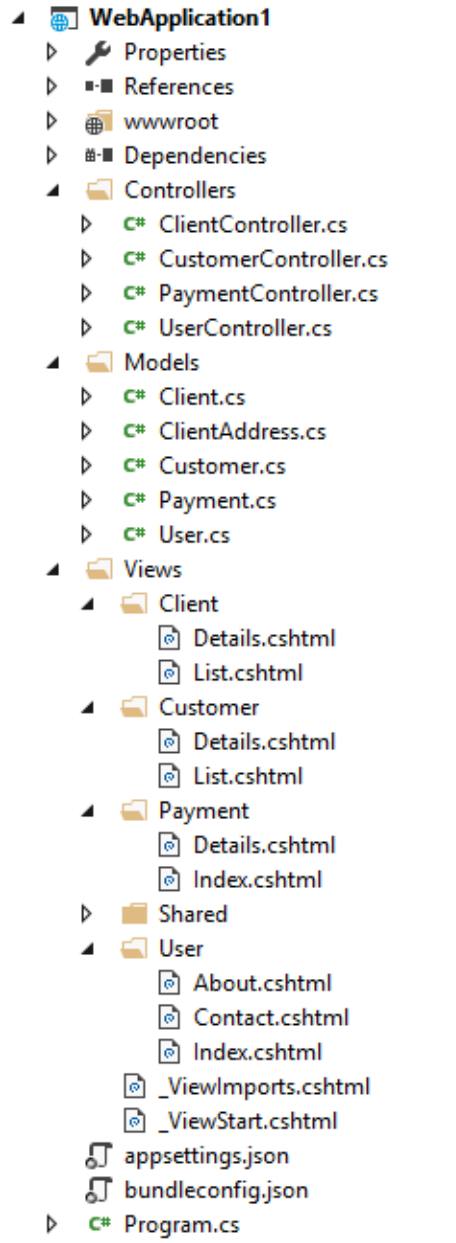

Рисунок 2.11 - Feature Folder

Перший варіант підходу називається Feature Folders. Такий варіант організації структури папок, особливо для великих проектів, останнім часом стає дуже популярним. Особливо актуально це для команд, які використовують підхід Vertical slice.

Під час організації проекту за допомогою Feature Folders, ви створюєте кореневу папку, а далі створюються вкладені папки для кожної моделі, контролера, інтерфейсу та репозиторію. У середньому, ми отримуємо від 5 до 10 файлів в кожній папці. І всі ці файли тісно пов’язані між собою. Серед переваг такого підходу є:

1. легке масштабування проекту;
2. зручність при великій кількості файлів та розробці великою командою.
3. збільшується зв’язність файлів;
4. дозволяє тримати меньше папок відкритими в Solution Explorer.

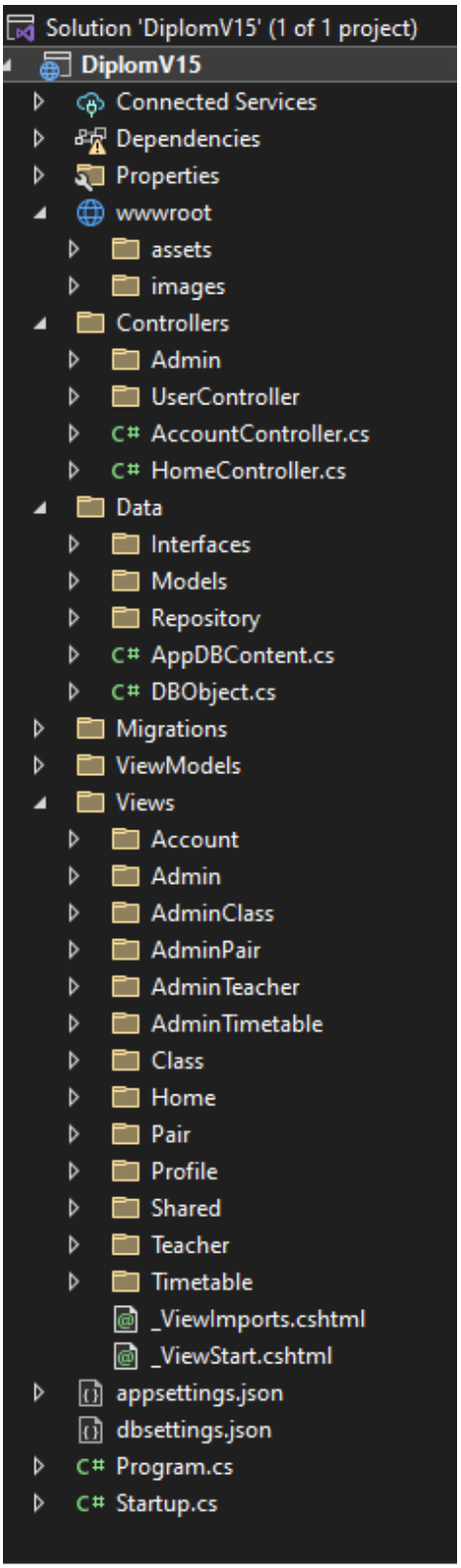

Рисунок 2.12 - Tech Folder

Також серед популярних підходів є Feature Folders, де для кожної моделі, контролера, інтерфейсу та репозиторію створюється своя папка. Це підвищує зв'язність файлів. Також це дає легко масштабувати проект, оскільки всі файли, зв'язані з одною моделлю, знаходяться в одній папці [17]. Розглянемо приклад організації Feature Folders на рисунку 2.11.

Другий варіант називається Tech Folder. Цей підхід реалізовано в ASP.NET за замовчуванням при створенні проекту. Всі моделі, інтерфейси, репозиторії, представлення та контролери розподілені кожен в свою папку. Розглянемо основні переваги підходу Tech Folder:

1. дуже проста структура, особливо якщо ви вже працювали з проектами на ASP.NET MVC, ви відразу зорієнтуєтеся у проекті;
2. логічна організація;
3. зручність, якщо вам потрібно шукати, наприклад контролер, то ви відразу знаєте де почати шукати.

