НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту
допущено”
Завідувач
кафедри ЦТЕ
Наталія АУШЕВА
“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*
на тему:

«Веб-система пошуку та рейтингування навчальних закладів»

Виконав:
студент IV курсу, групи ТР-01

Яценко Мирослав Ігорович
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
к.т.н., доц. Тихоход Володимир Олександрович
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент: _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

Нормоконтролер: *старший викладач Беспала Ольга Миколаївна* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

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

Київ – 2024

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
Наталія АУШЕВА

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту

ЯЦЕНКУ Мирославу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Веб-система пошуку та рейтингування навчальних закладів»

керівник роботи
Тихоход Володимир Олександрович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____»_____2024 р. №_____

2. Термін подання роботи студентом ***07.06.2024 року***

3. Вихідні дані до роботи

Процес дослідження застосунків для пошуку навчальних закладів, який включає: визначення аналогів, встановлення їх переваг та недоліків, визначення необхідного функціоналу. Розробка власної веб-системи. Вона складається з таких етапів: розробка серверної частини, розробка клієнтської частини, розробка бази даних, тестування.

4. Зміст роботи

Пошук систем зі схожим функціоналом, визначення необхідного функціоналу для покриття потреб користувачів. Дослідження технологій для

створення власної веб-системи із пошуку та рейтингування навчальних закладів.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу зображення, що демонструють роботу розробленої веб-системи, архітектура системи, бази даних, взаємодію користувачів із системою.
6. Дата видачі завдання 15 вересня *2023 р.*

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент _____ **Мирослав ЯЦЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Володимир ТИХОХОД**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 64 сторінках, містить 38 ілюстрацій, 18 таблиць, 1 додаток, 19 джерел в переліку посилань.

Мета роботи – створення веб-системи пошуку навчальних закладів для школярів та студентів.

Методи та засоби: мова програмування C#, ASP.NET Core для створення API та серверної частини застосунку, Azure Blob Storage для зберігання медіа файлів та їх відображення, Google Maps та Google Maps API для відображення та вибору локацій, Microsoft SQL Server для керування базою даних, Blazor WebAssembly для розробки клієнтських веб-застосунків.

Результат – веб-система що має необхідний функціонал для спрощення пошуку та рейтингування навчальних закладів для школярів.

Ключові слова: веб-система, платформа .NET, технологія ASP.NET Core, Blazor WebAssembly

ANNOTATION

The thesis consists of 64 pages, contains 38 illustrations, 18 tables, 1 appendix, 19 sources in the list of references.

The purpose of the work: create a web system for searching educational institutions for schoolchildren and students.

Methods and tools: C# programming language, ASP.NET Core for creating API and backend of the application, Azure Blob Storage for storing media files and displaying them, Google Maps and Google Maps API for displaying and selecting locations, Microsoft SQL Server for database management , Blazor WebAssembly for developing client web applications.

The result: web system that has the necessary functionality to simplify the search and rating of educational institutions for schoolchildren.

Keywords: web system, .NET platform, ASP.NET Core technology, Blazor WebAssembly.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

.NET – платформа-технологія від компанії Microsoft для створення різноманітних застосунків, таких як: консольні застосунки, мобільні застосунки, веб-системи, десктоп додатки.

ASP.NET Core – частина платформи .NET, яка спрямована на створення веб-додатків та веб-сервісів.

API – Application Programming Interface.

Blazor – інтерфейсна веб-платформа .NET, спрямована на створення клієнтських веб-застосунків.

DTO – Data Transfer Object

SPA – Single Page Application.

SDK – Software Development Kit.

IDE – Integrated Development Environment.

WASM – WebAssembly.

СКБД – система керування базами даних.

ВСТУП

Якість навчання дітей відіграє важливу роль у вихованні успішної особистості. Саме тому батькам та школярам необхідно відповідально ставитися до вибору навчального закладу.

Пошук та вибір навчальних закладів для навчання дітей є складною задачею в наш час, оскільки в процесі вибору необхідно врахувати багато різних аспектів, які можуть вплинути на процес навчання. Основним показником, який може якісно вплинути на цей процес є рейтинг закладу сформований учнями, які мають досвід навчання в ньому.

Рейтингування навчальних закладів є важливим інструментом для конкурентності та визначення якості освітніх послуг. Рейтингування навчального закладу за різними критеріями дозволяє виконати вибір закладу під можливості та спроможності учня або студента. На рейтинг навчального закладу впливають різні фактори, зокрема вимогливість педагогів, результати учнів у різноманітних олімпіадах та заходах, стан ремонту та інфраструктури,

Допомогти батькам та учням обрати той заклад, що підходить їм за їх критеріями вибору, а також посилити конкуренцію серед закладів може веб-система, що реалізує описаний функціонал.

Сучасні системи мають ряд недоліків, зокрема обмеженість складових, які формують рейтинг закладу, недостатність фільтрів пошуку, відсутність відгуків та оцінок учнів.

Отже метою роботи є розробка веб-системи пошуку навчальних закладів для школярів.

Результати виконаної роботи було апробовано на XXI-й міжнародній науково-практичній конференції молодих вчених та студентів «Сучасні проблеми наукового забезпечення».

1 ЗАДАЧА РОЗРОБКИ ВЕБ-СИСТЕМИ ПОШУКУ ТА РЕЙТИНГУВАННЯ НАВЧАЛЬНИХ ЗАКЛАДІВ

Метою роботи є створення системи пошуку навчальних закладів для школярів та студентів.

Розробка системи для покращення вибору навчального закладу є важливим завданням для освітнього процесу підростаючого покоління. Дана система повинна вирішити одразу дві проблеми. Перша з них - це дати можливість закладам розповісти про нього, оскільки сьогодення вимагає від них активного застосування інформаційних технологій для популяризації. Другою є спрощення процесу пошуку навчального закладу для учнів.

Для створення ефективної системи необхідно проаналізувати та визначити головні задачі для створюваної веб-системи. Для цього варто розглянути існуючі рішення, визначити їх переваги і недоліки, після цього визначити функціонал для системи.

1.1 Постановка прикладної задачі

До основних функціональних вимог до системи слід віднести:

- система повинна забезпечувати публічну обмежену функціональність для незареєстрованих користувачів та розширену функціональність для зареєстрованих категорій користувачів;
- права доступу користувачів повинні бути розмежовані за наступними ролями: незареєстрований користувач, абітурієнт, учень, модератор, адміністратор та супер адміністратор;
- для незареєстрованого користувача повинен бути доступний наступний функціонал:
    - форма запиту на реєстрацію учня (надається фото учнівського квитка);

- форма запиту на реєстрацію адміністрації закладу освіти (надається фото довідки з відділу кадрів про займану посаду);

- для абітурієнта повинен бути доступний наступний функціонал:
  - пошук навчальних закладів відповідно до встановлених фільтрів;
  - перегляд досягнень, коментарів та оцінок закладів освіти;
  - перегляд загального рейтингу закладу та рейтингу за певними категоріями;
- учень додатково до дій абітурієнта повинен мати можливість:
  - залишати коментарі та оцінки навчальних закладів;
  - видаляти або редагувати залишені ним коментарі та оцінки;
- адміністрація навчального закладу може виконувати наступні функції:
  - подавати заявку на додавання закладу до системи;
  - відповідати на коментарі залишені до його закладів;
  - актуалізує інформаціє про свій заклад освіти;
  - переглядати розширену аналітику його закладів;
- модератор відповідає за наступні дії:
  - дозволяє або відхиляє відгук до публікації;
  - слідкує за дотриманням політик та регламентів користувачами системи;
  - видаляє будь-які коментарі та оцінки;
- адміністратор може:
  - розглядає заявки на додавання закладу (підтвердити/відхилити)
  - додавати заклади для будь-якого користувача без подачі заявки;
  - видаляє або редагує будь-які заклади;
- Супер адміністратор відповідає за розподілення прав користувачів;
- система повинна реалізовувати алгоритми рейтингування.

