МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

# КУРСОВА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни «Бази даних»
на тему:
**«Освітній портал для школи»**


студента II курсу групи ІПЗ-20-4
спеціальності 121 «Інженерія програмного забезпечення»
Васьківського Віталія Юрійовича
(прізвище, ім'я та по-батькові)

Керівник: ст. викл. кафедри ІПЗ
Чижмотря О.В


Дата захисту: "___ " _____ 20__ р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії _____ Ольга КОРОТУН .
(підпис) (прізвище та ініціали)
_____ Світлана КРАВЧЕНКО .
(підпис) (прізвище та ініціали)
_____ Інна СУГОНЯК .
(підпис) (прізвище та ініціали)

Житомир – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет інформаційно-комп'ютерних технологій
Кафедра інженерії програмного забезпечення
Освітній рівень: бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
В.о зав кафедри ІПЗ
_Андрій МОРОЗОВ___
"____"_____2022 р.

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Васьківському Віталію Юрійовичу

1. Тема роботи: Розробка освітнього порталу для школи
керівник роботи: ст. викл. кафедри ІПЗ Чижмотря О.В .
2. Строк подання студентом: " 15 " липня 2022р.
3. Вихідні дані до роботи: Розробити освітній портал для школи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що підлягають розробці)
   - Аналіз інформаційних потоків та особливостей предметної області дослідження
   - Проектування бази даних за напрямком курсової роботи
   - Реалізація підсистеми обробки даних за напрямком курсової роботи
   - Адміністрування баз даних

5. Перелік графічного матеріалу(з точним зазначенням обов'язкових креслень)
   1. Презентація до КР
   2. Посилання на репозиторій: https://github.com/grozer221/EducationalPortal

6. Консультанти розділів проекту (роботи)

| Розділ | Прізвище, ініціали та посади консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1,2,3 | Чижмотря О.В , ст. викл. кафедри ІПЗ | 10.02.2022 | 10.02.2022 |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання " 10 " лютого 2022 р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1 | Постановка задачі | 10.02 | Виконано |
| 2 | Пошук, огляд та аналіз аналогічних розробок | 23.02 | Виконано |
| 3 | Формулювання технічного завдання | 01.04 | Виконано |
| 4 | Опрацювання літературних джерел | 09.04 | Виконано |
| 5 | Проектування структури | 29.04 | Виконано |
| 6 | Написання програмного коду | 04.06 | Виконано |
| 7 | Відлагодження | 20.06 | Виконано |
| 8 | Написання пояснювальної записки | 25.06 | Виконано |
| 9 | Захист | | |

**Студент** _____ Васьківський В. Ю.

(підпис) (прізвище та ініціали)

**Керівник проекту** _____ Олексій ЧИЖМОТРЯ

(підпис) (прізвище та ініціали)

## РЕФЕРАТ

Завданням на курсовой проект (роботу) було розробити освітній портал для школи.

Пояснювальна записка до курсового проекта на тему «Розробка освітнього порталу для школи» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 50 сторінках друкованого тексту.

Пояснювальна записка має 3 сторінки додатків. Список використаних джерел містить 11 найменувань і займає 1 сторінку. В роботі наведено 23 рисунок. Загальний обсяг роботи – 54 сторінок.

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова: SINGLE PAGE APPLICATION, JAVASCRIPT, TYPESCRIPT, REACT, REDUX, APOLLO CLIENT, ASP.NET, GRAPH QL .NET, MS SQL, ENTITY FRAMEWORK, СЕРВЕР, АВТОРИЗАЦІЯ

## Зміст

## ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних

КР – курсова робота

ПЗ – Програмне забезпечення

GRUD – Create – створення, Read – читання, Update – оновлення, Delete – видалення

JWT – Json Web Token

ТЗ – технічне завдання

СУБД – система управління базою даних

# ВСТУП

В даній курсовій роботі буде наведено процес побудови клієнтського додатку та розробка освітнього порталу для школи.

**Актуальність теми.** Використання баз даних є однією з характерних рис більшості сучасних інформаційних систем. По своїй суті бази даних є тим, навколо чого і будується інформаційна система будь-якого підприємства. Тому теорії створення та практиці використання баз даних приділяється достатня увага протягом періоду функціонування ІС. Тривалий час основним типом були реляційні бази даних, які на сьогодні вже вважаються класичними. Однак класичність не означає легко та швидко. Хоча на практиці і використовується чимало різноманітних баз даних, але для більшості з них існує велика кількість шаблонних фрагментів коду що, робить створення додатку нудним і призводить до однакових помилок при розобці. Тому для спрощення процесу розробки додатку потрібно знаходити щось зручніше, легше та сучасніше. Цьому завданню і було приділену головну увагу при розробці.

**Метою роботи** є дослідження особливостей проектування та реалізації бази днаих за визначеною темою курсової роботи.

**Завданням на курсову роботу є :**

- аналіз теоретичних засад проектування та реалізації систем на основі баз даних;
- визначення інформаційних потреб предметної області дослідження;
- аналіз напрямку ризиків інформаційних потоків та їх структури;
- проектування бази даних за визначеною предметною областю;
- розробка математичної та алгоритмічної моделі функціонування системи на основі БД;
- реалізація БД та інтерфейсних засобів інформаційної системи.

**Об'єктом дослідження** є методи та засоби проектування баз даних за визначеними предметними областями.

**Предметом дослідження** можливості застосування концепції БД та СУБД для забезпечення інформаційних потреб предметної області.

# РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

## 1.1 Аналіз інформаційних потреб та визначення предметної області дослідження

В даному курсовому проекті за предметну область взято базу даних, яка буде зберігати усю необхідну інформацію про освітні портали. Цей портал надає можливість отримувати знання з різних спеціальностей за допомогою мережі Internet, тому деякі дання потрібно зберігати саме в мережі. Щоб користувачі мали змогу одержати для себе більш повну інформацію про освітній процес, необхідно створити базу даних, яка зберігатиме усі необхідні дані. А для амінів, потрібно зробити можливість контролю даних та отримання статистичної інформації.

Для коректного виконання запитів на отримання даних, дані повинні вноситись та змінюватись у БД. За це будуть відповідати деякий перелік операцій:

- Вхід користувача. Функція, що дозволить розрізняти користувачів.
- Сворення навчальних років.
- Створення класів.
- Стоворення учнів або вчителів.
- Створення предметів та для них постів.
- Статистика. Є важливим аналізування інформації про виконання домашніх робіт учнями
- Створення бекапів. Адміністратору варто мати можливість зробити резервну копію бази щоб зберігти дані та потім мати змогу відновити її у випадку взлому, або пошкодження.
- Відновлення бази. Дана функція буде очищати таблиці поточної БД, та заповнювати їх даними з обраного бекапу.

## 1.2 Архітектура та засоби реалізації бази даних

Аналіз та вибір СУБД проведемо з урахуванням того, що число клієнтських місць становить від 30 до 500, а доступ до даних має бути максимально ефективним. Обчислювальна техніка працюватиме під керівництвом ОС Windows і Linux. В таблицях наведено порівняльні характеристики СУБД.

Таблиця 1.

| Характеристика | Microsoft SQL Server | MongoDB | Postgresql |
|---|---|---|---|
| Робота під керівництвом ОС | Задовільно | Добре | Добре |
| Складність | Відмінно | Добре | Відмінно |
| Доступ до даних | Відмінно | Відмінно | Відмінно |
| Використання у великих проектах | Відмінно | Добре | Відмінно |
| Використання у малих проектах | Добре | Відмінно | Відмінно |
| Схема даних | Відмінно | Відмінно | Добре |
| Підтримувані мови програмування | Відмінно | Відмінно | Відмінно |
| Підключення до Web | Відмінно | Відмінно | Відмінно |
| Графічні інструменти | Відмінно | Відмінно | Відмінно |
| Тригери | Відмінно | Добре | Відмінно |
| Паралелізм | Відмінно | Відмінно | Відмінно |
| Одночасний доступ декількох користувачів | Добре | Добре | Відмінно |

| | | | |
|---|---|---|---|
| Обробка даних мультимедіа | Слабо | Добре | Відмінно |
| Пошук за регулярними виразами | Добре | Відмінно | Відмінно |
| Засоби аналізу | Відмінно | Відмінно | Відмінно |

Пояснення таблиці:

MongoDB є провідною базою даних документів . Яка побудована на розподіленій масштабованій архітектурі та стала комплексною хмарною платформою для керування та доставки даних у програми. MongoDB обробляє транзакційні, операційні та аналітичні навантаження.

MongoDB зберігає дані як документи у двійковому представленні під назвою BSON (Binary JSON). Поля можуть відрізнятися від документа до документа; немає необхідності оголошувати системі структуру документів – документи описуються самі. Якщо до документа потрібно додати нове поле, це поле можна створити, не впливаючи на всі інші документи в колекції, не оновлюючи центральний системний каталог, оновлюючи ORM і не переводячи систему в автономний режим. За бажанням перевірку схеми можна використовувати для забезпечення контролю керування даними над кожною колекцією.

PostgreSQL є прикладом добре керованого проекту з відкритим кодом.

PostgreSQL називає себе системою об’єктно-реляційної бази даних з відкритим кодом. Це база даних SQL, яка має певні стратегії для обробки індексування, підвищення паралельності та впровадження оптимізацій і підвищення продуктивності, включаючи розширене індексування, розділення таблиць та інші механізми. Об’єктна частина PostgreSQL пов’язана з багатьма розширеннями, які дозволяють включати інші типи даних, такі як об’єкти даних JSON та XML.

MS SQL Server - система управління базами даних, яка розробляється корпорацією Microsoft. Як сервер даних виконує головну функцію по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватися як на тому ж самому сервері, так і у мережі. Мова, що використовується для запитів — Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.

Модель даних документа MongoDB природним чином відображається на об’єктах у коді програми, що полегшує розробникам вивчення та використання. Документи дають можливість представляти ієрархічні зв’язки для зберігання масивів та інших складніших структур. Документи JSON можуть зберігати дані в полях, у вигляді масивів у вигляді вкладених під документів. Таким чином пов’язану інформацію можна зберігати разом для швидкого доступу до запитів за допомогою багатофункціональної та виразної мови запитів MongoDB .

Однією з найпотужніших функцій реляційних баз даних, що полегшує написання програм, є транзакції ACID. Деталі того, як визначаються та реалізуються транзакції ACID, містять багато підручників з інформатики. Велика частина дискусій у сфері IT стосується рівнів ізоляції в транзакціях бази даних . PostgreSQL за замовчуванням використовує рівень ізоляції для читання та дозволяє користувачам налаштовувати його до рівня ізоляції, який можна серіалізувати.

У реляційній базі даних дані, про які йде мова, моделюватимуться в окремих батьківських і дочірніх таблицях у табличній схемі. Це означає, що оновлення всіх записів одночасно потребуватиме транзакції. У певному сенсі базам даних документів легше реалізовувати транзакції, оскільки вони

згруповують дані в документі, а запис і читання документа є операцією для якої не потрібна транзакція з кількома документами. Одне або кілька полів можна записати за одну операцію, включаючи оновлення кількох піддокументів і елементів масиву. MongoDB гарантує повну ізоляцію під час оновлення документа. Будь-які помилки ініціюватимуть операцію оновлення до відкату, скасовуючи зміни та гарантуючи, що клієнти отримають узгоджений вигляд документа.

MongoDB також підтримує транзакції бази даних у багатьох документах, тому фрагменти пов'язаних змін можна зафіксувати або відкотити. Завдяки можливості транзакцій з багатьма документами MongoDB є однією з небагатьох баз даних, яка поєднує в собі гарантії ACID традиційних реляційних баз даних зі швидкістю, гнучкістю та потужністю моделі документів.

Транзакції в MongoDB схожі на транзакції, які використтовуються в PostgreSQL. Транзакції в MongoDB складаються з кількох інструкцій із подібним синтаксисом (наприклад, starttransaction and committransaction), тому будь-кому, хто має попередній досвід роботи з транзакціями, легко додати такі конструкції до будь-якої програми.

Властивості ACID у SQL Server забезпечують цілісність даних під час транзакції. SQL ACID є акронімом для атомарності, узгодженості, ізоляції, міцності.

Модель реляційної бази даних, яку використовує PostgreSQL, базується на зберіганні даних у таблицях і подальшому використанні мови структурованих запитів (SQL) для доступу до бази даних.

Щоб це працювало, у PostgreSQL та інших базах даних SQL необхідно створити схему бази даних і встановити зв'язки даних до того, як база даних буде заповнена даними. Пов'язана інформація може зберігатися в окремих таблицях, але пов'язана за допомогою зовнішніх ключів і JOIN. Більшість змін

у схемі вимагає процедури міграції, яка може перевести базу даних у автономний режим або знизити продуктивність програми під час її роботи.

Сильною стороною SQL є його потужна та широко відома мова запитів із великою системою інструментів. Проблема використання реляційної бази даних полягає в необхідності заздалегідь визначити її структуру. Змінити структуру після завантаження даних часто дуже складно , що потребує кількох команд розробників, адміністратора баз даних і операцій для тісної координації змін.

MongoDB не використовує SQL за замовчуванням. Натомість для роботи з документами в MongoDB і вилучення даних MongoDB надає власну мову запитів (MQL), яка пропонує майже таку саму потужність і гнучкість, як SQL. Наприклад, як і SQL, MQL дозволяє посилатися на дані з кількох таблиць, перетворювати та агрегувати ці дані, а також фільтрувати для конкретних результатів, які вам потрібні. На відміну від SQL, MQL працює ідіоматично для кожної мови програмування .

Продуктивність запитів у MongoDB можна прискорити, створивши індекси для полів у документах і піддокументах. MongoDB дозволяє індексувати будь-які поля документа, включно з тими, які глибоко вкладені в масиви та піддокументи, і ефективно надсилати запити.

Microsoft SQL Server як мову запитів використовує версію SQL, що отримала назву Transact-SQL. TSQL в основному використовується для написання цілої програми процедури блокової функції, яка визначає, як все має бути завершено, і створення програми, у якій кожна програма надсилає транзактний запит через SQL-сервер і немає взаємодії з базою даних. Він виконується як цілий блок із розширенням мови SQL. У T-SQL використовуються різні типи функцій T-SQL: скалярна функція, функція ранжирування, агрегатна функція, функція набору рядків

MongoDB створено для масштабування. Таким чином, випадки використання, які вимагають надшвидких запитів і величезних обсягів даних або обох, можуть бути оброблені шляхом створення все більших кластерів невеликих машин.