При створенні нового проекту, Tech folders показує себе собі непогано, поки не немає великої кількості функціоналу та файлів. Коли проект починає зростати, в кожній із папок становиться занадто велика кількість файлів. Стає важко орієнтуватися та шукати необхідний код. працює

досить добре, поки не великий функціонал і немає багато файлів.

При аналізі та порівнянні цих двох підходів ми прийшли до висновку використовувати підхід Tech Folder. Основними критеріями були простота та зручність при роботі з невеликими проектами.

Розглянемо на рисунку 2.12 структуру нашого проекту.

Перша папка - wwwroot. В ній зберігаються всі статичні файли - CSS, JS, SASS, файли шрифтів та зображення. Для кожного типу файлів створена своя папка.

В папці Controllers знаходяться всі контролери. Контролери розподілені на адміністративні контролери для керування даними та користувацькі. Знизу знаходяться контролери для роботи з неавторизованими користувачами.

В папці з назвою Data знаходяться всі інтерфейси, моделі та репозиторії. Кожна частина знаходиться в своїй папці. Нижче розташований контекст нашої бази даних.

В папці Migrations знаходяться всі міграції. ViewModels зберігає моделі представлень.

В папці Views зберігаються наші представлення. Кожна папка відповідає контролеру, до якого вона відноситься. Файли з назвами dbsettings.json та appsettings.json відповідають за підключення до нашої бази даних

2.5 Технологія Entity Framework

Entity Framework являється ORM(Object-Relational Mapping)-технологією на базі фреймворка .NET для доступу до даними. EF представляє значно вищий рівень абстракції порівняно зі стандартними засобами ADO.NET. Entity Framework дозволяє працювати з даними із бази незалежно від самої СУБД. Це

дозволяє працювати не з таблицями, ключами та індексами, як, наприклад при роботі з ADO.NET а з сутностями.[18]

Все це допомагає швидкості та простоті роботи з базами даних. По сути, більшу частину роботи, таку як підключення або запити, Entity Framework робить сам.

Розглянемо одне з ключових понять у Entity Framework - Entity Data Model(EDM). Вона зіставляє сутності із EF із таблицями із баз даних.

На рисунку 2.13 можна розглянути три основні частини Entity Data Model[19].

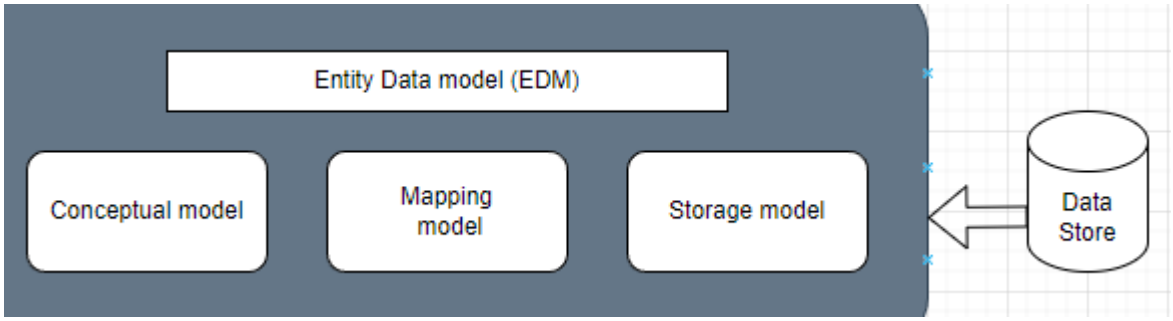

Рисунок 2.13 - Entity Data Model

Розглянемо ці три частини EDM більш докладно:

1. Концептуальний рівень. Ще називають рівнем визначеності концептуальної схеми. На цьому рівні визначаються класи сутностей, які використовуються в додатку;
2. Модель схеми зберігання. Ця частина являється схематичним представленням внутрішнього сховища даних. Визначає типи даних, таблиці, відносини та стовпці, які використовуються в базі даних;
3. Рівень відображення. Займає проміжне положення, являється відображенням між концептуальною рівнем та моделлю схеми зберігання.

Саме ці три частини допомагають нам працювати з базами даних через класи у Entity Framework.

Entity Framework Fluent API використовується для настройки класів [20]. EF Fluent API використовує шаблон Fluent API для формування результату шляхом з'єднання методів. Розглянемо основні можливості даної технології. В ній можна настроїти сутності, а також типи відношень між ними. Є можливість додати зовнішні та первинні ключи. Настроїти властивості та модель.

В заключення можна сказати, що Entity Framework є дуже зручною технологією. Вона дає велику кількість функціональних можливостей, дозволяє зменшити кількість коду та збільшити його простоту та читабельність. Але серед недоліків треба виділити те, що Entity Framework погіршує оптимізацію системи та збільшує навантаження на систему[21].

2.6 Рушій Razor

Razor – це синтаксис розмітки для вбудовування серверного коду у веб-сторінки [22].

При розробці рушія представлень Razor розробники виділили деякі основні цілі:

1. Перше - це зробити компактний, гнучкий код. При порівнянні Razor з, наприклад, чистим HTML, можна побачити, що коду значно менше.
2. Простий у засвоєнні. При написанні коду Razor можна використовувати навички та знання мови HTML. Основи Razor можливо вивчити за мінімальний термін.
3. Razor не являється новою імперативною мовою. Розробники можуть використовувати знання C# або VB.