До основних нефункціональних вимог до системи слід віднести:

- система повинна бути розділена на наступні компоненти:
  - веб-служби REST API;
  - веб-застосунок SPA;

- база даних закладів освіти;
- індексоване сховище для повнотекстового пошуку;
- сховище мультимедійних файлів (фото-відео файлів).

• для реалізації веб-служб REST API слід використати технологію ASP.NET Core Web API;

• для реалізації веб-застосунку рекомендується проаналізувати та розглянути технологію Blazor;

• для зберігання мультимедійних файлів рекомендується розглянути використання хмарного сховища Azure Storage або іншого подібного;

• в якості сховища закладів освіти рекомендується використовувати СКБД Microsoft SQL Server з використанням підходу «спочатку код» (Code First);

• веб-застосунок повинен надавати можливість пошуку та перегляду закладів на карті, для реалізації картографічної компоненти слід використати сервіс Google Карти та інтерфейс прикладного програмування Google Карти API.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

• провести дослідження щодо порівняння функціональності подібних систем, визначення їх переваг та недоліків;

• провести дослідження щодо найбільш популярних пошукових запитів, фільтрів, визначити набір критеріїв;

• сформувати структуру запитів та модель проблемної області;

• розробити алгоритми загального рейтингування та рейтингування за групами, категоріями закладів освіти;

• вибрати набір показників рейтингування навчальних закладів та розробити алгоритми рейтингування цих показників.

1.2 Огляд методів розробки веб-системи

Серед методів розробки веб-системи варто окремо розглянути методи розробки та тестування.

Для розробки веб-системи було використано підхід Agile та методологію Scrum. Він є підходом до розробки та управлінням проєктами та передбачає розподіл проєкту на етапи. Розробка відбувається за певним циклом, який займає від тижня до місяця, і складається з декількох етапів. Даний підхід було обрано тому що вона дозволяє динамічно встановлювати задачі та гарно підходить для розробки продукту під час переддипломної практики, оскільки можна зручно поділити на цикли довжиною в тиждень[8].

Цикл розробки продукту називається спринт та складається з таких етапів [9]:

- вибір робочих елементів для спринту. На цьому етапі власник продукту, scrum master та команда розробників обговорюють мету, яку повинно досягнути за спринт;
- створення плану роботи. На даному етапі виконується встановлення завдань, які необхідно виконати для досягання мети, їх розподіл між розробниками;
- щоденні стендапи. Стендап – це невеликі зустрічі на яких команда спілкується про статуси виконання задач;
- огляд спринту. Це можливість команди продемонструвати отриманий результат;
- ретроспектива спринту. Даний етап має на меті визнати сфери вдосконалення для наступного спринту.

На рисунку 1.1 наведено графічне відображення етапів спринту.

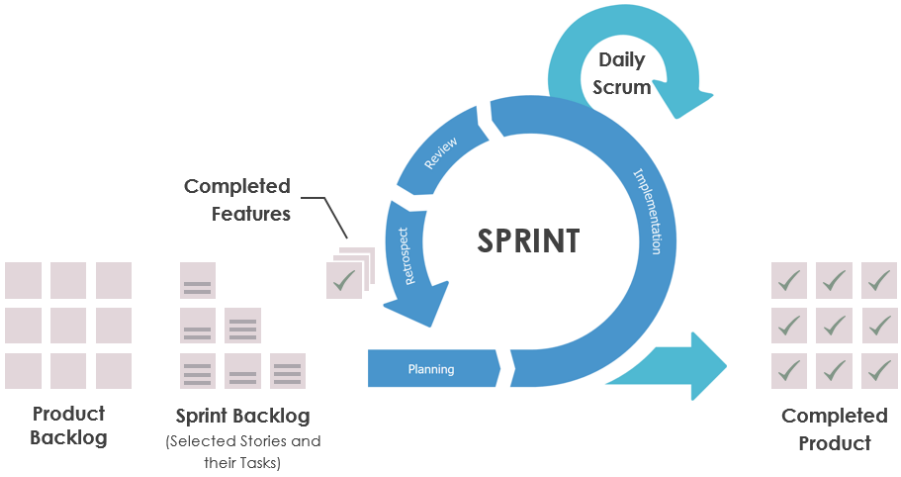

Рисунок 1.1 – Графічне відображення етапів спринту

Для тестування отриманих результатів було використано мануальне тестування. Воно поділяється на такі види:

- функціональне тестування. На даному етапі виконується перевірка відповідності програмного забезпечення вимогам специфікації та тестування основних функцій;
- нефункціональне тестування. Відбувається перевірка продуктивності, масштабованості та зручності використання;
- тестування безпеки. Перевіряється захищеність програмного забезпечення від загроз та несанкціонованого доступу.

Мануальне тестування відбувається за такими етапами:

- аналіз вимог. Цей етап включає вивчення специфікації та вимог програмного забезпечення;
- розробка тестових випадків. Створення наборів даних та умов для перевірки функціоналу;
- підготовка тестового середовища. Налаштування всього необхідного для проведення тестування;
- перевірка тестових випадків. Виконання тест-кейсів, документація результатів, виявлення дефектів;
- повторне тестування. Відбувається після виправлення дефектів.

2 РОЗГЛЯД ЗАСОБІВ РОЗРОБКИ ВЕБ-СИСТЕМИ

Розвиток технологій для розробки інформаційних систем надзвичайно швидкий у наш час, і на початку створення системи дуже важливо обрати ті технології, які матимуть довготривалу підтримку, гарну швидкодію і зручні системи для їх використання. При виборі важливо проаналізувати існуючі рішення, визначити їх переваги та недоліки, для того щоб в майбутньому користувачі мали змогу користуватися якісним продуктом.

2.1 Засоби розробки серверного застосунку

2.1.1 Мова програмування C# та платформа .NET

Для створення масштабованої веб-системи необхідно було обрати мову програмування, що підтримує розробку такого типу додатків. На рисунку 2.1 наведено рейтинг найпопулярніших мов програмування станом на 2024 рік[17]. Основними вимогами під час вибору мови програмування були: строга статична типізація, оновлення версій та підтримка від виробника. До таких вимог підпадали Typescript, Java, C#. Враховуючи зручність розробки системи та інтеграції сторонніх сервісів, було обрано мову програмування C# та платформу .NET.