MongoDB базується на розподіленій архітектурі, яка дозволяє користувачам масштабувати багато екземплярів, і доведено, що вона працює над величезними додатками, незалежно від того, вимірюється кількість користувачів або розмір даних . Стратегія масштабування базується на використанні більшої кількості менших машин. Цю стратегію можна розширити до сотень машин.

У PostgreSQL підхід до масштабування залежить від того, чи йдеться про запис чи читання даних. Для записів він базується на архітектурі масштабування, у якій одна основна машина, на якій працює PostgreSQL, має бути максимально потужною для масштабування. Для читання можна масштабувати PostgreSQL шляхом створення реплік, але кожна репліка повинна містити повну копію бази даних.

Техніка, яка робить MongoDB масштабованою, базується на ідеї інтелектуального розподілу (sharding) даних між примірниками в кластері. MongoDB не розбиває документи на частини, документи є незалежними одиницями, що полегшує їх розподіл між декількома серверами, зберігаючи локальність даних.

MongoDB реалізував сучасний набір елементів керування та інтеграції кібербезпеки як для локальної, так і для хмарної версії. Це включає потужні парадигми безпеки, як шифрування на рівні поля на стороні клієнта , що дозволяє шифрувати дані перед тим, як вони надсилаються через мережу до бази даних.

PostgreSQL має повний набір функцій безпеки, включаючи багато типів шифрування. PostgreSQL доступний у хмарі у всіх основних хмарних

провайдерів. Хоча це та сама база даних, робочі інструменти та інструменти розробника відрізняються залежно від постачальника хмари, що ускладнює міграцію між різними хмарами. MongoDB Atlas однаково працює в усіх трьох основних хмарних провайдерах, спрощуючи міграцію та багатохмарне розгортання.

Оскільки PostgreSQL покладається на стратегію масштабування, він повинен максимально використовувати доступні обчислювальні ресурси. PostgreSQL робить це за допомогою різноманітних стратегій для індексування та паралельності.

PostgreSQL пропонує різноманітні потужні типи індексів, які найкраще відповідають заданому навантаженню запиту. На додаток до зрілого планувальника запитів і оптимізатора, PostgreSQL пропонує оптимізацію продуктивності, включаючи розпаралелювання запитів на читання, розділення таблиць і своєчасну (JIT) компіляцію виразів.

База даних відповідає широкому спектру стандартів безпеки та має численні функції для підтримки надійності, резервного копіювання та аварійного відновлення, як правило, за допомогою інструментів сторонніх розробників.

Модель безпеки MS SQL Server — це тісна інтеграція між режимом автентифікації Windows у Windows Server і базою даних. Режим автентифікації Windows працює краще в таких сценаріях: коли є контролер домену, з екземплярами бази даних SQL Server Express або LocalDB, коли і програма, і база даних знаходяться в одному машинному середовищі. Змішаний режим включає процес автентифікації за допомогою Windows Server і MS SQL Server. Тут база даних розгортає механізми Windows Password Policy. Розроблена складність Windows Password Policy спрямована на запобігання хакерським атакам.

Крім того, що Microsoft SQL Server та Postgresql зберігають дані реляційно, а MongoDb використовує сховище документів, дані субд мають інші різні характеристики, більше того, ці характеристики можна оцінити, що і зображено у таблиці номер 1.

У всіх СУБД користувачі отримують доступ лише туди, куди йому його надають, що забезпечує безпеку даних.

Схема даних (Схема баз даних — це структура системи баз даних описана мовою, яка підтримується системою керування баз даних і відноситься до організації даних для створення плану побудови бази даних з розподілом на таблиці.). Підтримка фіксованої схеми та розгортання без схеми з можливістю взаємодії між ними реалізована лише не підтримується ні в одній з БД, у той час як в SQL Server, Postgresql та MongoDb потрібно розгортати лише з описаною схемою даних.

Підтримка одночасного маніпулювання даними (паралелізм), реалізовано у всіх СУБД, що є великим плюсом для цих СУБД.

Кожна з даних СУБД підтримує одночасний доступ кільком користувачам, що є теж неабиким плюсом. Однак кожна з них має різні ліміти та різне розподілення ресурсів для кількох користувачів, в результаті аналізування цих критеріїв і було виставлено оцінки в таблиці 1. Обробка медіаданих реалізована у всіх СУБД, але в SQL SERVER для цього потрібно вручну створити таблицю, яка буде зберігати не лише сам файл, а й ім’я, щоб не втратити цю інформацію.

Щодо пошуку даних за певним значенням, що має зустрітись в тексті (регулярні вирази), то всі СУБД, є на одному рівні, адже дозволяють гнучно змінювати запити та обробляти їх.

Отже можна зробити висновок, що MS SQL краще підходить для написання доволі складних проєктів, MongoDB – для невеликих та середніх, а PostgreSql можна використовувати в проєкті будь якого розміру.

**Висновки до першого розділу :**

В ході виконання першого розділу було проаналізовано три СУБД також було визначено переваги кожної з СУБД. Також було визначено головні функції програми та приблизну логіку доступу до даних. Також, під час виконання даного роздылу, було вирішено виконувати курсовий проект за допомогою СУБД MS SQL Server. Це дозволить реалізувати БД якісно, надійно та швидко, без необхідності ручного налаштування серверу БД.

# РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ ОСВІТНОГО ПОРТАЛУ ДЛЯ ШКОЛИ

## 2.1 Аналіз структури інформаційних процесів адміністраторів

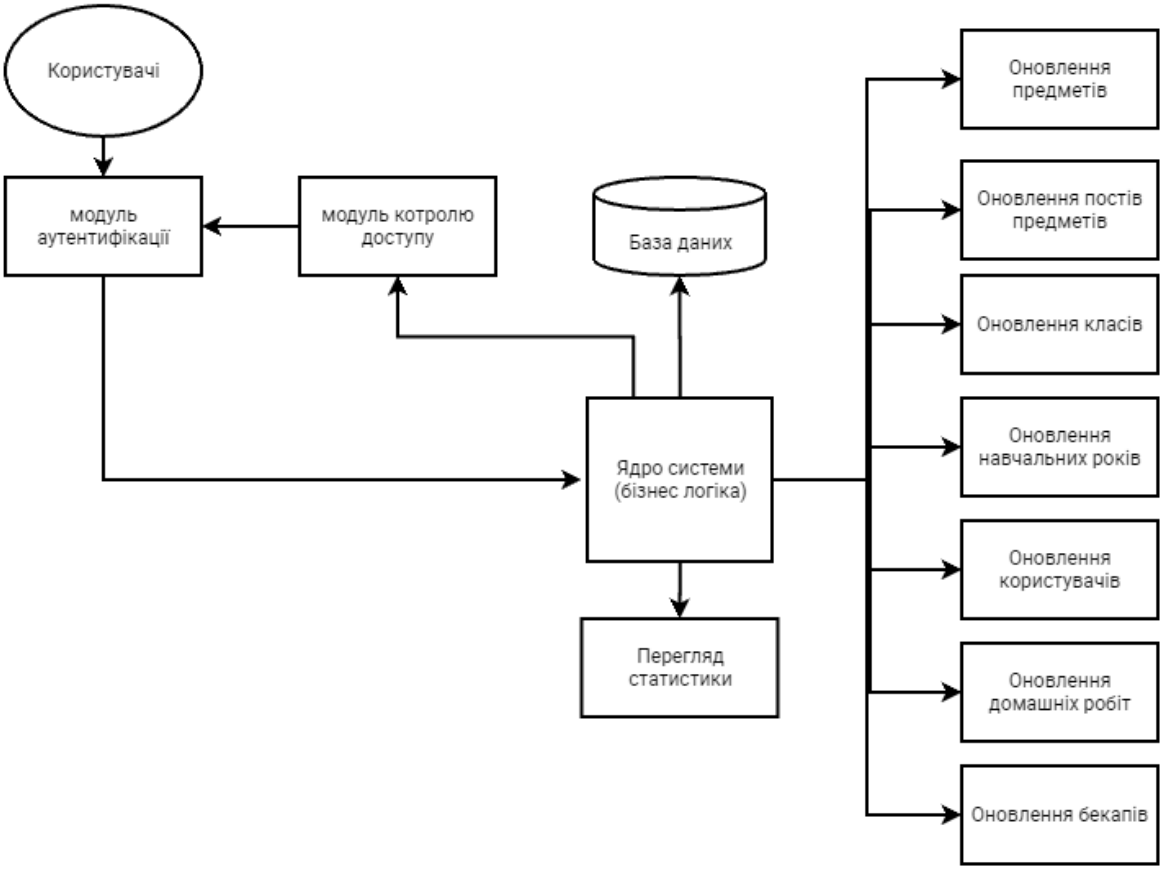


Рис. 2.1. Структура автоматизованої для освітнього порталу

Користувачі будуть мати змогу працювати з додатком після аутентифікації. Після успішного входу користувач зможе виконувати відповідні операції для своєї ролі(учень, вчитель). Адмін матиме повний доступ до зміни будь яких данних, а також до перегляду статистики сайту та створення резервних копій бази.

Далі наведено приблизні структурні схеми функцій системи.

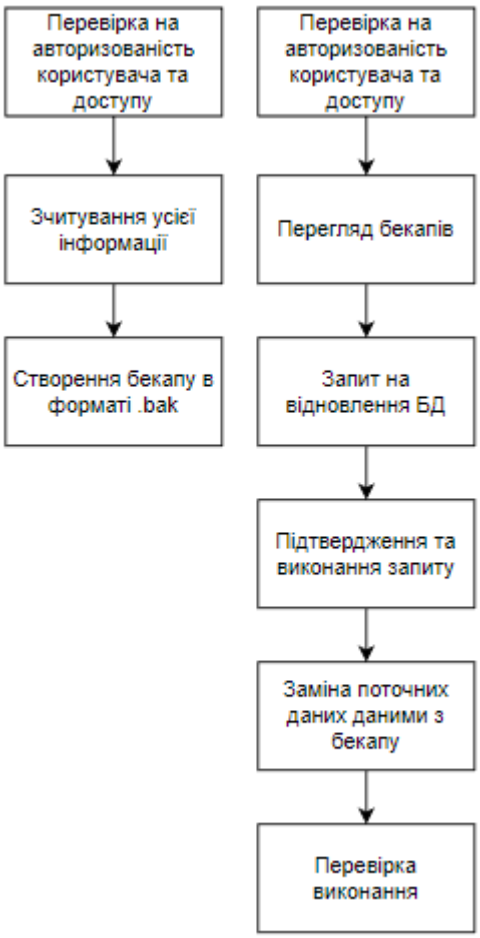


Рис. 2.2. Структурна схема кількох функцій системи, а саме – створення бакапу та відновлення бази

## 2.2 Проектування бази даних освітнього проталу

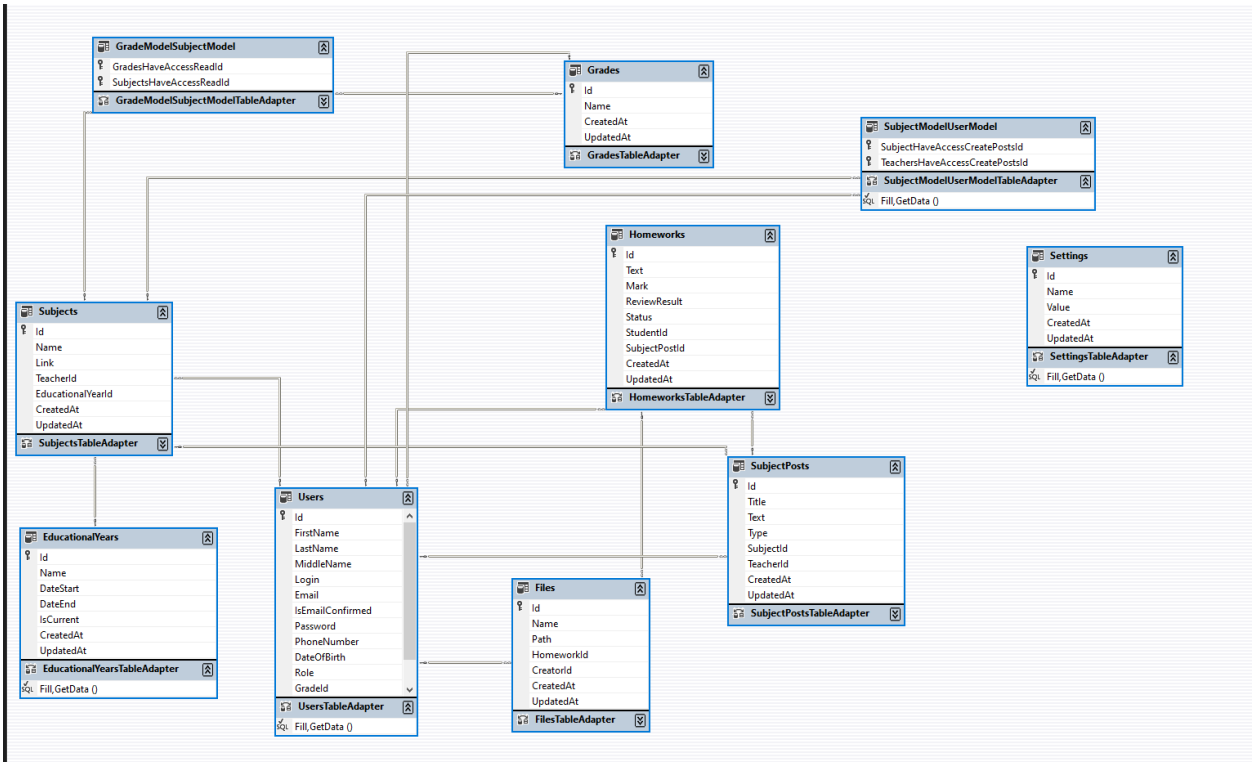


Рис. 2.3. Діаграма “сутність-за’язок” (фізичний рівень) системи

Внаслідок проектування до бази даних включено таблиці:

- EducationalYears
- Files
- GradeModelSubjectModel
- Grades
- Homeworks
- Settings
- SubjectModelUserModel
- SubjectPosts
- Subjects
- Users

Для збереження навчальних років призначена таблиця «EducationalYears». Структура таблиці наведена нижче:

Таблиця 2

Структура таблиці «EducationalYears»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Name | nvarchar(450) | - | - | Назва |
| DateStart | datetime2(7) | - | - | Дата початку |
| DateEnd | datetime2(7) | - | - | Дата закінчення |
| IsCurrent | bit | - | - | Чи є поточним |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження файлів призначена таблиця «Files». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «Files»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Name | nvarchar(MAX) | - | - | Назва |
| Path | nvarchar(MAX) | - | - | Шлях |
| HomeworkId | uniqueidentifier | + | + | Id домашньої роботи |
| CreatorId | uniqueidentifier | + | + | Id користувача, що створив |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження звязку між класами та предметами призначена таблиця «GradeModelSubjectModel». Структура таблиці наведена нижче:

Таблиця 4

Структура таблиці «GradeModelSubjectModel»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| GradesHaveAccessReadId | uniqueidentifier | + | + | Id класу |
| SubjectsHaveAccessReadId | uniqueidentifier | + | + | Id предмету |

Для збереження класів призначена таблиця «Grades». Структура таблиці наведена нижче:

Таблиця 5

Структура таблиці «Grades»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Name | nvarchar(MAX) | - | - | Назва |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження домашніх робіт призначена таблиця «Homeworks». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «Homeworks»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Text | nvarchar(MAX) | - | - | Текст |
| Mark | nvarchar(MAX) | - | - | Оцінка |
| ReviewResult | nvarchar(MAX) | - | - | Результат перевірки |
| Status | nvarchar(MAX) | - | - | Статус |

| StudentId | uniqueidentifier | + | + | Id учня |
|---|---|---|---|---|
| SubjectPostId | uniqueidentifier | + | + | Id поста предмету |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження налаштувань призначена таблиця «Settings». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «Settings»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Name | nvarchar(MAX) | - | - | Назва |
| Value | nvarchar(MAX) | - | - | Значення |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження звязку між предметами та користувачами призначена таблиця «SubjectModelUserModel». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «SubjectModelUserModel»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| SubjectHaveAccessCreatePostsId | uniqueidentifier | + | + | Id предмету |
| TeachersHaveAccessCreatePostsId | uniqueidentifier | + | + | Id вчителя |

Для збереження постів предмета призначена таблиця «SubjectPosts». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «SubjectPosts»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Title | nvarchar(MAX) | - | - | Заголовок |
| Text | nvarchar(MAX) | | | Текст |
| Type | int | - | - | Тип |
| SubjectId | uniqueidentifier | + | + | Id предмету |
| TeacherId | uniqueidentifier | + | + | Id вчителя |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження предметів призначена таблиця «Subjects». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «Subjects»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| Name | nvarchar(MAX) | - | - | Заголовок |
| Link | nvarchar(MAX) | | | Текст |
| TeacherId | uniqueidentifier | + | + | Id вчителя |
| EducationalYearId | uniqueidentifier | + | + | Id навчального року |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

Для збереження користувачів призначена таблиця «Users». Структура таблиці наведена нижче:

Таблиця 3

Структура таблиці «Users»

| Назва | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| Id | uniqueidentifier | + | - | Id |
| FirstName | nvarchar(MAX) | - | - | Ім’я |
| LastName | nvarchar(MAX) | - | - | Прізвище |
| MiddleName | nvarchar(MAX) | - | - | По-батькові |
| Login | nvarchar(MAX) | - | - | Логін |
| Email | nvarchar(MAX) | - | - | Email |
| IsEmailConfirmed | bit | - | - | Чи підтверджений email |
| Password | nvarchar(MAX) | - | - | Пароль |
| PhoneNumber | nvarchar(MAX) | - | - | Номер телефону |
| DateOfBirth | datetime2(7) | - | - | Дата народження |
| Role | nvarchar(MAX) | - | - | Роль |
| GradeId | uniqueidentifier | + | + | Id класу |
| CreatedAt | datetime2(7) | - | - | Дата створення |
| UpdatedAt | datetime2(7) | - | - | Дата оновлення |

## 2.3 Розробка математичної моделі та алгоритмів обробки інформації про освітнього порталу

Для збереження користувача в баз даних, юзеру доведеться заповнити відповідну форму реєстрації, де повнино бути коректно заповнено всі необхідні поля, а сам псевдонім, пошта, прізвище та ім'я (не є обов’язковими) та пароль. Після відправки форму, дані обробляються на бекенді, де додається користувацька роль та перевіряється вільність введеного юзернейму та пошти.

Виконання даних операцій користувачу буде доступне операція логіну, де він вводить власні дані, а саме пошту та пароль, і після перевірки корекності даних, надається доступ до користувацького запису. Для оновлення даних потрібно буде перейти до сторінки користувача та змінити дані, там же можна буде змамінити фото профілю.

Далів взалежностів, яка у коритувача роль, він зможе мати доступ до того чи іншого функціоналу порталу. Якщо роль учень(Student) він зможе переглядати пости преметів та предмети до яких дали доступ вчителі. Також відпраляти домашню роботу для поста з типом домашня роботи

Для користувачів з ролю вчитель(Teacher) функціоналу дещо більше. Користувач зможе перегладати свої та всі предмети порталу, домашні роботи, які надіслали учні, класи, учнів, вчителів, навчальні роки.

Далі розглянемо функції які доступні тільки адміну, а саме редагувати весь контент, а також створення резервних копій бази та відновлення з них.

Також потрібно розглянути те, як буде відбуватипя процес створення бекапів. Для створення бекапу адмін має перейти на сторінку з бекапами та натиснути на відповідну кнопку по створенню бекапу. Далі запит надслиється на бекенд, де він виконується, емулюючи роботу з консоллю та виконуючи відповідні команди для створення бекапу бази даних. Після створення бекапу, адмін побачить його в списку доступних для відновлення.

Для відновлення з резервної копії потрібно обрати бекап та виконати віідповідну команду, запит надсилається на бекенд, де він оброблюється, емулюючи роботу з консоллю та виконуючи відповідні команди для рестору БД. Під час цього процесу база очищається та заповнюється даними з бекапу.

**Висновки до другого розділу**

У результаті виконання даного розділу було проаналізовано інформаційні процеси, спроектовано структуру бази даних, описано приблизну логіку функцій для роботи з базою на клієнті, пояснено деяку

логіку обробки, збереження та генерації даних для бази. Також було описано як буде відбуватись побудова та генерація статистичної інформації платформи.

# РОЗДІЛ 3 ОПИС РОБОТИ З ДОДАТКОМ

## 3.1 Опис роботи з додатком (Опис інтерфейсу)

Після запуску веб-додатку з'являється сторінка входу. Без автентифікації порталом буде не можливо користуватися.

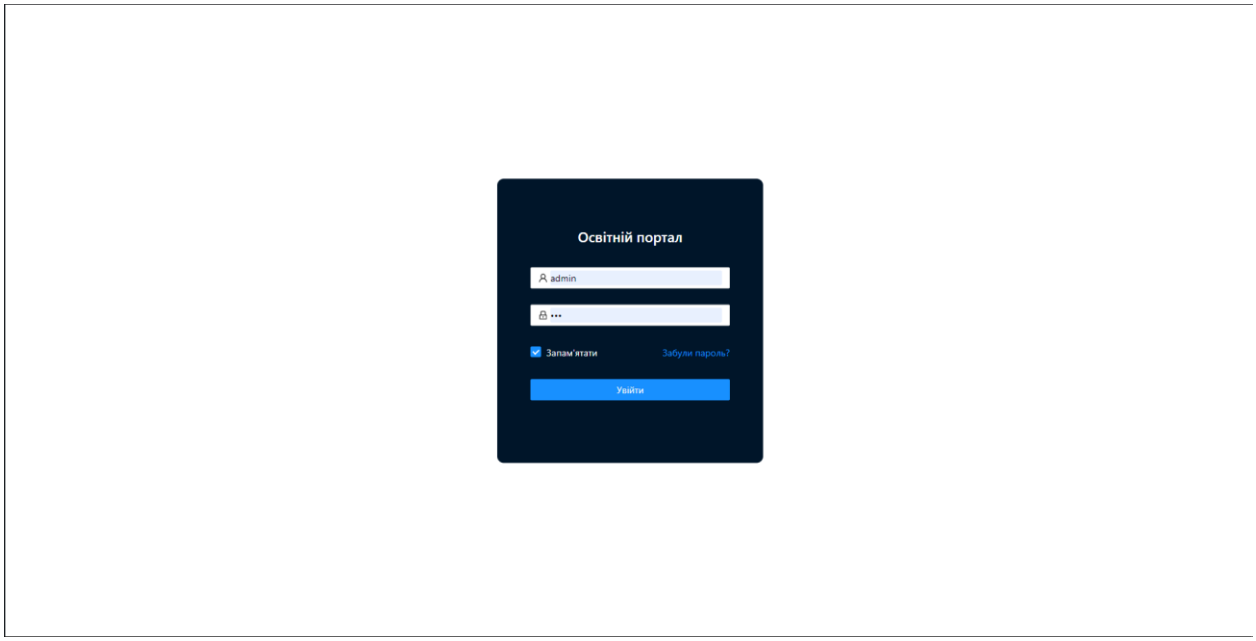


Рисунок 3.1. Сторінка «Вхід»

Після аутентифікації, як викладач, користувач потрапляє на сторінку «Мої предмети»:

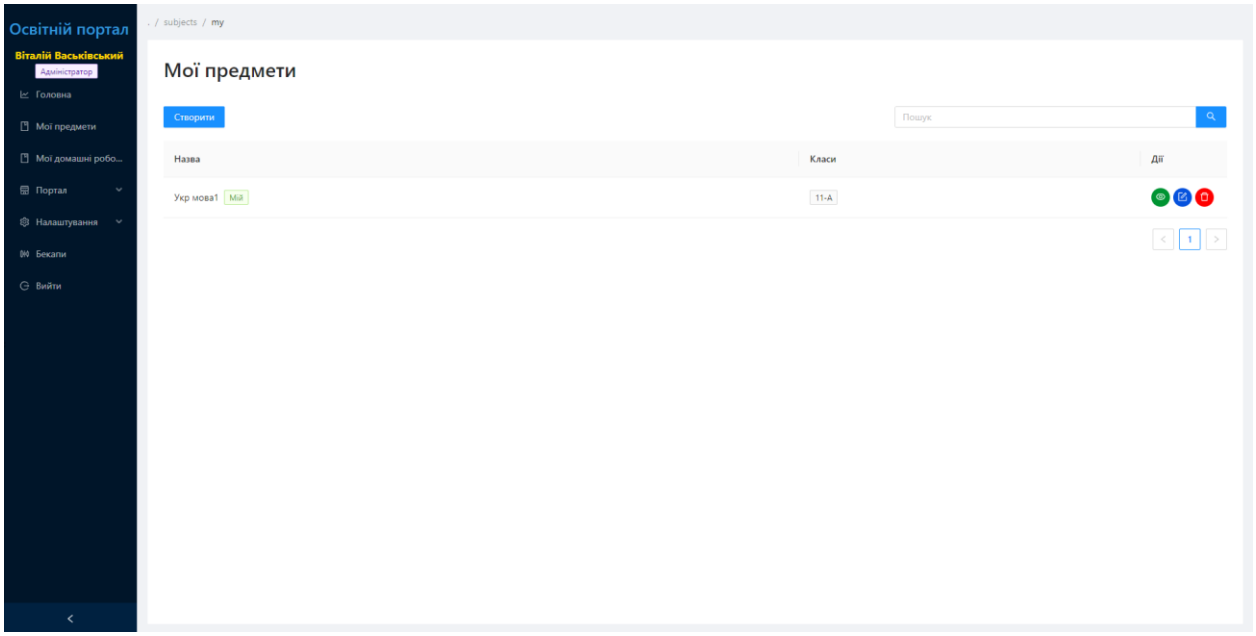


Рисунок 3.2. Сторінка «Мої предмети»

Також може створити новий предмет:

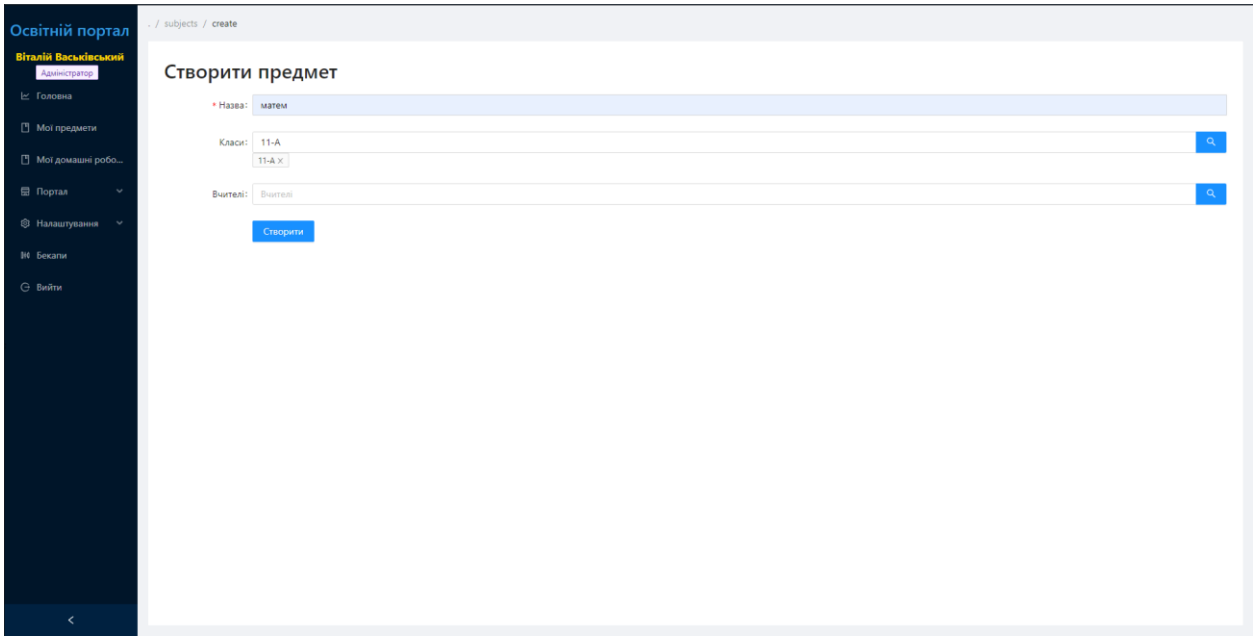


Рисунок 3.3. Сторінка «Створення предмету»

Відредагувати чи видалити свій предмет:

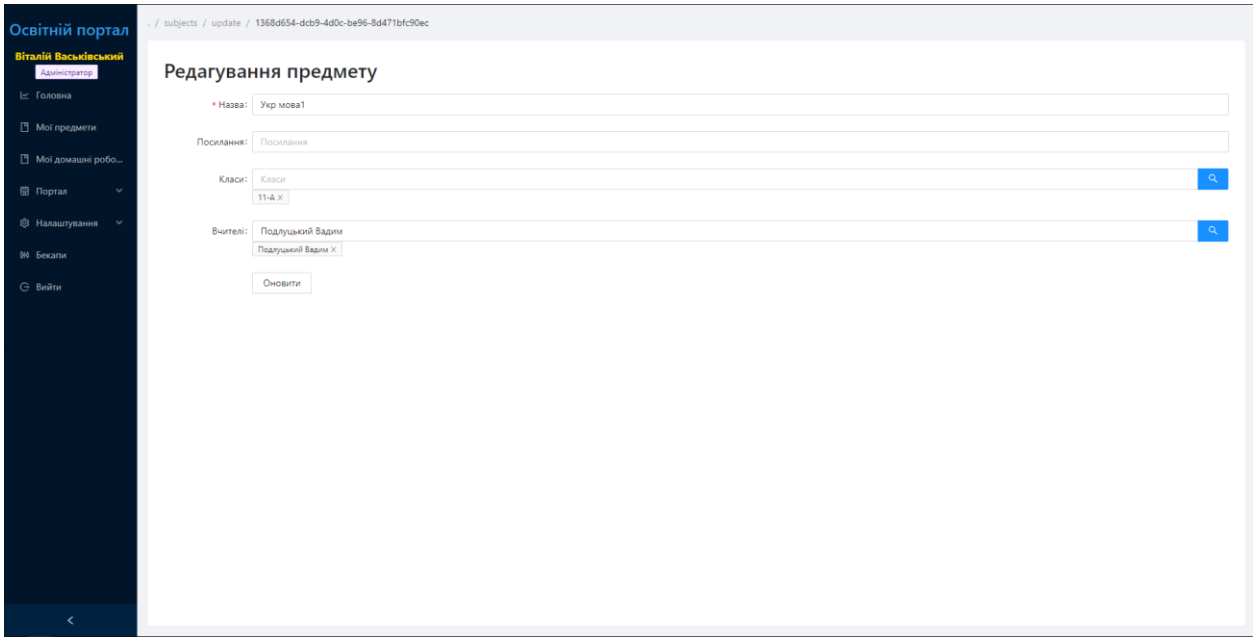


Рисунок 3.4. Сторінка «Редагування предмету»

Викладач може переглядати детальну інформацію про продмет, список постів, також керувати постами, якщо він має для цього доступ.

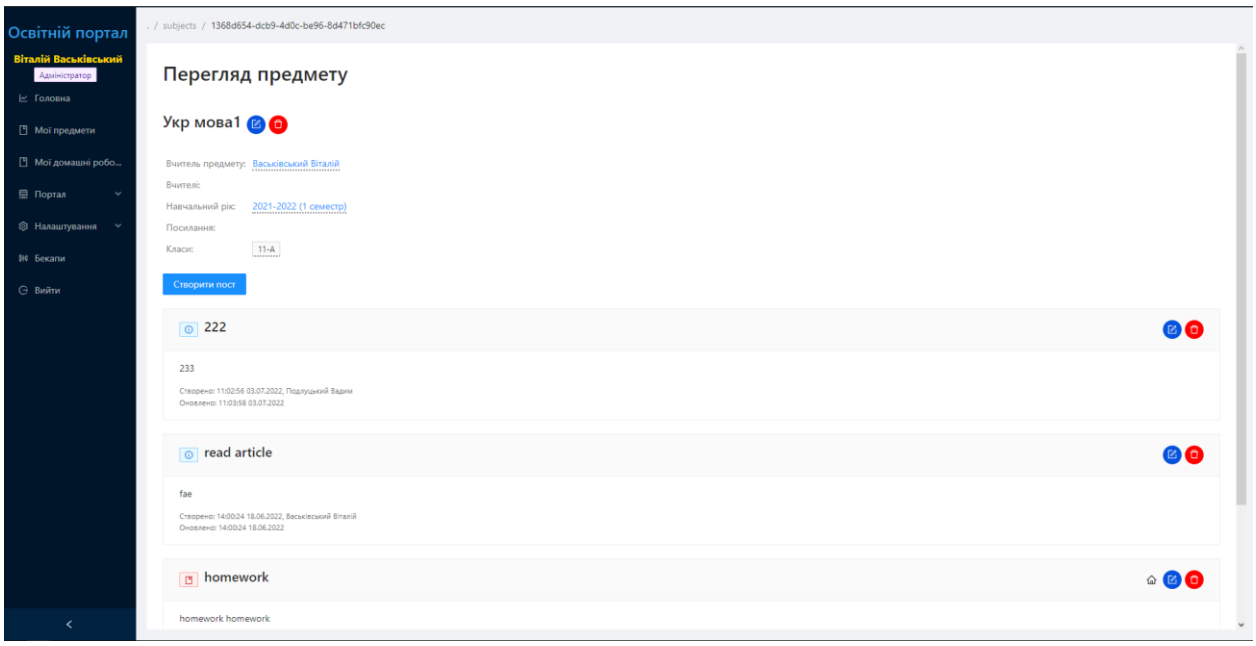


Рисунок 3.5. Сторінка «Перегляд предмету»

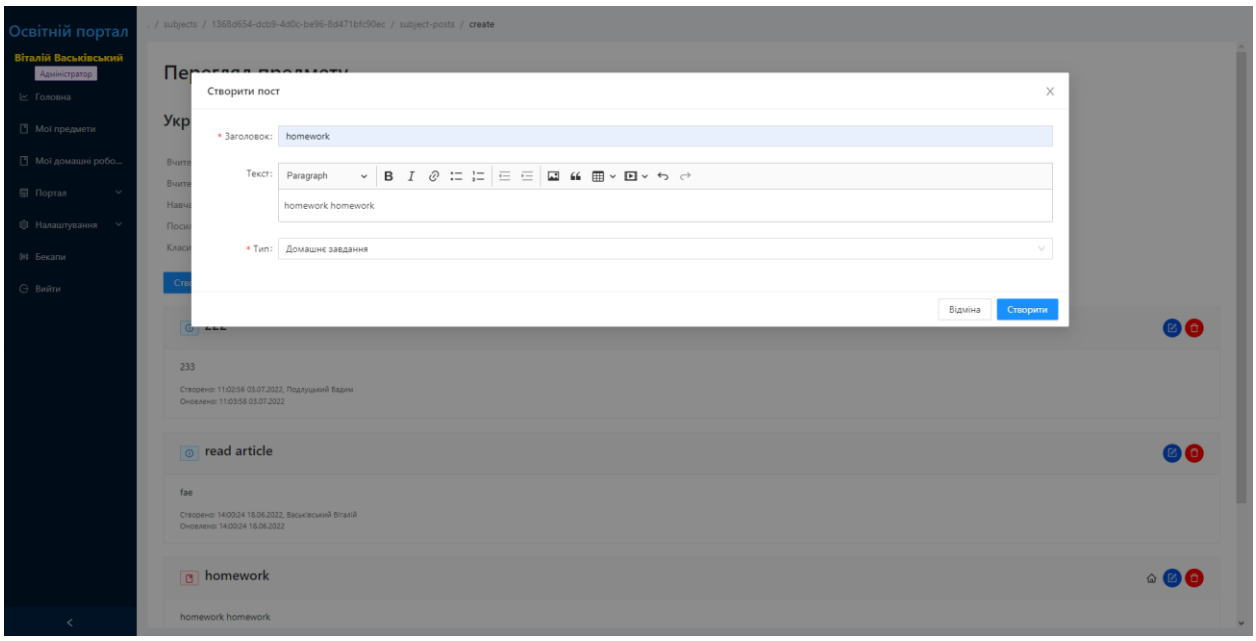


Рисунок 3.6. Сторінка «Створення посту»

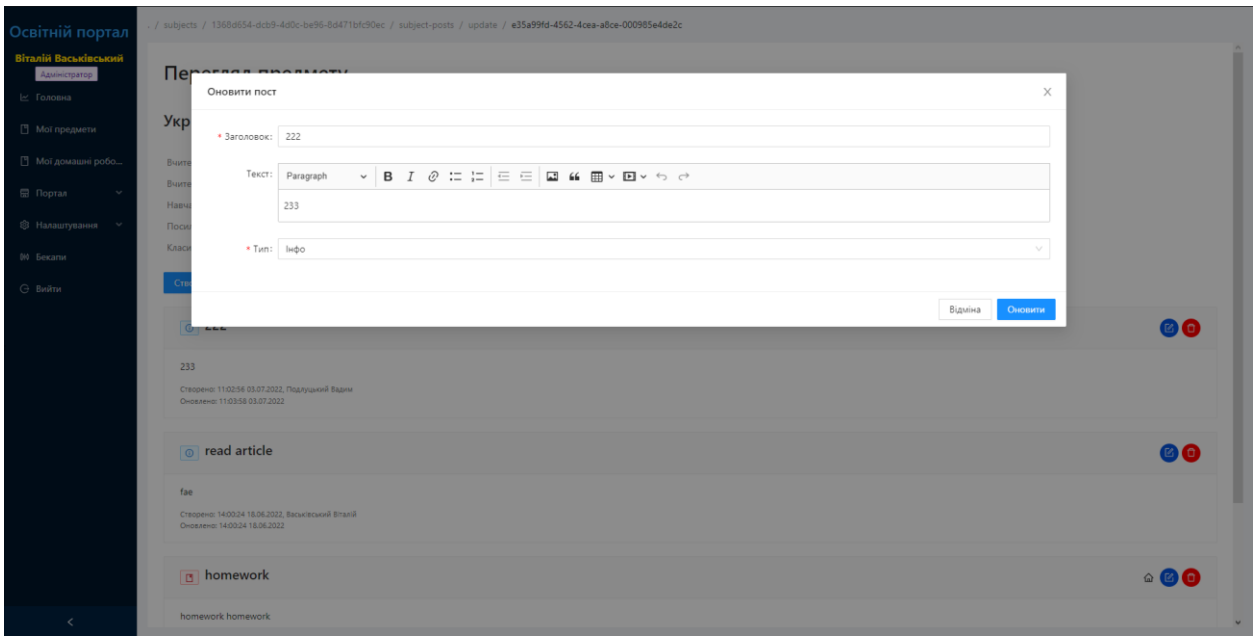


Рисунок 3.7. Сторінка «Редагування посту»

Якщо пост відноситься до типу «Домашнє завдання» то для нього можна переглянути, хто надіслав домашнє завдання

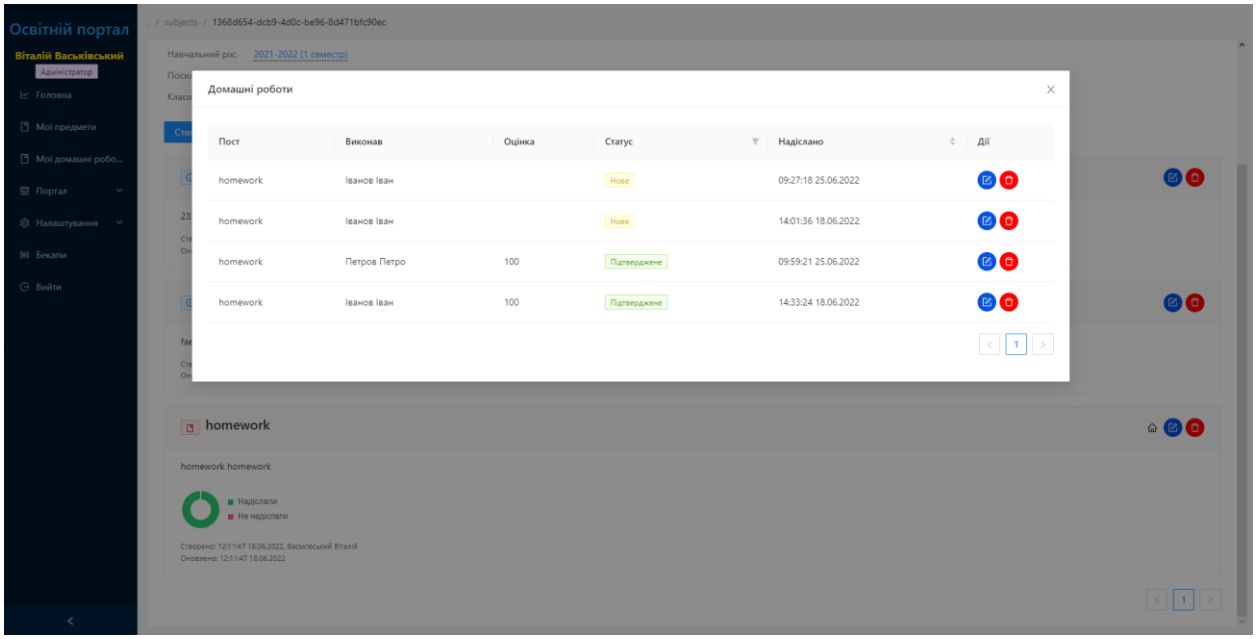


Рисунок 3.8. Сторінка «Домашні завдання для посту»

Також для постів з типом «Домашнє завдання» можна переглянути статистику скільки учнів його виконали:

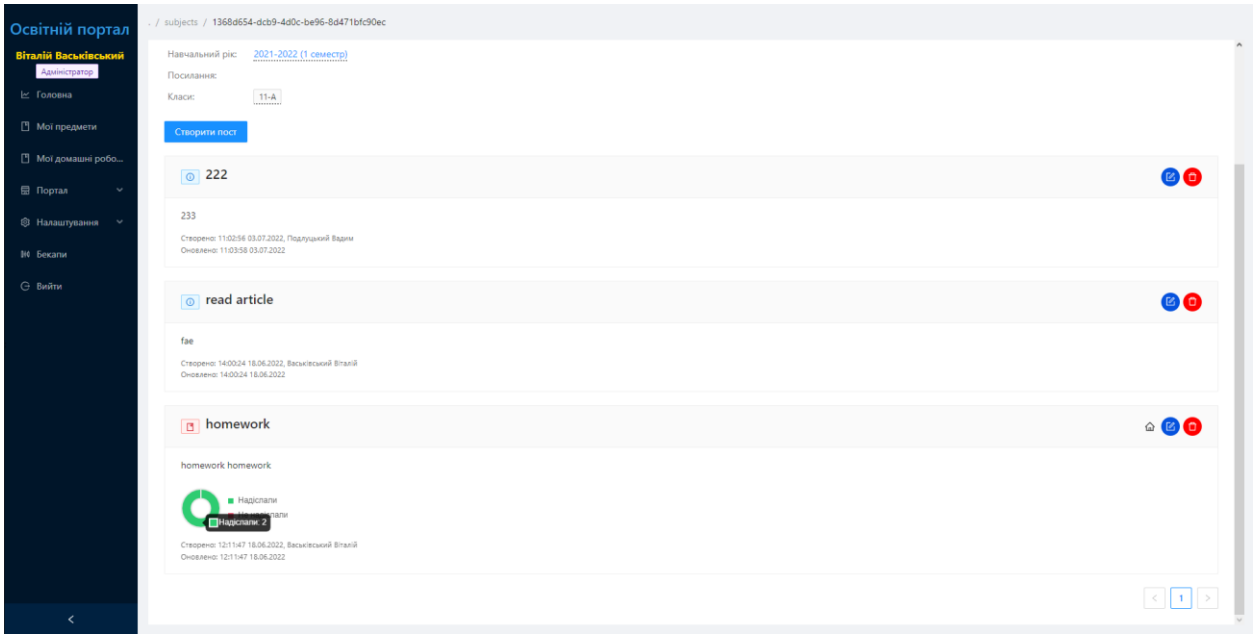


Рисунок 3.9. Сторінка «Перегляд предмету»

Також є сторінка для перегляду моїх всіх домашніх робіт, які надіслали учні. І Відповідно видалити чи відредагувати

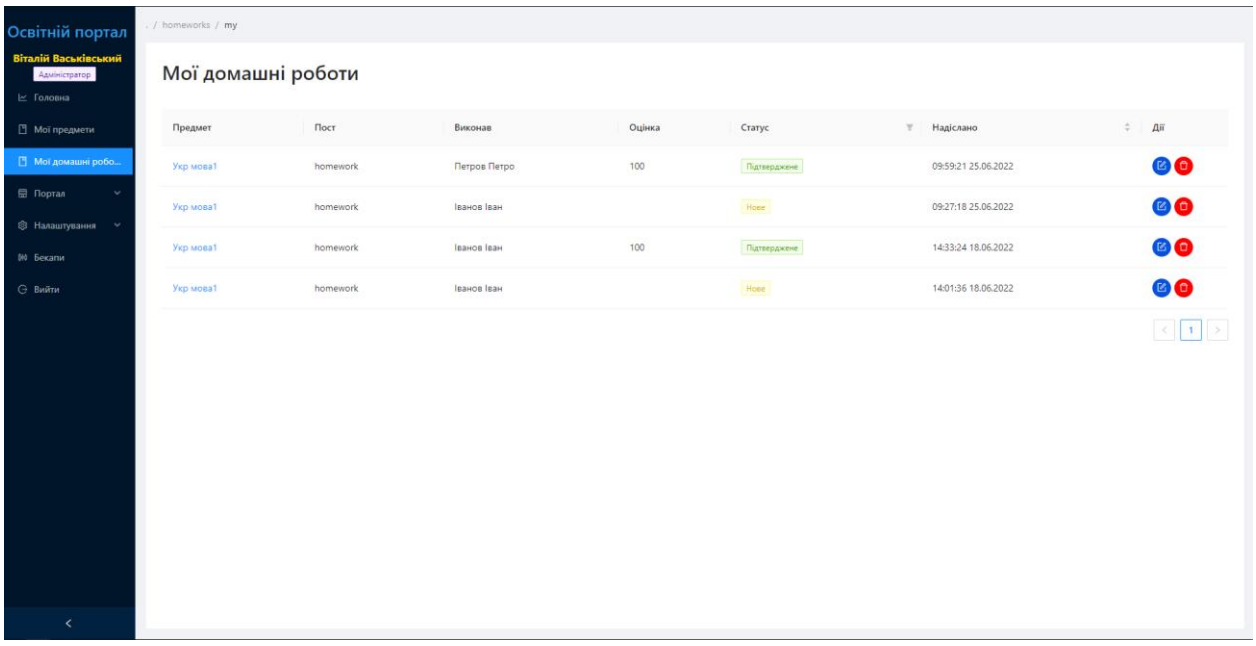


Рисунок 3.10. Сторінка «Мої домашні роботи»

Далі продемонстровано модуль класи. І відповідно CRUD операції з ними.

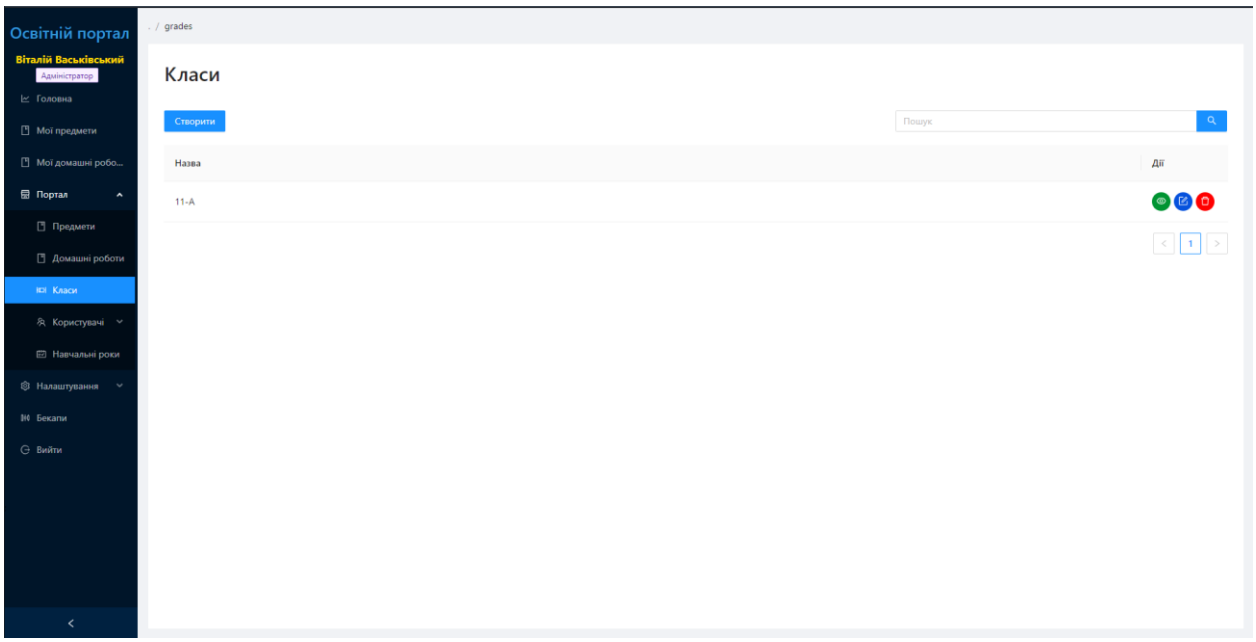


Рисунок 3.11. Сторінка модулю «Класи»

Далі продемонстровано модуль учні. І відповідно CRUD операції з ними.

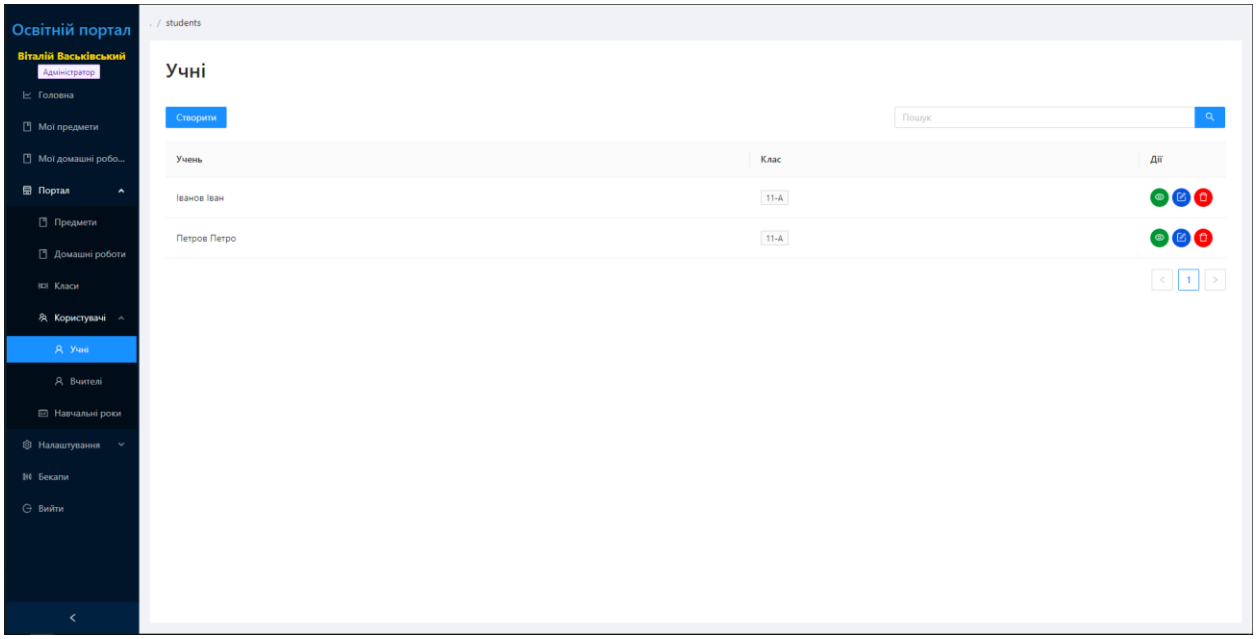


Рисунок 3.12. Сторінка модулю «Учні»

Далі продемонстровано модуль вчителі. І відповідно CRUD операції з ними.

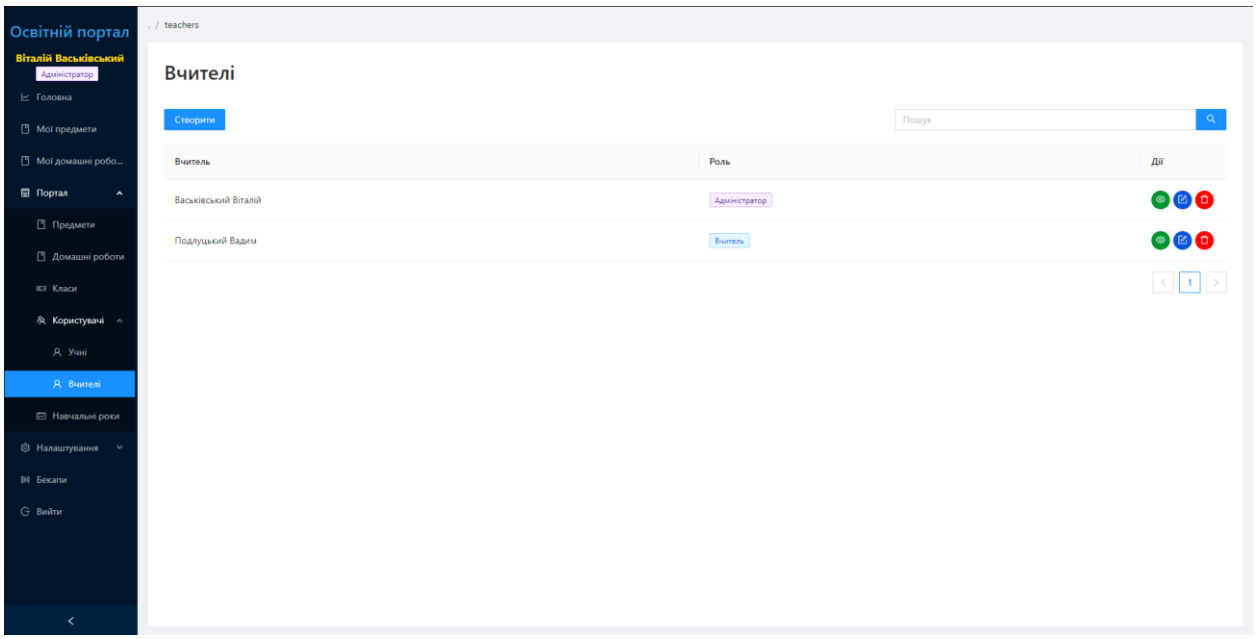


Рисунок 3.13. Сторінка модулю «Вчителі»

Далі продемонстровано модуль навчальні роки. І відповідно CRUD операції з ними.

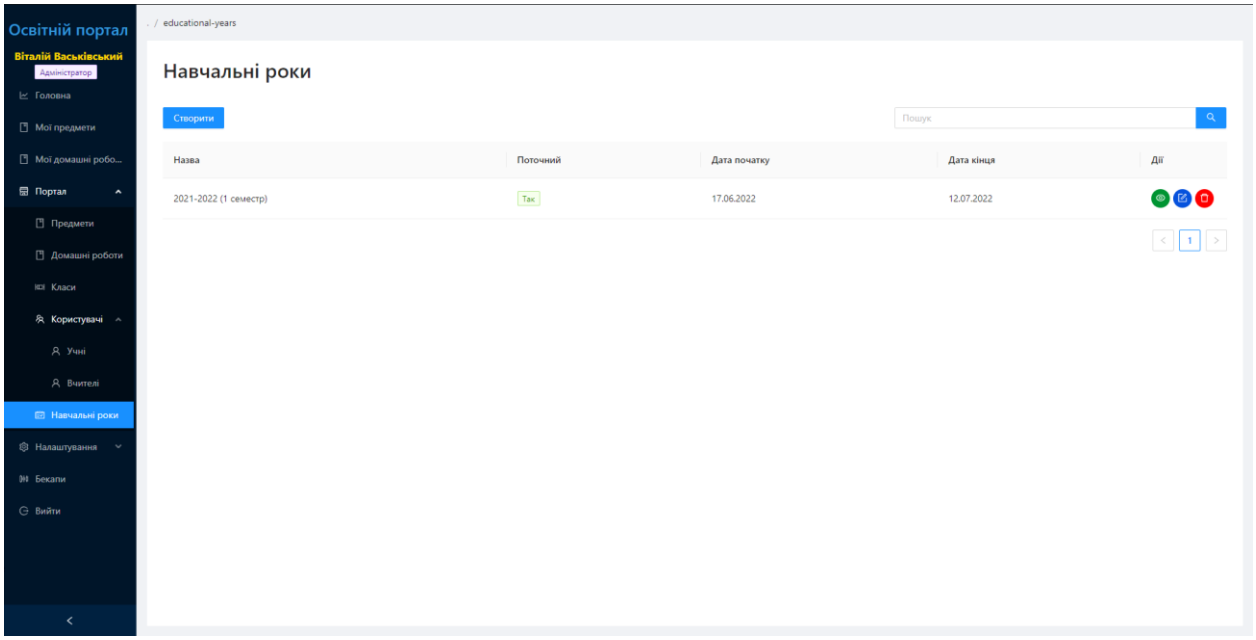


Рисунок 3.14. Сторінка модулю «Навчальні роки»

Далі продемонстровано сторінки налаштувань.