4. Для роботи з кодом Razor не потрібен певний інструмент. Для нього достатньо будь якого текстового редактору.
5. Зручна інтеграція у Visual Studio та підтримка Intellisense. Також Intellisense працює в Visual Web Developer.
6. Зручна реалізація юніт-тестування. Юніт-тести можна проводити навіть без веб-сервера або контролера.

3 РОЗРОБКА ВЕБ-ДОДАТКУ «ЕЛЕКТРОННИЙ РОЗКЛАД ФАКУЛЬТЕТУ» НА ПЛАТФОРМІ .NET

3.1 Створення моделей.

В даному розділі розглянемо сам процес розробки та написання коду до нашого веб-додатку “Електронний розклад факультету”. Оскільки в минулих розділах було вирішено розробляти наш додаток за допомогою схеми Model-View-Controller, першим етапом даного розділу буде написання моделей до нашої бази даних.

3.1.1 Способи взаємодії з базою даних у Entity Framework

В минулому розділі були вирішено використовувати Entity Framework для доступу к даним. В EF існує три способи взаємодії з базою даних. Розглянемо їх та виберемо найкращий.

Перший називається Database First. Цей варіант використовується в ситуаціях, коли база даних вже існує. Цей варіант був доданий в EF найпершим, і існує з моменту появи цієї технології. При роботі з Database First ми першим пунктом створюємо базу даних.

Для роботи з підходом Database First нам необхідно для початку додати в Solution Explorer файл з типом ADO.NET Entity Data Model. Потім, у вікні вибрати варіант Generate from database, після цього пройти кроки для підключення до необхідної бази даних. Solution Explorer має з’явитися файл з типом edmx. Нижче розглянемо код, необхідний для отримання даних з нашої бази:

```
using(userstoredbEntities db = new userstoredbEntities())
{
   var var1 = db.Name;
   foreach (Name var1 in Table1)
      Console.WriteLine("{0}.{1} - {2}", u.Field1, u.Field2, u.Field3);
}
```

Наступним варіантом роботи з базою даних у Entity Framework є підхід Model First. По суті являється протилежністю Database First. Спочатку створюється модель бази даних у коді, а по ній генерується база даних.

По аналогії з Database First для початку необхідно додати файл з типом ADO.NET Entity Data Model. При даному підходу у вікні треба вибрати Empty Model. Після цього за допомогою кнопки Add Entity добавляти необхідні сутності, підключення і т. ін. При цьому в файлі з розширенням .edmx буде відображатися візуалізація нашої бази даних.

Основною проблемою цих двох підходів являється те, що вони, після додавання Code First, по суті, втратили актуальність. Code First - це спосіб підключення до бази даних у Entity Framework, який об’єднує можливості обох підходів: Model First та Database First. Оскільки це найпопулярніший та найактуальніший підхід, то саме його ми використовували у даній роботі.

Для підключення та роботи з базою даних у цьому підході для кожної таблиці у БД створюється так звана модель. Розглянемо принцип створення та роботи з моделями на прикладі моделі до таблиці Pair у даній роботі. На рисунку 3.1 можна побачити першу частину коду для моделі Pair.

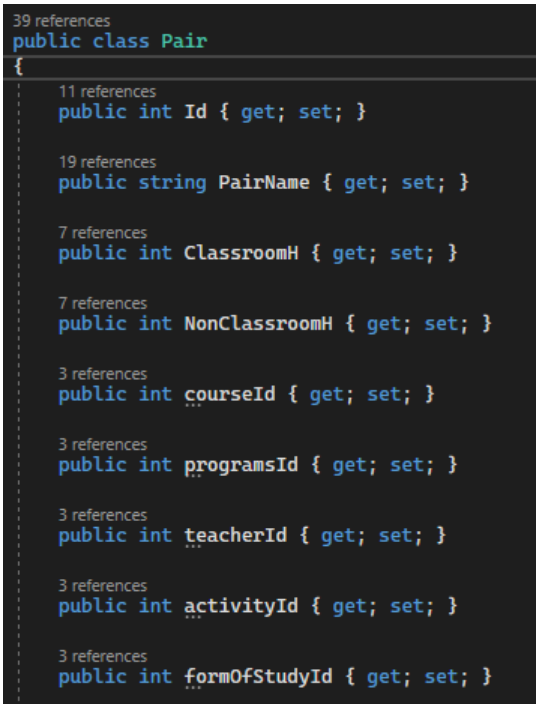

Рисунок 3.1 – Перша частина моделі таблиці Pair

Як можна побачити, при створенні моделі, для кожної рядка у таблиці створюється змінна з аналогічний типом та назвою. Спочатку створюємо змінну для первинного ключа, потім змінні з таблиці, потів змінні, які використовуються як зовнішні ключі. Також для кожної змінної при необхідності додаються гетери і сетери. В ситуаціях коли нам не можна змінювати дані додаються тільки гетери.

Далі на рисунку 3.2 розглянемо другу частину коду в моделі Pair.