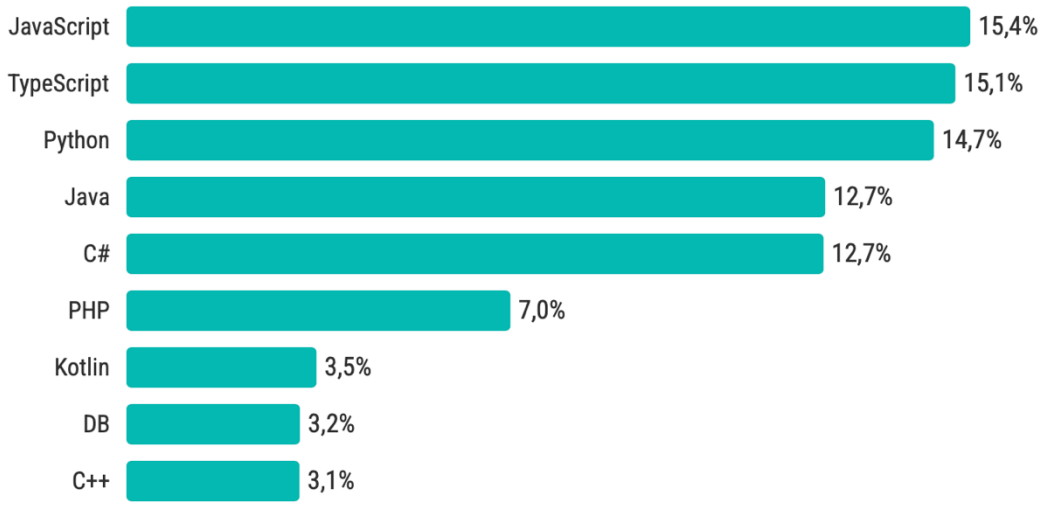

Рисунок 2.1 – Рейтинг мов програмування станом на 2024 рік

C# (C sharp) – це сучасна об’єкто-орієнтована мова програмування від компанії Microsoft розроблена у 2000 році, яка разом із F# та Visual Basic входять до платформи .NET [3].

Відповідно до стандарту ECMA-334, при розробці мови C# були поставлені такі вимоги та цілі:

- мова програмування повинна бути простою, сучасною, об’єктно-орієнтованою;
- мова повинна бути портативна;
- повинна бути підтримка безпечних принципів програмування, а саме: перевірка типів, виявлення спроб використання неініціалізованих змінних, перевірка меж масиву, автоматичне очищення зайнятої пам’яті.

В результаті отримано мову C# синтаксис якої схожий на Java або C++ [1]. Мова має статичну строгу типізацію, перевантаження операторів, поліморфізм, наслідування, інкапсуляцію, асинхронність, події, винятки тощо.

Найпопулярнішими середовищами програмування є:

- Visual Studio;
- Visual Studio Code;
- Rider;
- MonoDevelop.

В свою чергу, .NET є платформою представлена у 2002 році для розробки різноманітного програмного забезпечення. Основними типами додатків створюваними за допомогою цієї платформи є: веб-додатки, десктопні застосунки, мобільні додатки, хмарні сервіси та ігри[2].

Основними компонентами платформи .NET є:

- CLR (Common Language Runtime): компонент платформи на якому виконується код усіх мов платформи, виконується управління пам’яттю, збірка сміття;
- .NET libraries: набір бібліотек, що забезпечують розробку та роботу застосунку;
- інструменти для розробки настільних додатків WPF, WinForms;

- технології для розробки веб-систем: ASP.NET, ASP.NET Core;
- засоби для роботи з базами даних: ADO.NET, Entity Framework;
- інструменти розгортання та запуску додатків на хмарному сервісі Azure.

2.1.2 Технологія ASP.NET Core

Платформа ASP.NET Core є технологією розробленою корпорацією Microsoft яка призначена для розробки веб-застосунків різних типів. Розробка цієї технології почалась у 2014 році, а першу версію побачив світ у 2016 році. У 2019 з'явилась ASP.NET Core 3.1. Наразі є актуальною версія ASP.NET Core 6.0.

ASP.NET Core є крос-платформною та може бути розгорнута на усіх популярних операційних системах. Для цього можна використовувати IIS, або крос-платформний Kestrel. Також варто зазначити, що ASP.NET Core є фреймворком з відкритим вихідним код який розміщено на GitHub[5].

Поточну архітектуру платформи ASP.NET Core зображено на рисунку 2.2. На найвищому рівні побачимо різноманітні можливості взаємодії із користувачем. Це технології побудови інтерфейсу застосунку як MVC, Razor Pages, SPA, Blazor. Окрім цього також представлені послуги у вигляді вбудованих HTTP API, бібліотеки SignalR та сервісів gRPC [4].

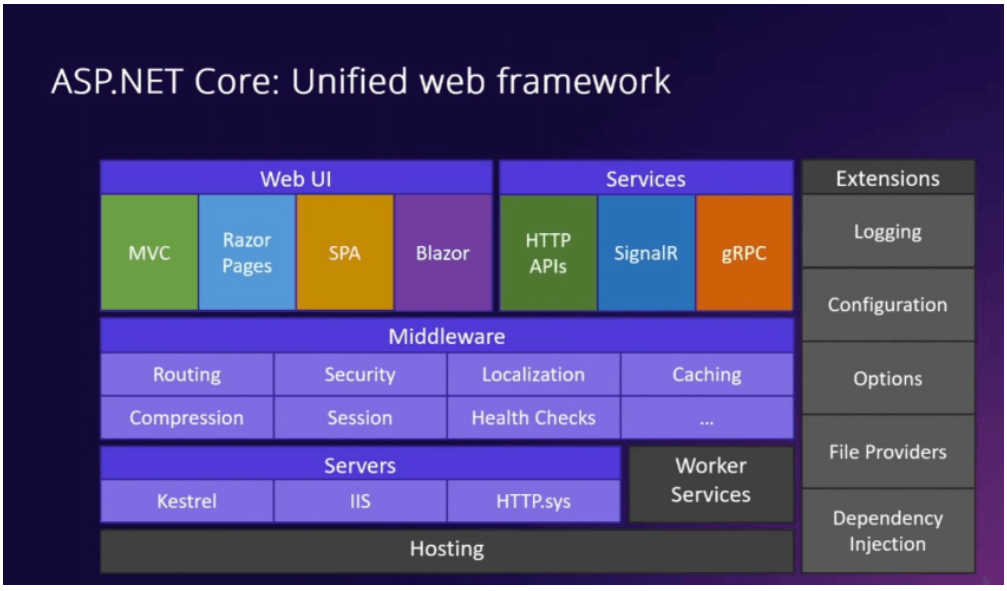

Рисунок 2.2 – Поточна архітектура платорми ASP.NET Core

Усі перераховані технології взаємодіють із ASP.NET Core представленим у вигляді різноманітних middleware-компонентів необхідних для обробки запиту.

На нижньому рівні додаток ASP.NET Core працює як певний веб-сервер, IIS, Kestrel, HTTP.sys.

Розглянемо основні моделі розробки програми ASP.NET Core [4]:

- базовий ASP.NET Core. Має увесь необхідний функціонал роботи веб-застосунку: маршрутизація, логування, робота з джерелами інформації тощо. Також було додано Minimal API що ще спростило процес написання коду;
- ASP.NET Core MVC. Працює поверх ASP.NET Core, представляє побудову застосунку навколо трьох головних компонентів M – Model (модель), V – View (представлення), C – Controller (контролер). На рисунку 2.3 наведено графічне представлення алгоритму роботи даної моделі;
- Razor Pages. Представляє модель у якій за обробку запиту відповідає сторінки Razor Pages. Кожну окрему сутність можна порівнювати з окремою веб-сторінкою;
- ASP.NET Core Web API є реалізацію патерну REST, де за кожним типом HTTP запиту призначено окремий ресурс. Цими ресурсами є методи контроллерів;
- Blazor представляє фреймворк для створення користувацьких програм як на стороні серверу так і на стороні клієнта. Технологія бульш детально буде розглянута у розділі 2.2.1.