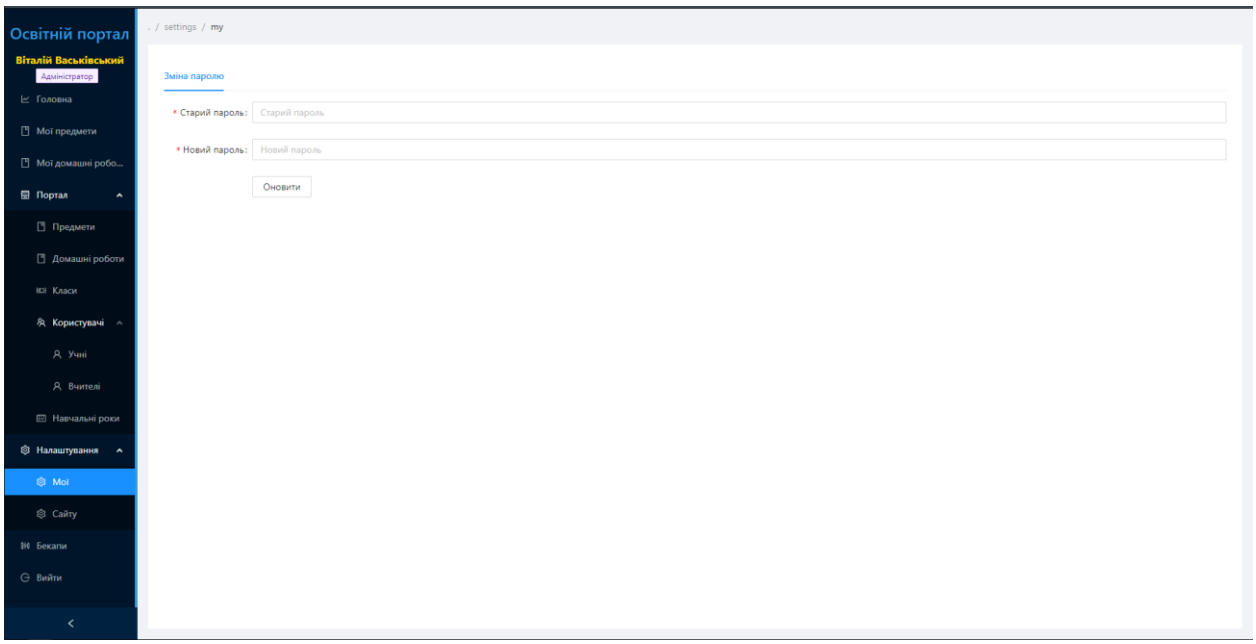


Рисунок 3.15. Сторінка «Мої налаштування»

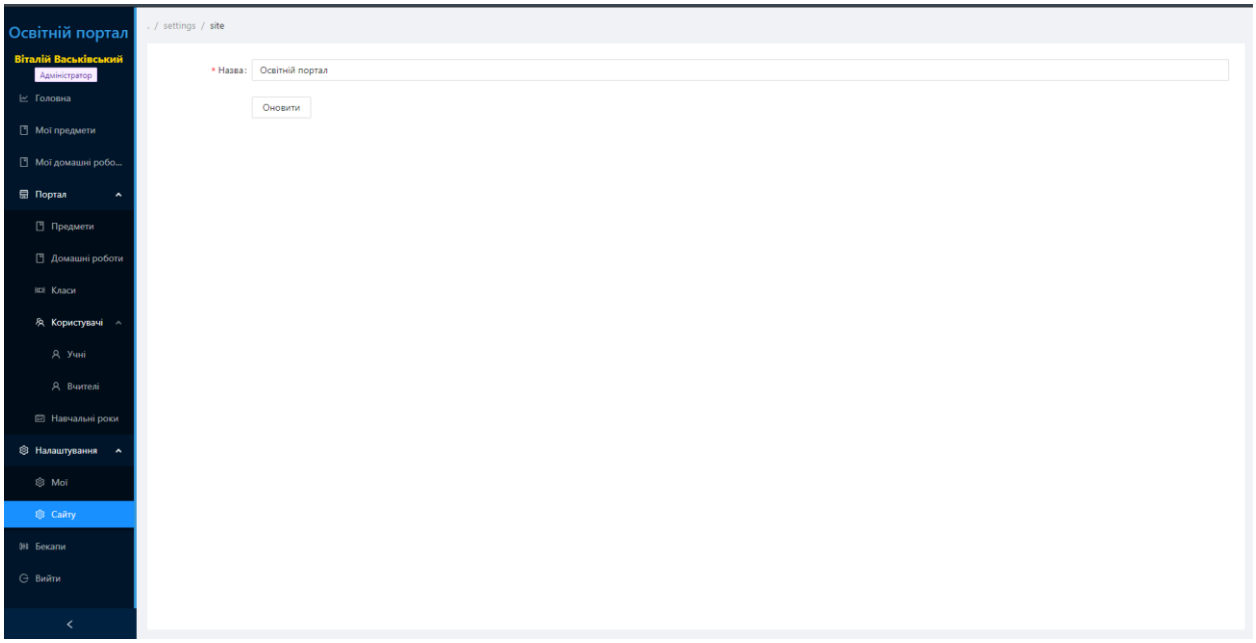


Рисунок 3.16. Сторінка «Налаштування сайту»

Також на порталі реалізовано автоматичне створення бекапів один раз на тиждень, або створення їх вручну кнопкою «Створити». Також за потреби можна натиснути кнопку «Відновлення» для відновлення до потрібного бекапу.

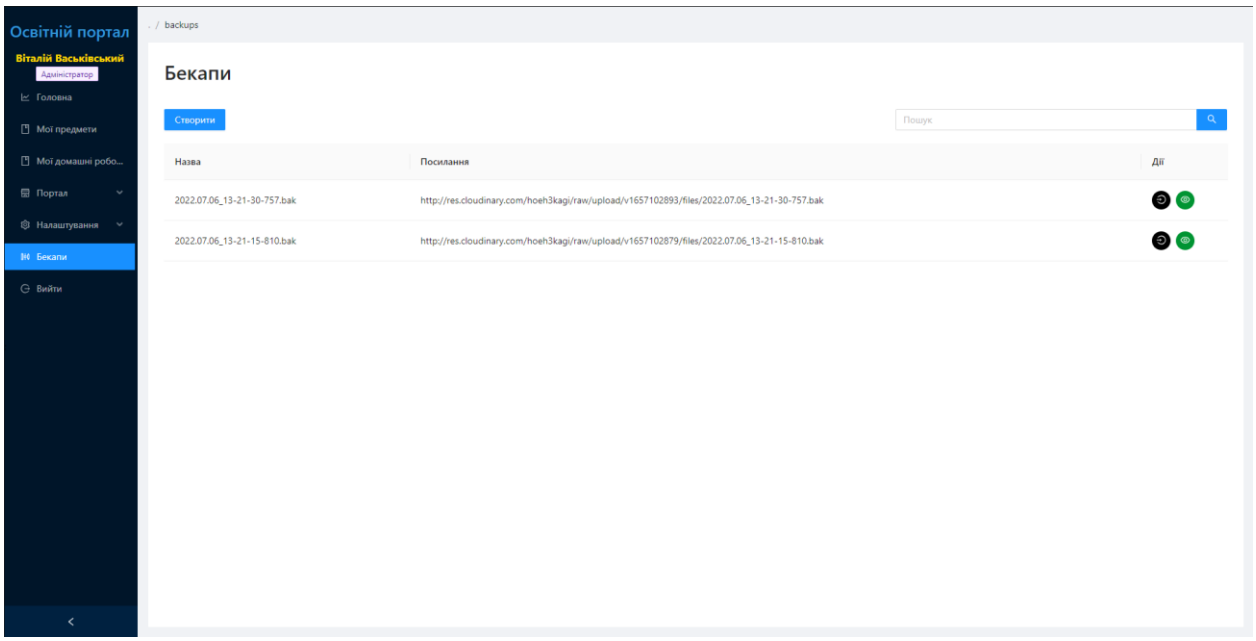


Рисунок 3.17. Сторінка «Бекапи»

Тепер увійдемо як учень. Потрапляємо на сторінку з «Моїми предметами», де можемо їх переглядати та надсилати домашні роботи.

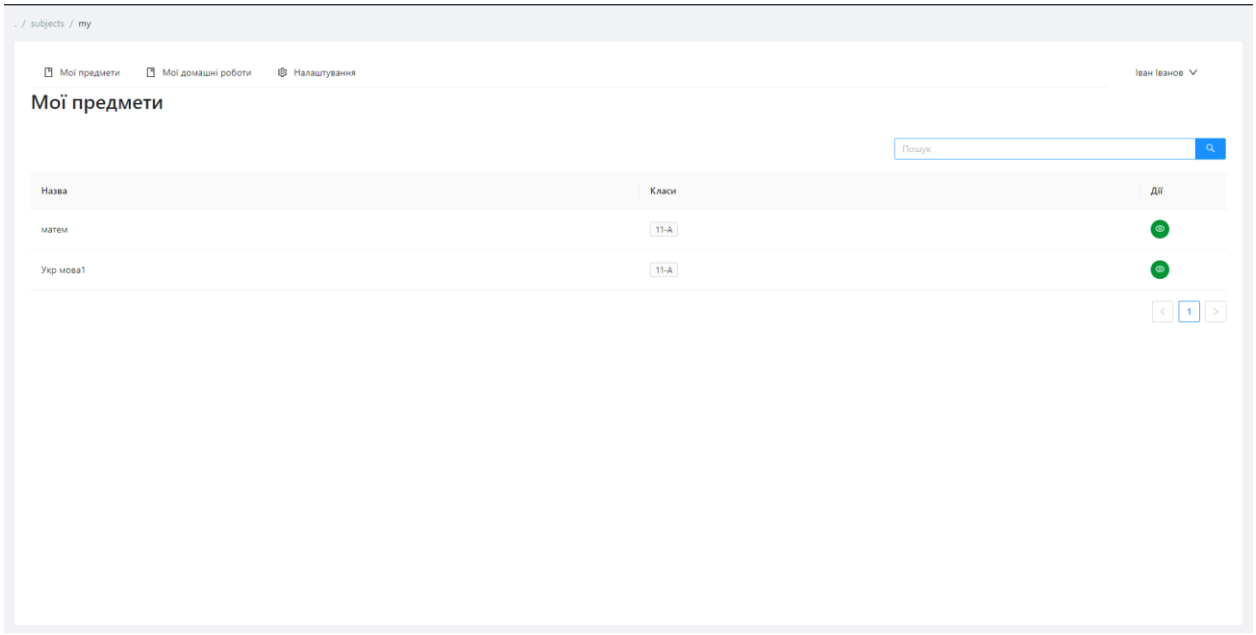


Рисунок 3.19. Сторінка «Мої предмети»

Також учень має можливість переглядати свої надіслані домашні завдання:

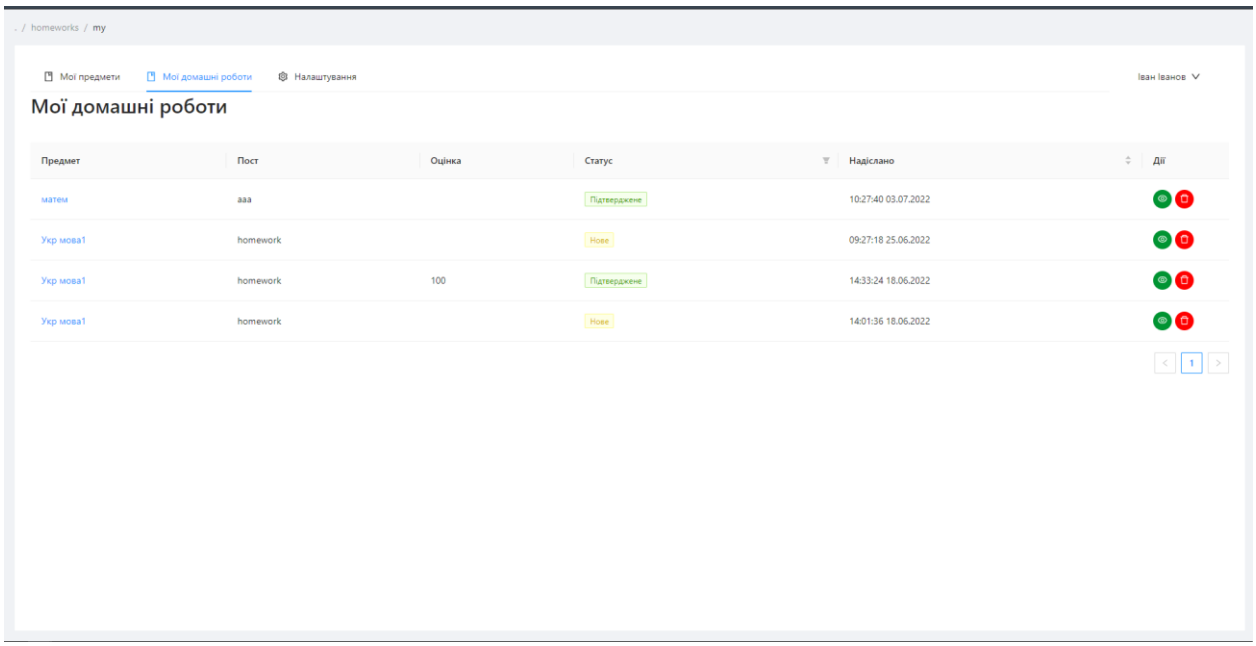


Рисунок 3.18. Сторінка «Мої домашні завдання»

Та налаштування:

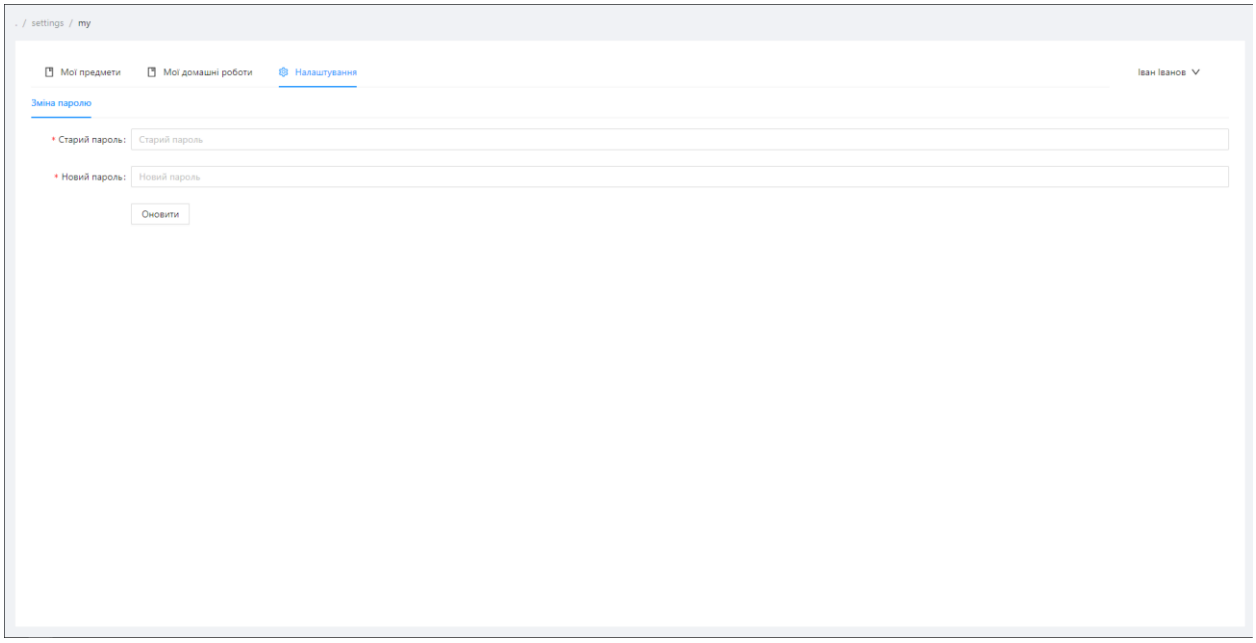


Рисунок 3.20. Сторінка «Налаштування»

### 3.2. Реалізації операцій обробки даних в БД

Оскільки будь-яка програма розпочинається з аутентифікації та контролю доступу, то потрібно розглянути, як саме створюється

користувач.

```
public override async Task<UserModel> CreateAsync(UserModel entity)
        {
            if (!string.IsNullOrEmpty(entity.Email))
            {
                List<UserModel> checkUniqeUserEmail = await GetOrDefaultAsync(e =>
e.Email == entity.Email);
                if (checkUniqeUserEmail.Count > 0)
                    throw new Exception("Користувач з введеним Email уже існує");
            }

            List<UserModel> checkUniqeUserLogin = await GetOrDefaultAsync(e =>
e.Login == entity.Login);
            if (checkUniqeUserLogin.Count > 0)
                throw new Exception("Користувач з введеним Логіном уже існує");

            await base.CreateAsync(entity);
            return entity;
        }
```

Дана функція приймає об’єкт, що містить пароль, пошту, псевдонім, ім’я, прізвище. Далі шукаємо чи користувач з даним email або логіном уже існує, якщо так то викидаємо помилку, в іншому випадку продовжуємо реєстрацію. Далі зберігаємо користувача в базі.

Тепер розглянемо метод для оновлення інформації про користувача

```
public override async Task<UserModel> UpdateAsync(UserModel newUser)
        {
            if (!string.IsNullOrEmpty(newUser.Email))
            {
                List<UserModel> checkUniqeUserEmail = await GetOrDefaultAsync(e =>
e.Email == newUser.Email && e.Id != newUser.Id);
                if (checkUniqeUserEmail.Count > 0)
                    throw new Exception("Користувач з введеним Email уже існує");
            }

            List<UserModel> checkUniqeUserLogin = await GetOrDefaultAsync(e =>
e.Login == newUser.Login && e.Id != newUser.Id);
            if (checkUniqeUserLogin.Count > 0)
                throw new Exception("Користувач з введеним Логіном уже існує");

            UserModel addedUser = await GetByIdAsync(newUser.Id);
            addedUser.FirstName = newUser.FirstName;
            addedUser.LastName = newUser.LastName;
            addedUser.MiddleName = newUser.MiddleName;
            addedUser.Email = newUser.Email;
            addedUser.Login = newUser.Login;
            addedUser.DateOfBirth = newUser.DateOfBirth;
            addedUser.Role = newUser.Role;
            addedUser.GradeId = newUser.GradeId;
            await _context.SaveChangesAsync();
            return newUser;
              }
```

В даному методі отримуємо тей самий обект, що і при реєстрації і потім шукаємо по id користувача, дані якого згодом оновлюємо.

Далі розглянемо авторизацію, що забезпечую вхід користувача.

```
public string GenerateAccessToken(Guid userId, string userLogin, UserRoleEnum
userRole)
        {
            SymmetricSecurityKey key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Environment.GetEnvironmentVariable("Auth
IssuerSigningKey")));
            SigningCredentials signingCredentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

            List<Claim> claims = new List<Claim>
            {
                new Claim(AuthClaimsIdentity.DefaultIdClaimType, userId.ToString()),
                new Claim(AuthClaimsIdentity.DefaultLoginClaimType, userLogin),
                new Claim(AuthClaimsIdentity.DefaultRoleClaimType,
userRole.ToString()),
            };
            JwtSecurityToken token = new JwtSecurityToken(
                issuer: Environment.GetEnvironmentVariable("AuthValidIssuer"),
                audience: Environment.GetEnvironmentVariable("AuthValidAudience"),
                claims: claims,
                expires: DateTime.Now.AddDays(30),
                signingCredentials: signingCredentials
            );
            return new JwtSecurityTokenHandler().WriteToken(token);
              }
```

Далі перевіряємо на співпадіння в базі, створюємо або отримуємо токен оновлення, в залежності від умов та генеруємо токен доступу, в результаті повертаємо токен доступу.

Розглянемо далі метод створення предмету.

```
  public override async Task<SubjectModel> CreateAsync(SubjectModel subject)
        {
            var currentEducationalYear = await
_educationalYearRepository.GetCurrentAsync();
            subject.EducationalYearId = currentEducationalYear.Id;
            await base.CreateAsync(subject);
            _context.Entry(subject).State = EntityState.Detached;

            SubjectModel addedSubject = await GetByIdAsync(subject.Id, s =>
s.GradesHaveAccessRead, s => s.TeachersHaveAccessCreatePosts);
            foreach (var gradeId in subject.GradesHaveAccessReadIds.Distinct())
            {
                GradeModel grade = await _gradeRepository.GetByIdAsync(gradeId);
                addedSubject.GradesHaveAccessRead.Add(grade);
            }
            foreach (var teacherId in
subject.TeachersHaveAccessCreatePostsIds.Distinct())
            {
                UserModel teacher = await _userRepository.GetByIdAsync(teacherId);
                addedSubject.TeachersHaveAccessCreatePosts.Add(teacher);
            }
            await base.UpdateAsync(addedSubject);
            return addedSubject;
              }
```

Даний метод приймає об’єкт зі всіма необхідними даними про предмет. Далі робимо перевірки та зберігаємо його.

Далі розглянемо метод оновлення предмету, який приймає тей самий об’єкт, що і попередній метод , але з Id.

```
public override async Task<SubjectModel> UpdateAsync(SubjectModel newSubject)
        {
            SubjectModel addedSubject = await GetByIdAsync(newSubject.Id, s =>
s.GradesHaveAccessRead, s => s.TeachersHaveAccessCreatePosts);
            addedSubject.Name = newSubject.Name;
            addedSubject.Link = newSubject.Link;

            addedSubject.GradesHaveAccessRead =
addedSubject.GradesHaveAccessRead.Where(g =>
newSubject.GradesHaveAccessReadIds.Any(gId => gId == g.Id)).ToList();
            foreach (var gradeId in newSubject.GradesHaveAccessReadIds.Distinct())
            {
                if (!addedSubject.GradesHaveAccessRead.Any(g => g.Id == gradeId))
                {
                    GradeModel grade = await _gradeRepository.GetByIdAsync(gradeId);
                    addedSubject.GradesHaveAccessRead.Add(grade);
                }
            }

            addedSubject.TeachersHaveAccessCreatePosts =
addedSubject.TeachersHaveAccessCreatePosts.Where(t =>
newSubject.TeachersHaveAccessCreatePostsIds.Any(tId => tId == t.Id)).ToList();
            foreach (var teacherId in
newSubject.TeachersHaveAccessCreatePostsIds.Distinct())
            {
                if (!addedSubject.TeachersHaveAccessCreatePosts.Any(t => t.Id ==
teacherId))
                {
                    UserModel teacher = await
_userRepository.GetByIdAsync(teacherId);
                    addedSubject.TeachersHaveAccessCreatePosts.Add(teacher);
                }
            }

            await _context.SaveChangesAsync();
            return addedSubject;
              }
```

Далі проводеимо перевірки і зберігаємо зміни.

Далі розглянемо метод отримання предмету. Він є generic методом, тому його можна використувати для всіх сутностей БД.

```
  public virtual async Task<GetEntitiesResponse<T>>
WhereOrDefaultAsync(Expression<Func<T, object>> predicate, Order order, int page,
Expression<Func<T, bool>>? condition = null, params Expression<Func<T, object>>[]
includes)
        {
            IQueryable<T> entities = includes.Aggregate(
                _context.Set<T>().AsQueryable(),
                (current, include) => current.Include(include));
```

```
            if (condition != null)
                entities = entities.Where(condition);

            entities = order == Order.Ascend
                ? entities.OrderBy(predicate)
                : entities.OrderByDescending(predicate);

            int total = entities.Count();

            int take = 20;
            int skip = (page - 1) * take;
            entities = entities.Skip(skip).Take(take);

            return new GetEntitiesResponse<T>
            {
                Entities = await entities.ToListAsync(),
                Total = total,
                PageSize = take,
            };
              }
```

Даний метод приймає такі параметри, як: query, skip, take, параметер сортування та порядок сортування. Тут формуємо відповідні лямбда вирази, щоб отримати дані.

Далі розглянемо метод для видалення предмету. Він є generic методом, тому його можна використувати для всіх сутностей БД.

```
public virtual async Task RemoveAsync(Guid id)
        {
            T entity = await GetByIdAsync(id);
            _context.Set<T>().Remove(entity);
            await _context.SaveChangesAsync();
        }
```

Даний метод приймає id предмету, потім шукає відповідний предмет в базі та видаляє його, що ми можемо бачити вище в коді.

### 3.3. Організація звітності системи

У даному підрозділі буде розглянуто лише ті звітності, що були розроблені спеціально для додатку, а саме статистика по виконаним домашніх роботам для посту

Лістинг наведено нижче :

```
Field<NonNullGraphType<ListGraphType<SubjectPostStatisticType>>,
IEnumerable<SubjectPostStatistic>>()
                .Name("Statistics")
                .ResolveAsync(async context =>
                {
                    if (context.Source.Type != PostType.Homework)
```

```
                    return new List<SubjectPostStatistic>();

                using var scope = serviceProvider.CreateScope();
                var homeworkRepository =
scope.ServiceProvider.GetRequiredService<IHomeworkRepository>();
                var subjectRepository =
scope.ServiceProvider.GetRequiredService<ISubjectRepository>();
                var gradeRepository =
scope.ServiceProvider.GetRequiredService<IGradeRepository>();

                var subjectId = context.Source.SubjectId;
                var subjectPostId = context.Source.Id;

                var subject = await subjectRepository.GetByIdAsync(subjectId, s
=> s.GradesHaveAccessRead);

                int studentsHaveAccessReadCount = 0;
                foreach (var gradeHaveAccessRead in
subject.GradesHaveAccessRead)
                {
                    var grade = await
gradeRepository.GetByIdAsync(gradeHaveAccessRead.Id, s => s.Students);
                    studentsHaveAccessReadCount += grade.Students.Count;
                }

                var homeworks = await homeworkRepository.GetOrDefaultAsync(h =>
h.SubjectPostId == subjectPostId);

                int sentCount = homeworks.DistinctBy(h => h.StudentId).Count();
                int notSentCount = studentsHaveAccessReadCount - sentCount;

                return new List<SubjectPostStatistic>
                {
                    new SubjectPostStatistic { Key = "Надіслали", Value =
sentCount, HashColor = "#2ecc71" },
                    new SubjectPostStatistic { Key = "Не надіслали", Value =
notSentCount, HashColor = "#FF6384" },
                };
            });
```

Даний метод нічого не приймає. В ході виконання даного методу, спочатку в відповідній таблиці бази даних відбираються записи, які відповідають заданому виразу, потім відбувається сортування по переданому полю та порядку, далі ці записи групуються по виразу групування, а вже потім вони конвертуються в потрібний обєкт.

## Висновки до третього розділу

У ході виконання даного розділу відбулось написання повного коду для взаємодії користувача з БД. Було спроектовано інтерфейс освітнього порталу. Були розроблені форми для предметів, постів предметів, домашніх

робіт, учнів, вчителів, навчальних років. Також були створені сторінки з можливістю перегляду, виконання запитів на фільтрації даних, виконання запитів на підгрузку даних. Крім цього, було створено можливість зберігати поточний стан бази, та відновлювати його з резервної копії його.

Також при виконання даного розділу було написано та описано логіку та код виконання звітності про систему за кількома типами, а саме – статистика по виконанню домашньої роботи для поста предмету

# РОЗДІЛ 4. АДМІНІСТРУВАННЯ БАЗ ДАНИХ

## 4.1. Розробка заходів захисту інформації в БД

Отже, розглянемо категорії користувачів інформаційної системи порталу: адміністратор, вчителі та учні. Найбільші права доступу до даних маєадміністратор, що необхідно для організації роботи. Він може переглядати будь-які дані даного порталу. Також головний повинен мати змогу робити бекапи та можливість відновлювати базу даних, для яких потрібні доступи до усіх таблиць. Тому доцільно створити окрему роль "Адміністратор" для забезпечення надання доступу до цих даних. Більш детально перелік об’єктів БД, доступ до яких надано ролі " Адміністратор " наведено в табл. 10. На перетині рядків і стовпців зазначено дії, які може виконувати користувач даного типу ролі: 1 – перегляд даних, 2 – редагування, 3 – видалення, 4 – повний доступ.