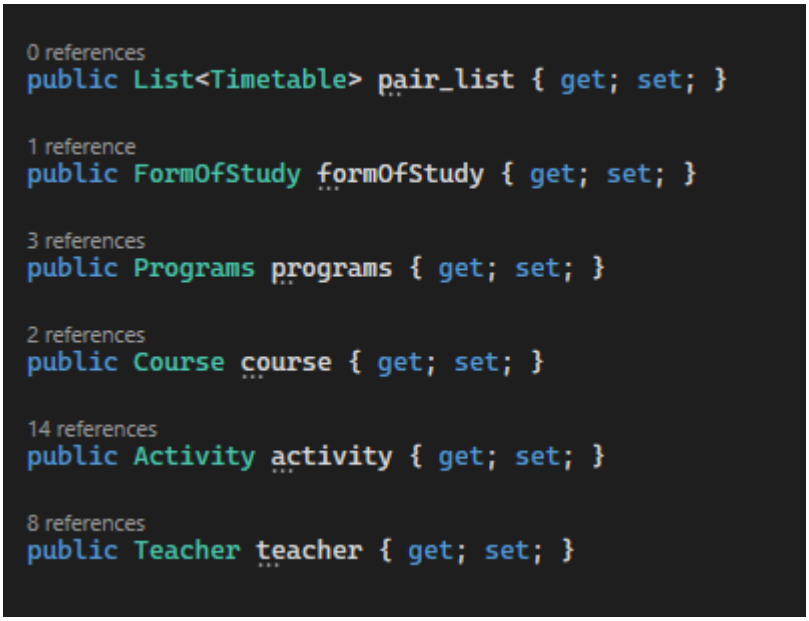

Рисунок 3.2 - Друга частина моделі таблиці Pair

Перша строка коду, яка представляє собою список, слугує для підключення цієї таблиці до іншої, в нашій ситуації до таблиці Timetable. За допомогою наступного коду ми можемо підключити іншу таблицю до цієї:

public {назва таблиці} {назва змінної} (за необхідністю { get; set; })

Розглянутий на даному рисунку код і являється підключення один до багатьох.

Далі розглянемо на рисунку 3.3 контекст нашої бази даних.

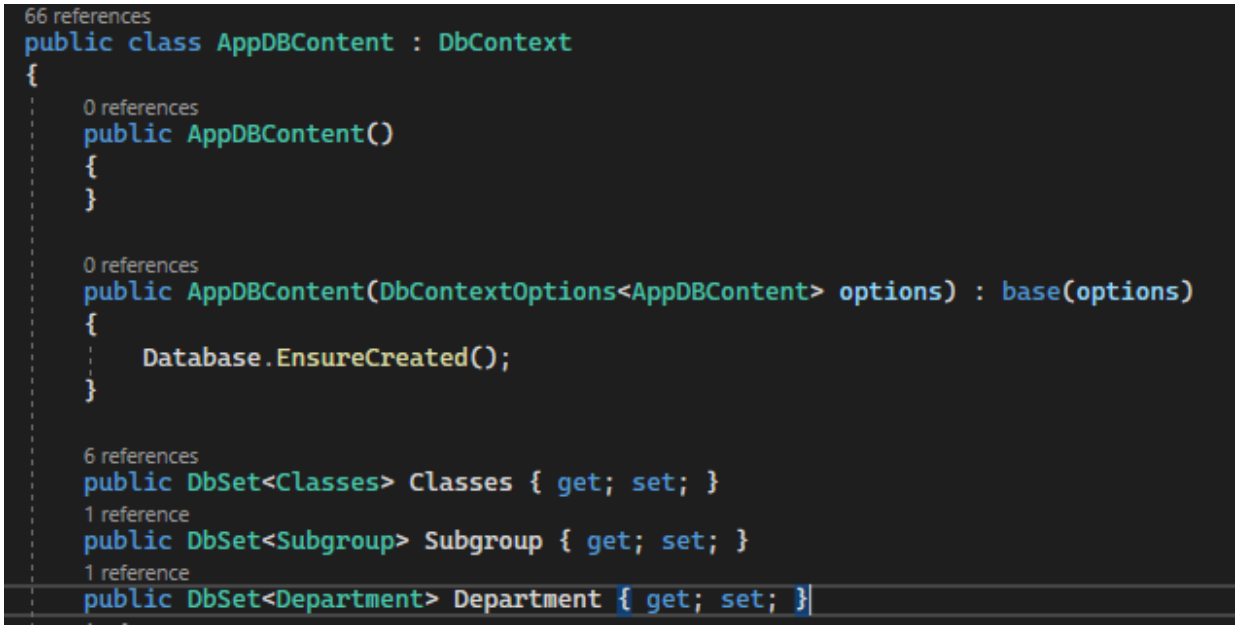

Рисунок 3.3 - Контекст нашої бази даних

Як можна побачити на рисунку, після конструктора для кожної моделі в нашому додатку створюється змінна з типом DbSet. По аналогії з моделями додається, при необхідності, гетер та сетер. Необхідно щоб наш контекст бази даних наслідував клас DbContext - це клас, який знаходиться в просторі імен System.Data.Entity, і представляє собою сеанс для роботи, збереження та зміни єкземплярів сутностей.

Також для того, щоб наші моделі могли підключатися до самої бази даних необхідно написати строку підключення в спеціально створеному для цього файлі dbsettings з розширенням json. Підключення:

"DefaultConnection":"Server={Назва серверу(при локальному серверу - крапка)}; Database={назва бази даних}; Trusted_Connection=True; MultipleActiveResultSets=true"

Також необхідно додати в файл Startup.cs код для додавання підключення за замовчуванням.

```
public Startup(IWebHostEnvironment hostEnv)
        {
                _confString = new
            ConfigurationBuilder().SetBasePath(hostEnv.ContentRootPath).AddJsonFi
            le("dbsettings.json").Build();
        }
```

Linq додає в Entity Framework синтаксис [23], подібний до SQL. Розглянемо Linq на прикладі строки для виводу інформації з таблиці Pair, де ActivityId дорівнює одиниці та розсортованої за Id із контролера PairController із даної роботи.

```
pair = m_IPair.pairs.Where(i => i.activity.Id.Equals(1)).OrderBy(i => i.Id);
```

Також розглянемо отримання інформації про розклад з бази даних за допомогою Linq на рисунку 3.4.

```
4 references
public Timetable GetPair(int id) => appDBContent.Timetable.FirstOrDefault(p => p.Id == id);
```

Рисунок 3.4 – Отримання пари за Id за допомогою Linq

3.1.2 Валідація

Важливу роль при розробці моделей в ASP.NET Core MVC грає валідація даних. Вона дозволяє перевірити вхідні дані, щоб вони не включили в себе некоректні або неправильні дані[24]. Валідація моделі знаходиться в просторі імен System.ComponentModel.DataAnnotations. Для початку розглянемо валідацію моделі на прикладі валідації моделі Pair із даної роботи на рисунку 3.5.