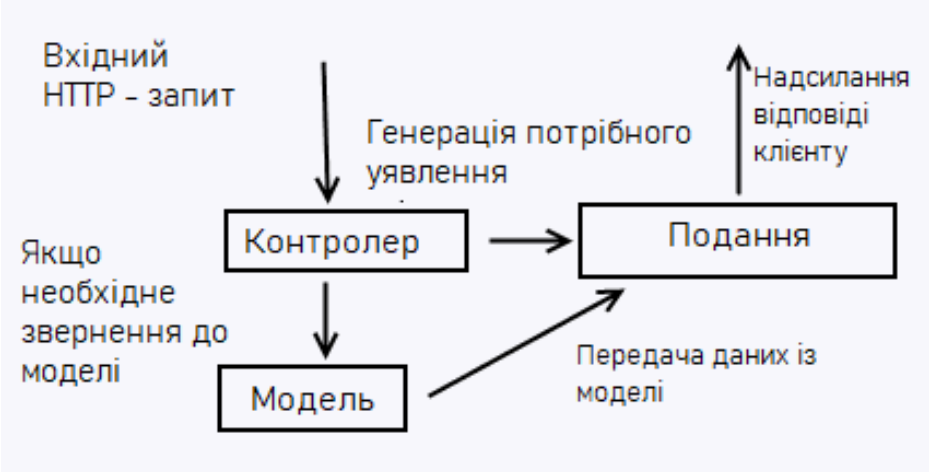

Рисунок 2.3 – Представлення алгоритму роботи моделі ASP.NET Core MVC

Підсумовуючи, варто виділити такі особливості ASP.NET Core:

- працює поверх .NET тому дозволяє використовувати увесь його функціонал;
- офіційно доступна підтримка мов C# та F#;
- великий інструментарій для розробки застосунків;
- вбудована підтримка Dependency Injection;
- модульний конвеєр HTTP-запитів;
- розширюваність реалізації;
- кросплатформеність.

2.1.3 Система керування базами даних Microsoft SQL Server та технологія Entity Framework Core

Для створення бази даних, що зберігатиме дані збережені у системі, необхідно обрати системи керування базами даних. Основними вимогами до СКБД є: можливість створювати реляційні бази даних, мати високу продуктивність та надавати високий степінь безпеки даних. Рейтинг найпопулярніших СКБД наведено на рисунку 2.4[18]. Усі найпопулярніші мови відповідають нашим вимогам, проте вибір було зроблено на користь Microsoft SQL Server, враховуючи те, що розробником є компанія Microsoft, яка також є власником обраної раніше платформи .NET. Такий вибір спростить інтеграцію та використання бази даних у застосунку.

Microsoft SQL Server – це система керування реляційними базами даних розроблена компанією Microsoft. Застосунки та засоби підключаються до екземпляру або бази даних SQL Server та взаємодіють за допомогою Transact-SQL (T-SQL) [6].

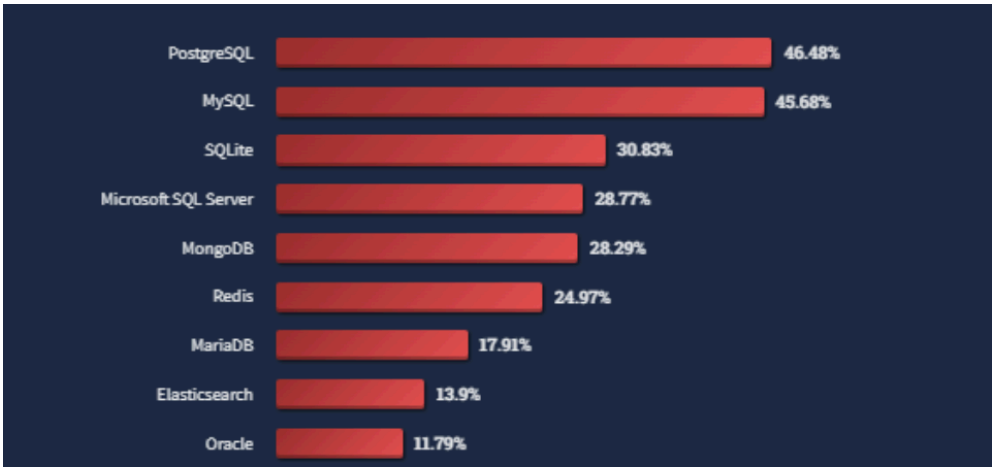

Рисунок 2.4 – Рейтинг найпопулярніших СКБД

До основних характеристик можемо віднести:

- SQL Server є реляційною СКБД, це означає, що дані мають чітку структуру та зберігаються у таблицях;
- транзакційна обробка, система підтримує транзакції, тим самим забезпечуючи цілісність даних і запобігаючи їх пошкодженню;
- підтримка збережених процедур і функцій;
- підтримка тригерів;
- підтримка індексів.

Основними компонентами системи керування є:

- ядро СКБД, являє собою основну службу для зберігання, обробки та забезпечення безпеки даних. Забезпечує контрольований доступ та обробку транзакцій відповідно до вимог усіх програм;
- служби машинного навчання. Підтримують інтеграцію машинного навчання з використанням популярних мов R і Python у корпоративні процеси;
- служби інтеграції. SQL Server Integration Services – це платформа для створення рішень для інтеграції даних високої продуктивності, включаючи пакети для обробки, вилучення, перетворення та завантаження даних;
- служби аналізу. SQL Server Analysis Services – платформа аналітики та набір інструментів для особистої, командної та корпоративної бізнес-аналітики;

• служби звітності. SQL Server Report Services надає корпоративні функції звітності з підтримкою веб-додатків;

• реплікація. Це набір технологій для копіювання та розповсюдження даних та об’єктів бази даних з однієї бази в іншу, а потім синхронізація між ними;

• послуги з якості даних. Служби якості даних надають рішення для очищення даних на основі знань. Служби дозволяють створити базу знань для того щоб потім виправити дані та видалити дублікати за допомогою автоматизованих та інтерактивних інструментів.

Entity Framework Core представляє крос-платформне ORM-рішення, що дозволяє автоматизувати зв’язок класів у програмі, написаною мовою C#, з таблицями у базі даних. Entity Framework Core має підтримку більшості сучасних популярних СКБД, наприклад: MySQL, MSSQL, Postres тощо.

Entity Framework Core підтримує обидва підходи розробки «Code First» та «Database First». У випадку застосування першого підходу все значно легше і автоматизовано. У випадку коли база даних розроблюється окремо, то необхідно прописати залежності між полями класу і відповідним колонкам таблиці.

2.2 Засоби розробки клієнтського застосунку

2.2.1 Технологія для створення веб-застосунків Blazor

Під час вибору технології для розробки клієнтського застосунку було визначено такі вимоги: фреймворк повинен бути простим у використанні, використовувати статичну строгу типізацію та мати високу продуктивність. Було порівняно найпопулярніші рішення, їх наведено на рисунку 2.5[19]. Відповідно до наших критеріїв, було виділено Angular та Blazor. В кінцевому результаті, було обрано технологію Blazor, оскільки вона використовує мову програмування C#, що спростить процес розробки та дозволить використовувати певні компоненти системи, як у серверному застосунку, так і у клієнтському.