Таблиця 10

| | EducationalYears | Grades | Homeworks | Settings | SubjectPosts | Subjects | Users |
|---|---|---|---|---|---|---|---|
| Admin | 4 | 4 | 1 | 4 | 4 | 4 | 4 |

Окрему групу користувачів системи складають учні та вчителі. Детальніше ролевий доступ зображений в таблиці 11.

Таблиця 11

| | EducationalYears | Grades | Homeworks | Settings | SubjectPosts | Subjects | Users |
|---|---|---|---|---|---|---|---|
| Учні | 0 | 0 | 1, 2(тільки власні дз) | 0 | 1, 2,3 (тільки власні коментарі) | 1(тільки власні предмети) | 0 |
| Вчителі | 1 | 1 | 1, 2 | 0 | 1, 2,3 (тільки власні пости) | 1, 2,3 (тільки власні предмети) | 1 |

У сервісах додані відповідні політики для користувачів

`services`

```
                .AddGraphQL(options =>
                {
                    options.EnableMetrics = true;
                    options.UnhandledExceptionDelegate = (context) =>
                    {
                        Console.WriteLine(context.Exception.StackTrace);
                        context.ErrorMessage = context.Exception.Message;
                    };
                })
                .AddSystemTextJson()
                .AddGraphTypes(typeof(AppSchema), ServiceLifetime.Transient)
                .AddGraphQLAuthorization(options =>
                {
                    options.AddPolicy(AuthPolicies.Authenticated, p =>
p.RequireAuthenticatedUser());
                    options.AddPolicy(AuthPolicies.Student, p =>
p.RequireClaim(ClaimTypes.Role, UserRoleEnum.Student.ToString(),
UserRoleEnum.Teacher.ToString(), UserRoleEnum.Administrator.ToString()));
                    options.AddPolicy(AuthPolicies.Teacher, p =>
p.RequireClaim(ClaimTypes.Role, UserRoleEnum.Teacher.ToString(),
UserRoleEnum.Administrator.ToString()));
                    options.AddPolicy(AuthPolicies.Administrator, p =>
p.RequireClaim(ClaimTypes.Role, UserRoleEnum.Administrator.ToString()));

                });
```

І далі для доступу до GraphQL резолверів використано метод AuthorizedWith(Політика) для обмеження доступу.

```
        Field<NonNullGraphType<GetEntitiesResponseType<SubjectType, SubjectModel>>,
GetEntitiesResponse<SubjectModel>>()
                .Name("GetSubjects")
                .Argument<NonNullGraphType<IntGraphType>, int>("Page", "Argument for
get Subjects")
                .Argument<NonNullGraphType<StringGraphType>, string>("Like",
"Argument for get My Subjects")
                .ResolveAsync(async context =>
                {
                    int page = context.GetArgument<int>("Page");
                    string like = context.GetArgument<string>("Like");
                    EducationalYearModel currentEducationalYear = await
educationalYearRepository.GetCurrentAsync();
                    return await subjectRepository.WhereOrDefaultAsync(s =>
s.CreatedAt, Order.Descend, page, s =>
                        s.EducationalYearId == currentEducationalYear.Id
                        && s.Name.ToLower().Contains(like.ToLower())
                    );
                })
                      .AuthorizeWith(AuthPolicies.Teacher);
```

### 4.2. Налаштування параметрів роботи MS SQL Server.

Для підклювення MS SQL server до ASP.NET потрібно отримати з віддаленого сервера стрічку підклювення, далі передаєму її в зміні оточення проекту, а потім уже робимо переконстрюювання в стрічка, задовільняє потреби метода бібліотеки, яка дає змогу підключення до БД. Код переконструювання наведено нижче :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {

optionsBuilder.UseSqlServer(Environment.GetEnvironmentVariable("MS_SQL_DATABASE_URL"
) ?? @"Data Source=(localdb)\ProjectModels;Initial
Catalog=EducationalPortal;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mu
ltiSubnetFailover=False");

        }
```

Для створення віддаленої бази MS SQL потрібно зареєструватися на somee, після цього творити базу даних.

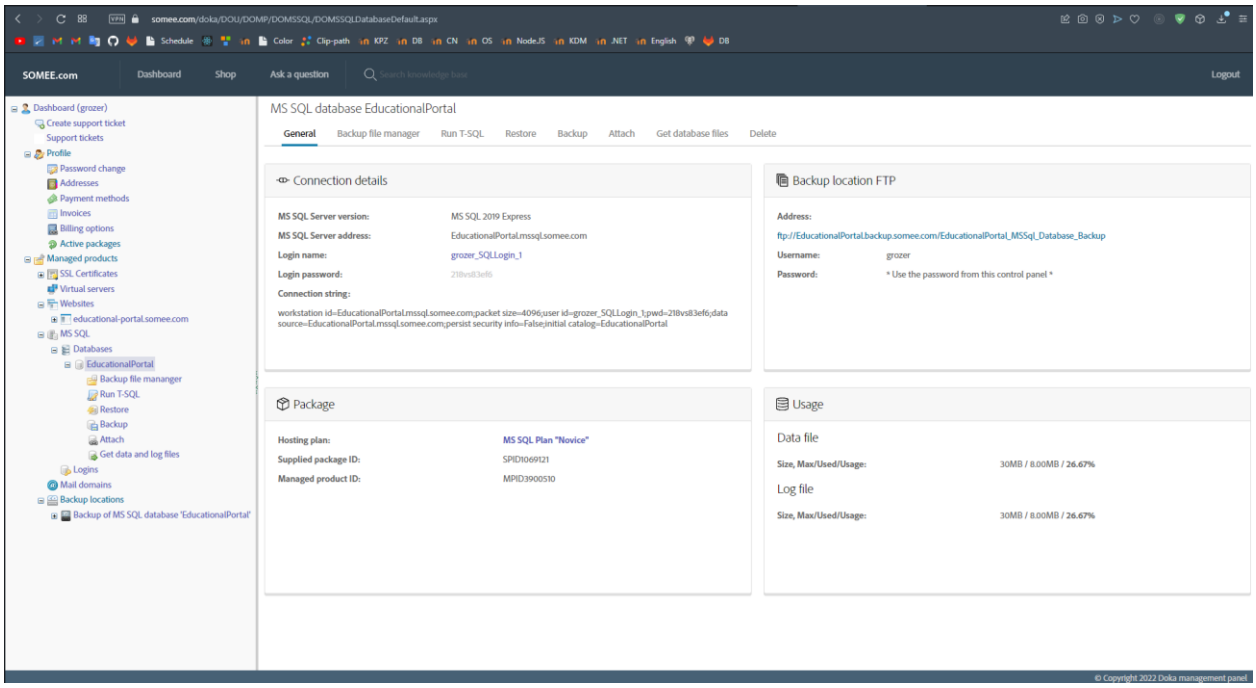


Рис 4.1. Somee.com

Також варти мати на увазі, оскільки дана база є безкоштовною то має свої обмеження, а саме сторення бекапів через sql код.

**Висновки до четвертого розділу**

У ході виконання даного розділу проведено налаштування прав доступу користувача до бази, а саме контроль за допомогою ролей та певних контролюючих доступ методів. Також у ході виконання цього розділу було

показано, як саму відбулось підключення додатку до БД, створення та наштування віддаленої бази даних MS SQL Server.

# ВИСНОВКИ

Під час написання даної курсової роботи було отримано навички роботи з віддаленими базами даних, створення резервних копій та відновлення з них даних баз даних.

В першому розділі курсової роботи, було проаналізовано три СУБД, визначено переваги кожної з СУБД та обрано одну для реалізації додатку. Також було визначено головні функції програми та приблизну логіку доступу до даних.

В другому розділі було проаналізовано інформаційні процеси, спроектовано структуру бази даних, описано приблизну логіку функцій для роботи з базою на клієнті, пояснено деяку логіку обробки, збереження та генерації даних для бази. Також було описано як буде відбуватись побудова та генерація статистичної інформації платформи.

В третьому розділі відбулось написання повного коду для взаємодії користувача з БД та спроектовано інтерфейс освітнього порталу. Крім цього, було створено можливість зберігати поточний стан бази, та відновлювати його з резервної копії. Також було написано та описано логіку та код виконання звітності системи.

В четвертому розділі проведено налаштування прав доступу користувача до бази, а саме контроль за допомогою ролей та певних контролюючих доступ методів. Також у ході виконання цього розділу було показано, як саму відбулось створення, наштування та підключення додатку до віддаленої бази даних MS SQL Server.

В результаті виконання курсової роботи отримано освітній портал для школи, який повністю відповідає запланованому функціоналу, умовам та вимогам, поставленим на початку проектування, підтримуючи резервне копіювання бази даних та певну статистику.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Документація React [електронний ресурс]. Режим доступу: https://reactjs.org/ .
2. Документація Redux [електорнний ресурс]. Режим доступу: https://redux.js.org/ .
3. Документація Axios [електорнний ресурс]. Режим доступу: https://axios-http.com/docs/intro.
4. Документація Ant Design [електорнний ресурс]. Режим доступу: https://ant.design/components/overview/ .
5. Документація Apollo Client [електорнний ресурс]. Режим доступу: https://www.apollographql.com/docs/react/.
6. Документація ASP.NET [електорнний ресурс]. Режим доступу: https://dotnet.microsoft.com/en-us/apps/aspnet .
7. Документація MS SQL [електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/sql/?view=sql-server-ver16 .
8. Документація Graph QL .NET [електорнний ресурс]. Режим доступу: https://graphql-dotnet.github.io.
9. Документація Entity Framework [електорнний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/ef.
10. Стаття «Авторизація за допомогою JWT-токенів» [електорнний ресурс]. Режим доступу: https://metanit.com/sharp/aspnet5/23.7.php.
11. Список відтворення «Курс «React JS путь самурая 1.0, уроки практика» » [електорнний ресурс]. Режим доступу: https://www.youtube.com/playlist?list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8 .

# ДОДАТКИ

Додаток А

Лістинг класу BaseRepository, який є базовим класом для всі репозиторіїв:

```
using EducationalPortal.Business.Abstractions;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;

namespace EducationalPortal.MsSql.Abstractions
{
    public abstract class BaseRepository<T> : IBaseRepository<T> where T : BaseModel
    {
        private readonly AppDbContext _context;

        public BaseRepository(AppDbContext context)
        {
            _context = context;
        }

        public virtual Task<T> GetByIdAsync(Guid? id, params Expression<Func<T,
object>>[] includes)
        {
            Task<T?> entity = GetByIdOrDefaultAsync(id, includes);
            if (entity == null)
                throw new Exception($"Не знайдено {typeof(T).Name.Replace("Model",
"")}");
            return entity;
        }

        public virtual Task<T?> GetByIdOrDefaultAsync(Guid? id, params
Expression<Func<T, object>>[] includes)
        {
            return includes.Aggregate(_context.Set<T>().AsQueryable(),
                (current, include) => current.Include(include))
                    .FirstOrDefaultAsync(e => e.Id == id);
        }

        public virtual Task<List<T>> GetAsync(params Expression<Func<T, object>>[]
includes)
        {
            Task<List<T>> entities = GetOrDefaultAsync(includes);
            if (entities == null)
                throw new Exception($"Не знайдено {typeof(T).Name.Replace("Model",
"")}");
            return entities;
        }

        public virtual Task<List<T>> GetOrDefaultAsync(params Expression<Func<T,
object>>[] includes)
        {
            return includes.Aggregate(_context.Set<T>().AsQueryable(),
               (current, include) => current.Include(include))
                    .ToListAsync();
        }

        public virtual async Task<List<T>> GetAsync(Expression<Func<T, bool>>
condition, params Expression<Func<T, object>>[] includes)
        {
            List<T> entities = await GetOrDefaultAsync(condition, includes);
            if (entities == null || entities.Count() == 0)
                throw new Exception($"Не знайдено {typeof(T).Name.Replace("Model",
"")}");
            return entities;
        }
```

```
        public virtual Task<List<T>> GetOrDefaultAsync(Expression<Func<T, bool>>
condition, params Expression<Func<T, object>>[] includes)
        {
            return includes.Aggregate(_context.Set<T>().AsQueryable(),
                (current, include) => current.Include(include))
                .Where(condition).ToListAsync();
        }

        public virtual async Task<GetEntitiesResponse<T>>
WhereAsync(Expression<Func<T, object>> predicate, Order order, int page,
Expression<Func<T, bool>>? condition = null, params Expression<Func<T, object>>[]
includes)
        {
            GetEntitiesResponse<T> getEntitiesResponse = await
WhereOrDefaultAsync(predicate, order, page, condition, includes);
            if (getEntitiesResponse == null || getEntitiesResponse.Total == 0)
                throw new Exception($"Не знайдено {typeof(T).Name.Replace("Model",
"")}");
            return getEntitiesResponse;
        }

        public virtual async Task<GetEntitiesResponse<T>>
WhereOrDefaultAsync(Expression<Func<T, object>> predicate, Order order, int page,
Expression<Func<T, bool>>? condition = null, params Expression<Func<T, object>>[]
includes)
        {
            IQueryable<T> entities = includes.Aggregate(
                _context.Set<T>().AsQueryable(),
                (current, include) => current.Include(include));

            if (condition != null)
                entities = entities.Where(condition);

            entities = order == Order.Ascend
                ? entities.OrderBy(predicate)
                : entities.OrderByDescending(predicate);

            int total = entities.Count();

            int take = 20;
            int skip = (page - 1) * take;
            entities = entities.Skip(skip).Take(take);

            return new GetEntitiesResponse<T>
            {
                Entities = await entities.ToListAsync(),
                Total = total,
                PageSize = take,
            };
        }

        public virtual async Task<T> CreateAsync(T entity)
        {
            _context.Entry(entity).State = EntityState.Added;
            await _context.Set<T>().AddAsync(entity);
            await _context.SaveChangesAsync();
            return entity;
        }

        public virtual async Task<T> UpdateAsync(T entity)
        {
            _context.Entry(entity).State = EntityState.Modified;
            _context.Set<T>().Update(entity);
            await _context.SaveChangesAsync();
            return entity;
```

```
        }

        public virtual async Task RemoveAsync(Guid id)
        {
            T entity = await GetByIdAsync(id);
            _context.Set<T>().Remove(entity);
            await _context.SaveChangesAsync();
        }
    }
}
```