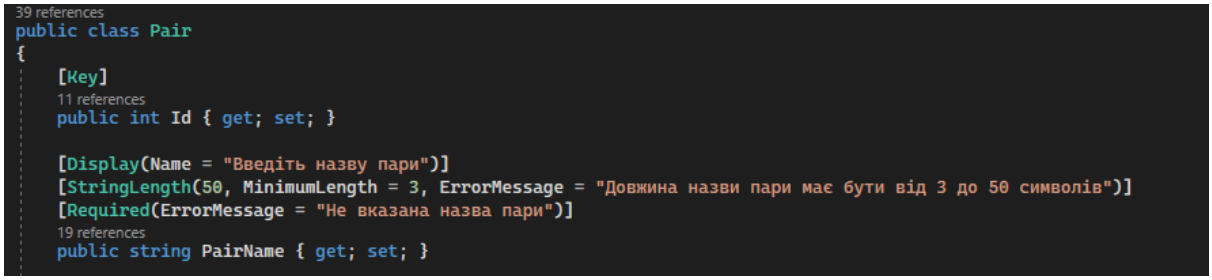

Рисунок 3.5 - Валідація моделі на прикладі Pair

Розглянемо основні положення. Атрибут Key - додає первинний ключ до моделі. Display додає текст, який відображається замість назви змінної. StringLength вказує максимальну та мінімальну довжини поля string. Required вказує на те, що це поле обов’язкове до заповнення.

Розглянемо також інші популярні атрибути валідації моделі. Атрибут RegularExpression вказує на те, що вираз має відповідати вказаному даному атрибуту. Атрибут Range вказує діапазон можливих значень для числових типів даних. Атрибут Compare гарантує, що дві властивості об'єкта будуть мати однакове значення. Наприклад, при підтвердженні паролю.

Валідація на стороні сервера здійснюється за допомогою об’єкту ModelState. Цей об’єкт зберігає всі дані, додані користувачем у модель, і при знаходженні помилки поверне false. Розглянемо валідацію на стороні сервера на прикладі контролера авторизації із даної роботи на рисунку 3.6.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Login(LoginModel model)
{
    if (ModelState.IsValid)
    {
        User user = await dBContent.Users
            .Include(u => u.Role)
            .FirstOrDefaultAsync(u => u.Email == model.Email && u.Password == model.Password);

        if (user != null)
        {
            await Authenticate(user);

            return RedirectToAction("Index", "Home");
        }
        ModelState.AddModelError("", "Некоректний логін та(або) пароль");
    }
    return View(model);
}
```

Рисунок 3.6 - Валідація на стороні сервера

Як можна побачити у коді, спочатку за допомогою ModelState.IsValid перевіряються всі дані додані користувачем у моделі. Якщо помилок немає повертається true і проходить авторизація, в іншому випадку за допомогою ModelState.AddModelError додаток видає помилку.

Також розглянемо tag-хелпери валідації у представленні. Вона виконується за допомогою коду:

3.2 Інтерфейси та репозиторії

На цьому етапі розглянемо реалізацію функціоналу моделей. За цей в даній роботі відповідають папки з назвами Interface та Repository. Для кожної моделі створимо реалізацію функціоналу у даних папках. Інтерфейси служать для визначення ряду методів, а репозиторії для їх реалізації. Отже, для початку розглянемо інтерфейс для моделі Pair у нашій роботі на рисунку 3.7.

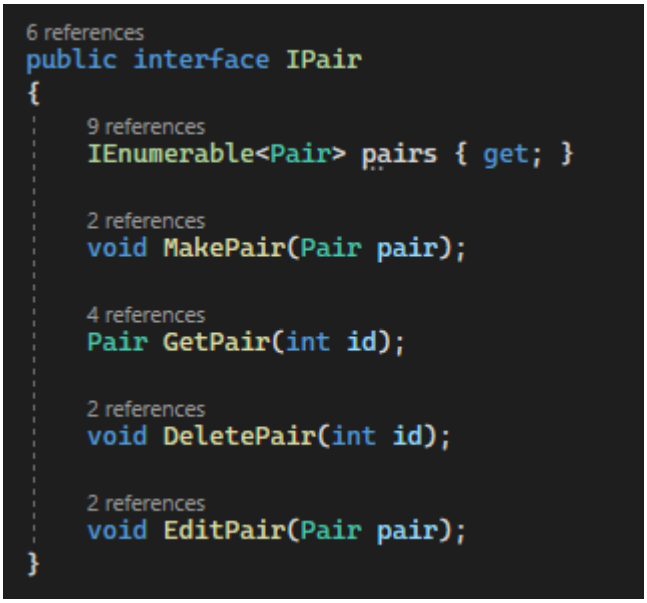

Рисунок 3.7 - Реалізація інтерфейсу Pair

Як можна побачити, в інтерфейсі створено один перелік та чотири методи для роботи з даними із БД. Перелік має тип IEnumerable, який підтримує ітерацію по даним. Він буде зберігати в собі всі необхідні дані із таблиці Pair, при роботі з додатком. Методи GetPair та DeleterPair будуть отримувати та видаляти пари за айді. MakePair буде створювати нове пару, а EditPair її змінювати. Це основна множина задач для обробки даних, яку ми отримали при аналізі предметної області в минулому розділі. Далі розглянемо реалізацію даного інтерфейса на рисунку 3.8.