Feature	React	Angular	Vue	Blazor
Type	Front-end Library	Framework	Library	ReleasedFramework
Release Date	2013	2010	2014	2015
Developer	Facebook	Google	Evan You	Microsoft
License	MIT	MIT	MIT	Apache
Language	JavaScript	TypeScript	JavaScript	WebAssembly C#
DOM	Virtual DOM	Regular DOM	Virtual DOM	Incremental DOM
Testing & Debugging	Uses Jest	Uses Jasmine	Vue DevTools	PlaceHolder
Data Binding	One-way Uni-directional	Two-way (Bi-directional)	Two-way (Bi-directional)	One-way (Uni-directional)
Learning Curve	Easy to learn	Steep	Easy to learn	Smooth & Easy
Popularity, Growth & Community Support	Supported by Facebook and Uber developers community	Backed by a large number of developer & Google, Wix developers	Supported by Open-source project sponsored through crowd-sourcing	Microsoft as well as a robust upcoming community of contributors

Рисунок 2.5 – Порівняння найпопулярніших рішень для розробки клієнтських застосунків

Технологія Blazor – це фреймворк для створення інтерфейсу користувача розроблений компанією Microsoft для платформи .NET. Він використовується для створення інтерактивних додатків що можуть працювати як на стороні сервера, так і на стороні клієнта. Blazor має відкритий вихідний код розміщений на ресурсі GitHub[12].

До переваг технології Blazor можемо віднести:

• використання мови C#, на відміну схожих рішень що використовують JavaScript [11]. Це може використовуватися для того щоб серверна та клієнтська частина мали спільний код. Але у випадку необхідності взаємодії із DOM браузера можна використати JavaScript код;

• існує багато вбудованих шаблонів для спрощення створення застосунку;

• можливість використання елементів та алгоритмів платформи .NET.

Також важливо розрізняти дві технології: Blazor Server та Blazor WebAssembly.

Blazor Server надає можливість створювати серверні додатки і підтримується ASP.NET Core. Дана технологія була представлена у 2019 році і фактично була

першим представником даного фреймворку.

На відміну від попередньої технології, Blazor WebAssembly не містить бізнес логіки і інформація отримується від сторонніх сервісів (наприклад API). Blazor WebAssembly створює SPA застосунки, які працюють використовючи технологію WebAssembly. Є більш новою технологією і побачила світ у 2020 році.

Blazor WebAssembly завжди запускається на стороні користувача у браузері. Коли програма завантажується усі її залежності завантажуються браузером.

Проєкт Blazor WebAssembly має структуру зображену на рисунку 2.6 [12].

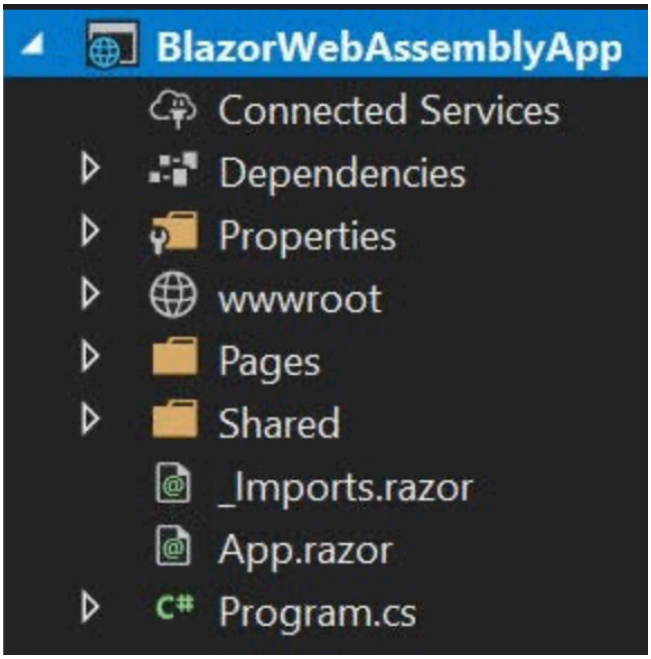

Рисунок 2.6 – Структура проєкту Blazor WebAssembly

Папка wwwroot є кореневою після запуску застосунку і містить статичні медіа файли та файли програм.

У теці Pages знаходяться сторінки та компоненти застосунку які відповідальні за створення інтерфейсу.

Наступною папкою є Shared, яка містить компоненти макетів та компоненти що майже не змінюють вигляд впродовж життєвого циклу застосунку, наприклад меню навігації, топбар, футер.

Файл _Imports.razor містить додаткові імпорти просторів імен, які необхідні

для функціонування компонентів.

Файл App.razor є кореневим компонентом усього застосунку.

Файл Program.cs є точкою входу в програму, є звичайним для ASP.NET Core класом що запускає та конфігурує застосунок.

2.2.2 Бібліотека компонентів Radzen Blazor

Radzen Blazor Components – це безкоштовна колекція з відкритим вихідним кодом попередньо створених компонентів для створення інтерфейсу користувача, які дозволяють розробникам швидко створювати масштабовані інтерактивні веб-застосунки. Компоненти охоплюють багато сценаріїв використання, що полегшує додавання елементів до застосунку[13].

Основні переваги даної бібліотеки:

- широкий набір компонентів. Бібліотека надає форми для введення даних, компоненти для виведення та роботи із даними, компоненти для сповіщення, графіки та діаграми та багато іншого;
- легка інтеграція. Radzen компоненти легко інтегруються в Blazor Server та Blazor WebAssembly застосунки. Для інтеграції необхідно лише встановити відповідний пакет із менеджера пакетів NuGet;
- високий рівень кастомізації компонентів. Компоненти мають багато подій та параметрів для встановлення бажаної поведінки. Також компоненти можна стилізувати за допомогою CSS;
- інформативна документація та високий рівень підтрики;
- підтримка інтернаціоналізації для створення багатомовних додатків.

2.3 Інтеграція сервісів

2.3.1 Сервіс зберігання файлів Azure Blob Storage

Azure Blob Storage – це сервіс, що дає можливість зберігати неструктуровані дані в хмарному сховищі, надаючи доступ до них з будь-якого ресурсу. Структура

сховища наведена на рисунку 2.7 [14].

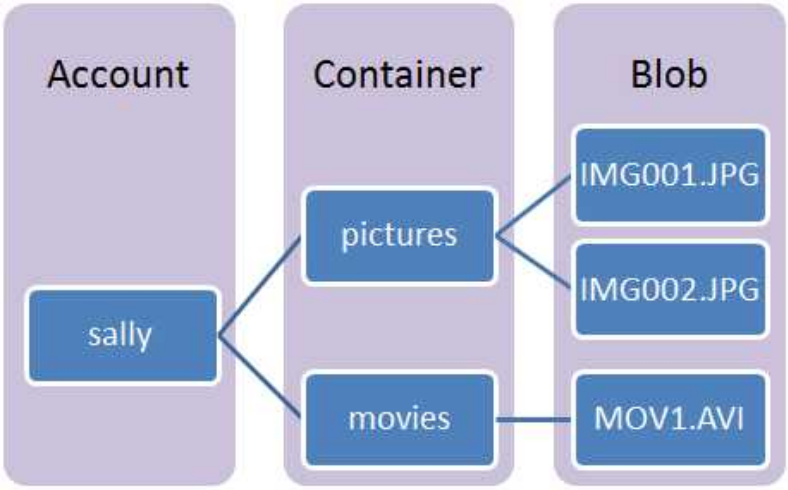

Рисунок 2.7 – Структура хмарного сховища Azure Blob Storage

Структура складається з таких трьох компонентів:

• Storage Account. Доступ до сховища здійснюється за допомогою Azure Storage Account, який створюється у веб-системі Azure Portal;

• Container. Виконує роль певної папки яка групує декілька файлів. Важливо зазначити, що файли можуть зберігатися тільки у контейнерах, кількість контейнерів необмежена;

• Blob – файл будь-якого розміру та формату. Сервіс надає три типи блобів: Block blob, Append blob, Page blob.

Block blob є найкращим вибором для зберігання двійкових та текстових файлів.

Append blob дещо схожий на попередній тип, але більш адаптований для додавання новий компонентів. Такий тип є чудовим рішенням для логування.

Page blob має більший об’єм пам’яті (до 1 ТБ) і найкраще підходить для частих операцій читання та запису. Даний тип використовується віртуальними машинами Azure для зберігання операційних систем на дисках.

Також у цьому сервісі існують певні правила імен у сховищі. Перше правило, це формування посилання для доступу до файлу, воно завжди формується за таким

шаблоном:

http://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>

Наступне правило стосується назви контейнера, по-перше воно повинно бути записане у нижньому регістрі, по-друге назва повинна починатися з букви або цифри і складатися лише з букв цифр і тире, по-третє після тире повинна бути або літера або цифра, і останнє, довжина назви повинна бути від 3 до 63 символів.

Останнє правило стосується назви блобу, вона повинна відповідати таким критеріям:

- мінімальна довжина 1 символ, максимальна 1024;
- назви є регістрозалежними;
- назва може складатися з будь-якої комбінації символів.

2.3.2 Google карти API

Google Maps API – це набір сервісів, що дозволяють інтегрувати функціонал карт у веб-застосунки. API надає широкий спектр можливостей для роботи із картами, локаціями та маршрутами.

Основні компоненти Google maps платформи включають [15]:

- Maps API. Основний компонент який дозволяє відображати карти різних типів, додавати маркери та інші шари на карту, відображати зображення вулиці;
- Routes API. Компонент орієнтований на прокладання маршрутів;
- Places API. Цей компонент отримувати локацію користувача, конвертувати адресу у координати та навпаки, працювати із часовими поясами;
- Environment API. Надає інформацію про якість повітря, сонячну активність та пилок у повітрі.

Для використання Google Maps API необхідно виконати наступні кроки:

- зареєструватися у веб-системі Google Maps Platform;
- у системі створити новий проєкт із інформацією про ціль використання API;
- отримати ключ доступу до API, він є унікальним ідентифікатором який необхідно додавати до кожного запиту.

2.4 Система контролю версій

Під час розробки було використано систему контролю версій Git. Дана система контролює зміни у файлах і є незамінним інструментом для створення застосунків командою розробників. Завдяки їй ви завжди можете повернутися в необхідний момент розробки, відслідкувати та знайти які зміни призвели до того чи іншого наслідку.

Загальна концепція контролювання версій зображена на рисунку 2.8 і має такий сенс [16]:

1. розробник виконує певні зміни у коді, які становлять закінчений вид;
2. додає нові файли індексу (опціонально, нові файли можуть бути додані автоматично на наступному кроці);
3. виконує операцію коміт із заданням опису виконаної роботи. У випадку необхідності додати файли на цьому етапу до запиту додається «-a»;
4. оновлює віддалений репозиторій оновленою версією (за необхідності).

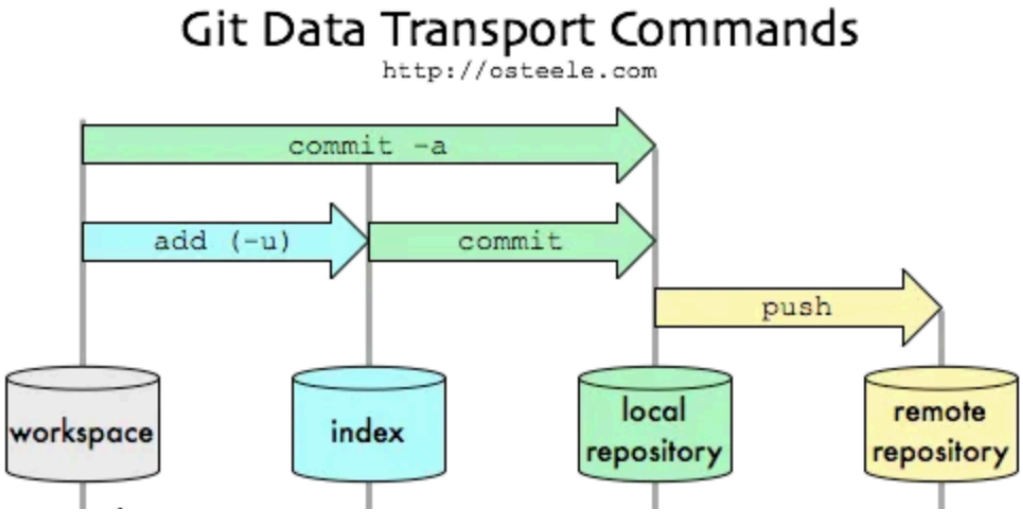

Рисунок 2.8 – Графічне зображення алгоритму додавання змін до системи контролю версій Git

Але сама по собі система контролю версій не зможе допомогти команді розробників, вона працює в парі із сервісами збереження репозиторіїв проєктів. У нашому випадку це GitHub, який є найпопулярнішим рішенням у цій сфері.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Система, перш за все, повинна бути корисна для кінцевого користувача, для цього необхідно створити зрозумілий та гарний інтерфейс та необхідний функціонал. Увесь функціонал повинен бути протестований та налагоджений.

3.1 Опис функціональної моделі системи

Згідно з функціональними вимогам до системи, веб-система повинна мати розподіл функціоналу по ролям користувачів. Для повноцінного функціонування системи та контролем над нею, варто створити такі ролі: абітурієнт, учень, шкільний адміністратор, модератор, адміністратор, супер адміністратор. Перші три є ролями сторонніх користувачів, останні три – це системні ролі, які забезпечують функціонування застосунку згідно правил та політик. На рисунку 3.1 наведено діаграму прецедентів для веб-системи пошуку та рейтингування навчальних закладів.

Користувачу із роллю «Абітурієнт» доступний функціонал перегляду інформації, а саме пошук шкіл відповідно до заданих фільтрів, перегляд інформації про вибраний заклад, зокрема відгуки учнів, загальний рейтинг та оцінки по критеріям.

Наступною роллю є «Учень». Користувачі із такою роллю мають можливості абітурієнта і додатково можуть залишати коментарі та оцінки до навчальних закладів, редагувати та видаляти залишені коментарі та оцінки.

Для того, щоб представник школи міг додавати заклади, швидко оновлювати інформацію, аналізувати залишені коментарі та підтримувати зв’язок з учнями, що залишили коментарі, було додано роль «Шкільний адміністратор». Додавання до системи користувача із такою роллю відбувається після перегляду заявки адміністратором для дотримання інформаційної безпеки веб-системи.