```
2 references
public class PairRepository : IPair
{
    private readonly AppDBContent appDBContent;

    0 references
    public PairRepository(AppDBContent appDBContent)
    {
        this.appDBContent = appDBContent;
    }

    9 references
    public IEnumerable<Pair> pairs => appDBContent.Pair.Include(c => c.course).Include(c => c.formOfStudy).
        Include(c => c.programs).Include(c => c.teacher).Include(c => c.activity);

    4 references
    public Pair GetPair(int id) => appDBContent.Pair.FirstOrDefault(p => p.Id == id);

    2 references
    public void DeletePair(int id)
    {
        Pair pair = GetPair(id);

        appDBContent.Pair.Remove(pair);

        appDBContent.SaveChanges();
    }

    2 references
    public void MakePair(Pair pair)
    {
        appDBContent.Pair.Add(pair);

        appDBContent.SaveChanges();
    }

    2 references
    public void EditPair(Pair pair)
    {
        appDBContent.Pair.Update(pair);

        appDBContent.SaveChanges();
    }
}
```

Рисунок 3.8 - Реалізації методів із інтерфейсу Pair

Як можна побачити, реалізація методів доволі інтуїтивно зрозуміла, вона включає в себе такі методи як, Remove, Add, Update та SaveChanges. Ці методи включає в себе класс DbContext, який наслідує контекст нашої бази даних.

3.2.1 Життєвий цикл

При розробці додатку на ASP.NET Core дуже важливо розуміти різницю між різними видами життєвого циклу впроваджуваних залежностей. Впроваджувані

залежності - це метод досягнення слабкої пов'язаності між об'єктами та їх залежностями[25]. Розглянемо основні види життєвих циклів:

1. AddTransient. Найбільше підходить невеликим та простим програмам. Має на увазі, що сервіс створюється кожен раз, коли цього вимагає програма;
2. AddScoped - сервіс створюється один раз для кожного запиту;
3. AddSingleton. Являється протилежністю AddTransient. Сервіс створюється лише один раз, а потім тільки визивається з кешу. Найбільше підходить великим додаткам, працюючих на веб-серверах.

Настроюється життєвий цикл в класі Startup.cs за допомогою метода ConfigureServices.

Розглянемо налаштування життєвого циклу залежностей на пригладі коду із даної роботи на рисунку 3.9.

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDBContent>(options => options.UseSqlServer(_confString.GetConnectionString("DefaultConnection")));
    string connection = _confString.GetConnectionString("DefaultConnection");
    services.AddDbContext<AppDBContent>(options => options.UseSqlServer(connection));
    services.AddTransient<IGroups, ClassesRepository>();
    services.AddTransient<IProgram, ProgramRepository>();
```

Рисунок 3.9 - Приклад створення життєвого циклу AddTransient

Підсумуємо різниці між різними видами життєвого циклу в таблиці 3.1.

Таблиця 3.1 - Види життєвого циклу

Вид життєвого циклу	У рамках єдиного http-запиту	Для двох http-запитів
AddTransient	Новий інстант	Новий інстант
AddScoped	Вже існуючий інстант	Новий інстант
AddSingleton	Вже існуючий інстант	Вже існуючий інстант

3.3 Створення та реалізація контролерів

Вище ми описали основну реалізацію та функціонал моделей із схеми Model-View-Controller. Тепер розглянемо створення другого важливого елементу схеми MVC - контролерів. Контролер зберігає деякі спеціальні функції, які будуть повертати інший основний компонент MVC - представлення[26].

Для початку розглянемо реалізацію методу Login у контролері AccountController, який відповідає за авторизацію у нашому додатку на рисунку 3.10.

```
[HttpGet]
0 references
public IActionResult Login()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Login(LoginModel model)
{
    if (ModelState.IsValid)
    {
        User user = await dBContent.Users
            .Include(u => u.Role)
            .FirstOrDefaultAsync(u => u.Email == model.Email && u.Password == model.Password);

        if (user != null)
        {
            await Authenticate(user);

            return RedirectToAction("Index", "Home");
        }
        ModelState.AddModelError("", "Некоректний логін та(або) пароль");
    }
    return View(model);
}
```

Рисунок 3.10 - Метод Login із контролера AccountController

Як можна побачити на рисунку, метод Login розділений на дві частини, вона розділяються за допомогою атрибутів HTTPGet та HTTPPost. Перший метод просто відправляє користувачеві представлення з формою для заповнення, другий виконується саме під час авторизації. Атрибут ValidateAntiforgeryToken створений для протидії підробці міжсайтових запитів. Обидва методи повертають змінну типу IActionResult, який абстрактним класом, в якому визначено один метод ExecuteResult. Як можна побачити на рисунку, після заповнення і відправки форми користувачем, система перевіряє на наявність користувача з даним логіном та паролем. Якщо користувач є в базі даних, його аутентифіковує, авторизовує та відправляє на головну сторінку. Якщо користувача не знайдено, виводиться напис “Некоректний логін та(або) пароль” та повертає на сторінку авторизації.

Аутентифікація проходить за допомогою метода Authenticate(User), реалізованому також в даному контролері. Розглянемо цей метод на рисунку 3.11.

```
2 references
private async Task Authenticate(User user)
{
    var claims = new List<Claim>
    {
        new Claim(ClaimsIdentity.DefaultNameClaimType, user.Email),
        new Claim(ClaimsIdentity.DefaultRoleClaimType, user.Role?.Name)
    };

    ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType,
        ClaimsIdentity.DefaultRoleClaimType);

    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));
}
```

Рисунок 3.11 - Реалізація методу Authenticate

Також на прикладі контролера AdminClassController, розглянемо маршрутизацію та приховування інформації від деяких ролей на рисунку 3.12.

```
[Authorize(Roles = "admin")]
[Route("AdminClass/List")]
[Route("AdminClass/List/{program}")]
0 references
public ViewResult List(string program)
{
    string _program = program;
    IEnumerable<Classes> classes = null;

    string currProgram = "";
    if (string.IsNullOrEmpty(program))
    {
        classes = m_IGroups.classes.OrderBy(i => i.groupName);
        currProgram = "Всі групи";
    }
```

Рисунок 3.12 - Контролер AdminClassController

Атрибут Authorize(Roles = “{Назва ролі, Назва ролі…}” додає можливість показувати деякий контент тільки авторизованим користувачам з деякою роллю. В даній роботі це дозволяє розділити контролери на три групи:

1. Контролери, якими можуть користуватися тільки адміністратори;
2. Контролери, якими можуть користуватися авторизовані користувачі;
3. Та ті сторінки, які бачить навіть не авторизований користувач.