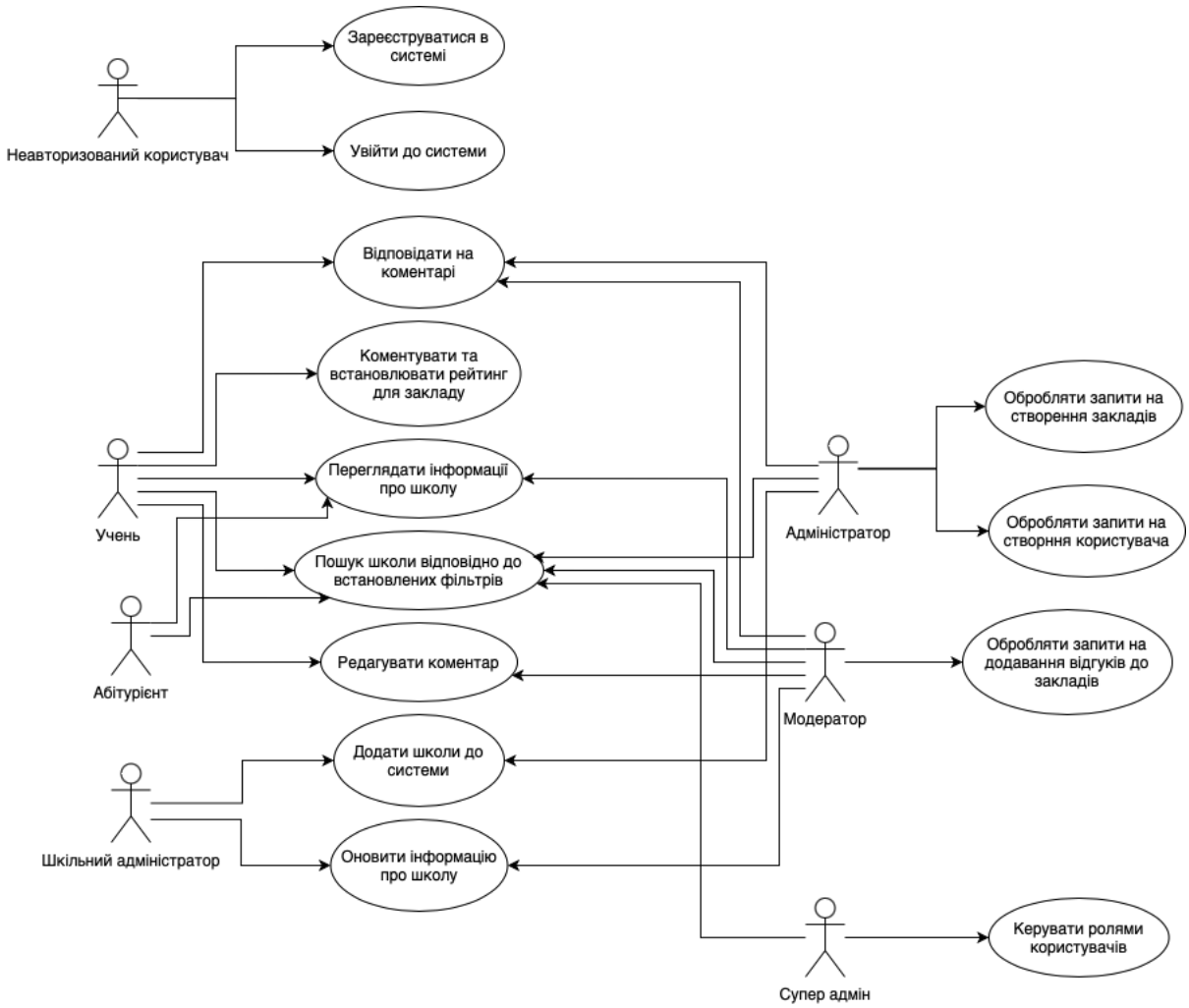

Рисунок 3.1 – Діаграма прецедентів веб-системи пошуку та рейтингування навчальних закладів

Наступною роллю є «Модератор», такий користувач відповідальний за дотримання політик системи. Він має доступ до редагування та видалення будь-якої інформації у системі, зокрема інформації про школи, залишені коментарі та оцінки. Ще модератор повинен розглядати запити на додавання коментарів, це пов’язано із дотриманням правил інформаційної безпеки.

Для ролі «Адміністратор» є додаткові функціональні можливості для обробки запитів на створення нових користувачів та додавання нових закладів до системи.

3.2 Опис серверного застосунку

3.2.1 Опис архітектури серверного застосунку

Серверний застосунок веб-системи було розроблено із використанням тришарової архітектури. Зображення архітектури серверного застосунку наведено на рисунку 3.2.

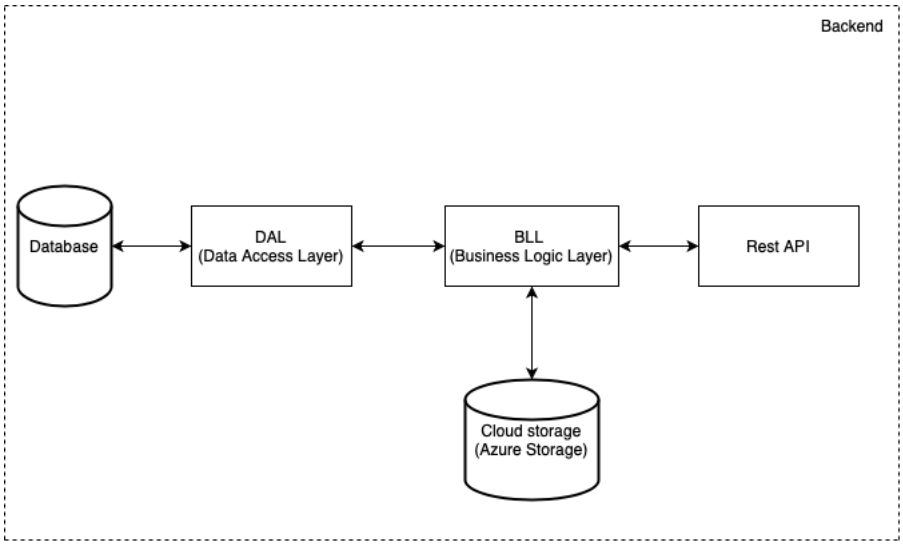

Рисунок 3.2 – Архітектура серверного застосунку

Функціонування застосунку відбувається у такій послідовності: REST Api отримує запит від клієнтського застосунку і викликає необхідні методи шару бізнес логіки для його обробки. У свою чергу шар бізнес логіки виконує необхідну до запиту роботу, майже завжди вона пов’язана із роботою з рівнем DAL.

Основної складовою роботи REST Api є отримання запитів від клієнтського застосунку. Для цього було створено контролери відповідно до потреб системи, а саме: AuthController, RegistrationController, SchoolController, CommentController, RatingController, ReplyController, SchoolRegistrationController, CommentCreationController та UserController.

AuthController відповідає за авторизацію користувача в системі та має один ендпоінт. Його опис наведено у таблиці 1.