Одним з найважливіших атрибутів, є Route, який додає можливість маршрутизації у контролер. За допомогою даного кода можна додати маршрут:

[Route(“Маршрут/{Деяка змінна}”)]

Наприклад, в даній роботі, цією змінною являється назва кафедри, яка передається у метод List.

Також для реалізації маршрутизації із контролера треба додати код в клас startup.cs. Розглянемо даний код на рисунку 3.13.

```
app.UseMvc(routes =>
{
    routes.MapRoute(name: "default", template: "{controller=Home}/{action=Index}/{id?}");
    routes.MapRoute(name: "programFilter", template: "Class/{action}/{program?}", defaults: new { Controller = "Class", action = "List" });
    routes.MapRoute(name: "pairFilter", template: "Pair/{action}/{pairProgram?}", defaults: new { Controller = "Pair", action = "List" });
```

Рисунок 3.13 - Реалізація маршрутизації в класі startup.cs

Як можна побачити, даний код додає шаблон для маршрутизації, назву та маршрути за замовчуванням.

3.4 Вид

Розглянемо реалізації останнього елементу схеми MVC - View. Як було вирішено вище, в даній роботі для реалізації представлень буде використано рушій Razor. Розглянемо роботу розмітки Razor на прикладі виводу розкладу у вигляді таблиці із даного проекту на рисунку 3.14.

```
<table class="table">
    <thead>
        <tr>
            <th></th>
            <th>Назва</th>
            <th>Викладач</th>
            <th>Група</th>
            <th>Підгрупа</th>
            <th>Вид</th>
            <th>Аудиторія</th>
            <th>Коментар</th>
        </tr>
    </thead>
    <tbody>
        @foreach (Timetable pair in Model.AllPairs)
        {
            <tr>
                <td>@pair.pairNumber.pairNumber</td>
                <td>@pair.pair.PairName</td>
                <td>@pair.pair.teacher.FullName</td>
                <td>@pair.classes.groupName</td>
                <td>@pair.subgroup.subgroup</td>
                <td>@pair.pair.activity.ActivityType</td>
                <td>@pair.audience.AudienceNumber</td>
                <td>@pair.Comment</td>
            </tr>
        }
    </tbody>
</table>
```

Рисунок 3.14 – Приклад роботи розмітки Razor

Як можна побачити, синтаксис роботи Razor складається з коду C# та HTML розмітки. При цьому можна побачити, що він інтуїтивно простий. Результат роботи даного коду можна розглянути на рисунку 3.15.

	Назва	Викладач	Група	Підгрупа	Вид	Аудиторія	Коментар
1	Література	Дахно Н. Б.	МІТ-11	Всі групи	Практика	405	
2	Література	Дахно Н. Б.	МІТ-11	Всі групи	Практика	101	
3	Вища математика	Махович О. І.	МІТ-11	Підгрупа 1	Лекція	309	
3	Англійська	Кравченко Ю. В.	МІТ-11	Підгрупа 2	Лабораторна	212	
4	Вища математика	Махович О. І.	МІТ-11	Всі групи	Лекція	309	Відміна пари

Рисунок 3.15 - Приклад результату роботи коду Razor

Також розглянемо код, який використовується для редагування інформації про пару.

```
@model Pair
<div class="form-group">
        <label asp-for="courseId"></label>
        <div class="col-md-5">
           <select name="courseId" asp-for="courseId">
              <option value="1">1 курс</option>
              <option value="2">2 курс</option>
              <option value="3">3 курс</option>
              <option value="4">4 курс</option>
           </select>
        </div>
     </div>
```

А також розглянемо результат роботи даного коду на рисунку 3.16.

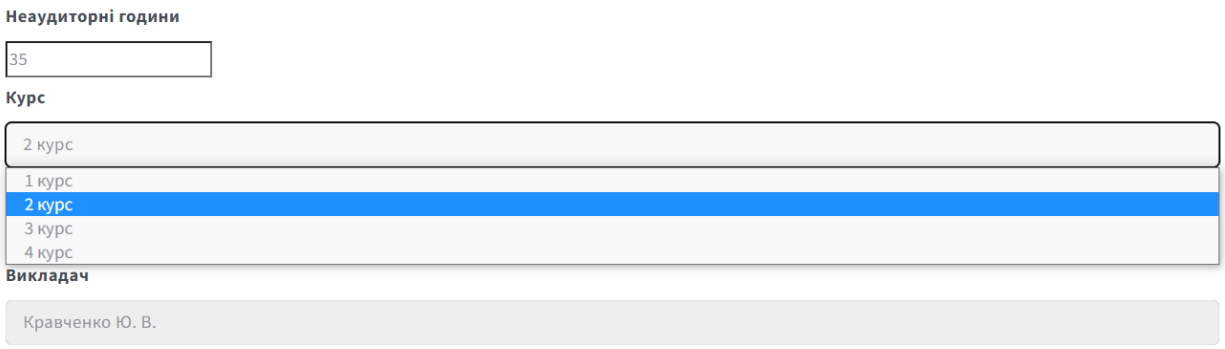

Рисунок 3.16 - Приклад редагування інформації про пару

ВИСНОВКИ

1. Автоматизація та інформатизація стає невід'ємною частиною життя будь-якої людини. Через це велику актуальність набирає можливість користування відповідних технологій у житті. Сучасні бази даних зберігають та обробляють величезні масиви даних зі складною організацію. Вони можуть задовольнити найрізноманітніші вимоги користувачів. Ціль будь-якої бази даних - обробляти, зберігати та виводити інформацію із реального світу в деякій галузі. В даній роботи було проведено проектування та реалізацію однієї з таких баз даних, що може в майбутньому значно збільшити популярність та зручність систем "Електронний розклад факультету".
2. В першому розділі була проведена робота з літературою по проектуванню бази даних та розробці додатку на платформі .NET. Були розглянуті такі технології, як реляційні та нереляційни бази даних. Був проаналізований ринок систем управління базами даних, та на основі переваг та недоліків обрано СУБД MS SQL Server. Була проведена робота із вивчення технологій SQL та ASP.NET Core, їх можливостей, переваг та недоліків, а також процедурного розширення Transact-SQL.
3. У другому розділі була проведена робота з проектування схеми бази даних. Була проаналізована предметна область "Електронний розклад факультету" на функціональні можливості. Далі була розглянута архітектура технології ASP.NET Core та моделі Model-View-Controller. Була проведена робота з розробки структури проекту, що значно спростило роботу з проектом далі. Далі були розглянути так технології, як рушій Razor та ORM-технологія Entity Framework.
4. В останньому розділі була проведена робота з розробки веб-додатку "Електронний розклад факультету". Застосовувалися вже розглянуті та

вивчені технології на практиці. За допомогою зручної структури, проведення аналізу предметної області та зручної та ефективної бази даних, розробку можна було розбити на прості етапи. Серед основних: розробка моделей, написання функціоналу до моделей, підключення додатку до бази даних, створення та реалізація контролерів та вигляду нашого додатку. На основі цього був створений прототип додатку “Електронний розклад факультету”. Даний прототип та базу даних можна використовувати при розробці реального сайту для розкладу будь-якого навчального закладу.

ПЕРЕЛІК ПОСИЛАНЬ

1. [Електронний ресурс]. – Режим доступу: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-a-relational-database/#whatis
2. [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/ACID
3. [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/254773/
4. [Електронний ресурс]. – Режим доступу: https://metanit.com/sql/sqlserver/11.1.php
5. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель, SQL Полное руководство Третье издание – 959 с.
6. [Електронний ресурс]. – Режим доступу: https://coderlessons.com/tutorials/bazy-dannykh/uchit-sql/sql-obzor
7. [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85
8. Офіційний сайт Oracle Database [Електронний ресурс]. – Режим доступу: https://www.oracle.com/ru/index.html
9. Офіційний сайт PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/
10. Офіційний сайт MS SQL Server[Електронний ресурс]. – Режим доступу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019?ranMID=24542&ranEAID=a1LgFw09t88&ranSiteID=a1LgFw09t88-HpXfoeALYk1aVD1J3DuwoA&epi=a1LgFw09t88-HpXfoeALYk1aVD1J3DuwoA&irgwc=1&OCID=AID2000142_aff_7593_1243925&tduid=%28ir__16rjrcdbqwkfq2k1kk0sohz3
11. [Електронний ресурс]. – Режим доступу: http://programer.org.ua/arhitektura-obrobky-sql-zapytiv-v-microsoft-sql-server/
12. [Електронний ресурс]. – Режим доступу: https://www.simplilearn.com/tutorials/sql-tutorial/transact-sql
13. Офіційна документація ASP.NET Core [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/?ranMID=24542&ranEAID=a1LgFw09t88&ranSiteID=a1LgFw09t88-MVk0aIKJ4VxQI7m8miEXwQ&epi=a1LgFw09t88-MVk0aIKJ4VxQI7m8miEXwQ&irgwc=1&OCID=AID2000142_aff_7593_1243925&tduid=(ir__16rjrcdbqwkfq2k1kk0sohz3wf2xugilguc2mf2b00)(7593)(1243925)(a1LgFw09t88-MVk0aIKJ4VxQI7m8miEXwQ)()&irclickid=_16rjrcdbqwkfq2k1kk0sohz3wf2xugilguc2mf2b00&view=aspnetcore-5.0
14. [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

15. [Електронний ресурс]. – Режим доступу: https://www.codemag.com/Article/2111062/What%E2%80%99s-New-in-ASP.NET-Core-in-.NET-6
16. [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/488054/
17. [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/321392/
18. Офіційна документація Entity Framework [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/ef/?ranMID=24542&ranEAID=a1LgFw09t88&ranSiteID=a1LgFw09t88-K07AtVgvOTbdvFphGO61qA&epi=a1LgFw09t88-K07AtVgvOTbdvFphGO61qA&irgwc=1&OCID=AID2000142_aff_7593_1243925&tduid=%28ir__16rjrcdbqwkfq2k1kk0sohz3wf2xuginjec2mf2b00%29%287593%29%281243925%29%28a1LgFw09t88-K07AtVgvOTbdvFphGO61qA%29%28%29&irclickid=_16rjrcdbqwkfq2k1kk0sohz3wf2xuginjec2mf2b00
19. [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/entity_framework/entity_framework_data_model.htm
20. [Електронний ресурс]. – Режим доступу: https://www.entityframeworktutorial.net/efcore/fluent-api-in-entity-framework-core.aspx
21. [Електронний ресурс]. – Режим доступу: https://skillbox.ru/media/code/entity_framework/
22. [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-5.0
23. [Електронний ресурс]. – Режим доступу: https://www.entityframeworktutorial.net/querying-entity-graph-in-entity-framework.aspx
24. [Електронний ресурс]. – Режим доступу: https://metanit.com/sharp/aspnet5/19.1.php
25. [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/otus/blog/539762/