Таблиця 1 – Ендпоінти AuthController

Шлях	Структура запиту	Структура відповіді	Відповідь сервера	Опис

/api/Auth/login	{
	"email": "string",
	"password": "string"
	}
		{ "token": "string",
		"expireOn": "datetime",
		"user": { "id": "string",
		"firstName": "string",
		"lastName": "string",
		"roles": ["string"] },
		"isEmpty": false }
			200	Застосовується для авторизації користувача в системі

Наступний контролер відповідає за додавання нових користувачів системи, для цього створено ендпоінти, що дозволяють опрацьовувати і створення запитів на створення акаунту і створення одразу нових користувачів. Опис цих ендпоінтів наведено у таблиці 2.

Таблиця 2 – Ендпоінти RegistrationController

Шлях	Структура запиту	Структура відповіді	Відповідь сервера	Опис
/api/Registration/create	RegistrationForm	bool	200 400 401	Додає запит на створенння акаунту

/api/Registration/find	{
	"pageIndex": 0,
	"pageSize": 0,
	"searchText": "string",
	"state": 0,
	"sortBy": 1,
	"orderBy": 1
	}
		{
		"totalCount": 0,
		"values": [],
		"count": 0
		}
			200
			400
			401
			403
				Знаходить запити на створення акаунтів відповідно фільтру

/api/Registration/register	{
	"firstName": "string",
	"lastName": "string",
	"email": "string",
	"password": "string",
	"role": "string"
	}
		bool	200
			400
			401
				Додає акаунт до системи

/api/Registration/update	RegistrationForm	bool	200 401 403	Оновлює запит на реєстрацію

Для роботи із різноманітною інформацією про школи у системі, було створено SchoolController, CommentController, RatingController, ReplyController відповідальні за роботи із закладами, відгуками, оцінками та відповідями до коментарів відповідно. вони надають можливість додавати нову інформацію, вносити зміни до існуючої, видаляти, шукати відповідно до встановлених фільтрів тощо. Детальний опис ендпоінтів цих контролерів наведено у таблиці 3.

Таблиця 3 – Ендпоінти SchoolController, CommentController, RatingController, ReplyController

Шлях	Структура запиту	Структура відповіді	Відповідь сервера	Опис
/api/School/create	SchoolDto	bool	200 401	Додає нову школу
/api/School/delete/{schoolId}		bool	200 401	Видаляє школи із системи
/api/School/find	SchoolFilter	PagedList<SchoolDto>	200 401	Знаходить школи відповідно до заданого фільтру
/api/School/update	SchoolDto	bool	200 401 403	Оновлює інформацію про школу
/api/Comment/create	Приймає обєкт класу CommentDto	bool	200 400 401	Додає відгук до системи
/api/Comment/delete/ {commentId}		bool	200 400 401	Видаляє відгук та оцінки з системи
/api/Comment/find	CommentFilter	PagedList <CommentDto>	200 400 401	Знаходить відгуки відповідно до заданого фільтру
/api/Comment/update	Приймає обєкт класу CommentDto	bool	200 400 401	Оновлює інформацію про відгук

/api/Rating/create	[
	{
	"id": "guid",
	"category": 1,
	"value": 0
	}
	]
		bool	200
			400
			401
				Додає оцінку для закладу

/api/Rating/delete/{ratingId}		bool	200 400 401	Видаляє оцінку для закладу

/api/Rating/find	RatingFilter	{
		"totalCount": 0,
		"values": [],
		"count": 0
		}
			200
			400
			401
				Знаходить оцінки відповідно до фільтру

/api/Rating/update	[
	{
	"id": "guid",
	"category": 1,
	"value": 0
	}
	]
		bool	200
			400
			401
				Оновлює оцінку закладу

/api/Reply/add	ReplyDto	bool	200 401	Додає відповідь до коментаря
/api/Reply/delete/{replyId}		bool	200 401	Видаляє відповідь до коментаря
/api/Reply/find	ReplyFilter	PagedList<ReplyDto>	200 401	Отримує відповіді до коментарів
/api/Reply/update	ReplyDto	bool	200 401	Оновлює відповідь

Для роботи із запитами на створення шкіл та додавання коментарів створено контролери SchoolRegistrationController та CommentCreationController. У таблиці 4 наведено ендпоінти для роботи з ними.

Таблиця 4 – Ендпоінти SchoolRegistrationController та CommentCreationController

Шлях	Структура запиту	Структура відповіді	Відповідь сервера	Опис
/api/SchoolRegistration/create	SchoolCreationRequestDto	bool	200 401	Додає запит на створення школи

/api/SchoolRegistration/find	{
	"pageIndex": 0,
	"pageSize": 0,
	"minRange": 0,
	"maxRange": 0,
	"minTotalRating": 0,
	"maxTotalRating": 0,
	"ratingCategoryFilters": [
	{
	"category": 1,
	"minValue": 0,
	"maxValue": 0
	}
	],
	"ownerId": "guid",
	"searchText": "string",
	"sortBy": 1,
	"orderBy": 1,
	"userLocation": {
	"id": "guid",
	"latitude": 0,
	"longitude": 0
	},
	"state": 0
	}
		{
		"totalCount":
		0,
		"values": [],
		"count": 0
		}
			200
			401
			403
				Знаходить запити на створення школи

/api/SchoolRegistration/update	SchoolCreationRequestDto	bool	200 401	Оновлює запит на створення школи
/api/CommentCreation/create	Приймає обєкт класу CommentCreationRequestDto	bool	200 400 401	Додає запит на створення відгуку

/api/CommentCreation/find	{
	"pageIndex": 0,
	"pageSize": 0
	}
		{
		"totalCount":0,
		"values": [],
		"count": 0
		}
			200
			400
			401
			403
				Знаходить запити на додавання коментарів

/api/CommentCreation/update	Приймає обєкт класу CommentCreationRequestDto	bool	200 400 401 403	Оновлює запит

Контролер UserController необхідний для роботи із користувачами системи. Він має два ендпоінта для пошуку користувачів та оновлення інформації про них. У таблиці 5 описано це ендпоінти.

Таблиця 5 – Ендпоінти UserController

Шлях	Структура запиту	Структура відповіді	Відповідь сервера	Опис

/api/User/find	{
	"pageIndex": 0,
	"pageSize": 0,
	"searchText": "string",
	"roles": [
	"string"
	]
	}
		{
		"totalCount": 0,
		"values": [],
		"count": 0
		}
			200
			401
				Знаходить користувачів системи

/api/User/update	{
	"id": "string",
	"firstName": "string",
	"lastName": "string",
	"roles": [
	"string"
	]
	}
		bool	200
			401
			403
				Оновлює акаунт користувача

Для реалізації необхідного функціоналу серверного застосунку було створено шар бізнес логіки, який виконує операції із базою даних та сервісом хмарного сховище Azure Blob Storage.

За виконання основного функціоналу серверного застосунку відповідають такі сервіси:

- AuthService, RegistrationService, RegistrationFormService, UserService, які відповідають за реєстрацію, авторизацію та роботою із інформацією користувача;
- SchoolService, CommentService, RatingService, ReplyService, що забезпечують роботу із інформацією про заклади та залишені до них відгуки у системі;
- SchoolRegistrationService, CommentCreationService, які необхідні для роботи із запитами на створення шкіл та відгуків до них;
- BlobStorageService для роботи із сервісом Azure Blob Storage.

Більш детально взаємодію компонентів серверного застосунку наведено на діаграмі класів на рисунку 3.3.

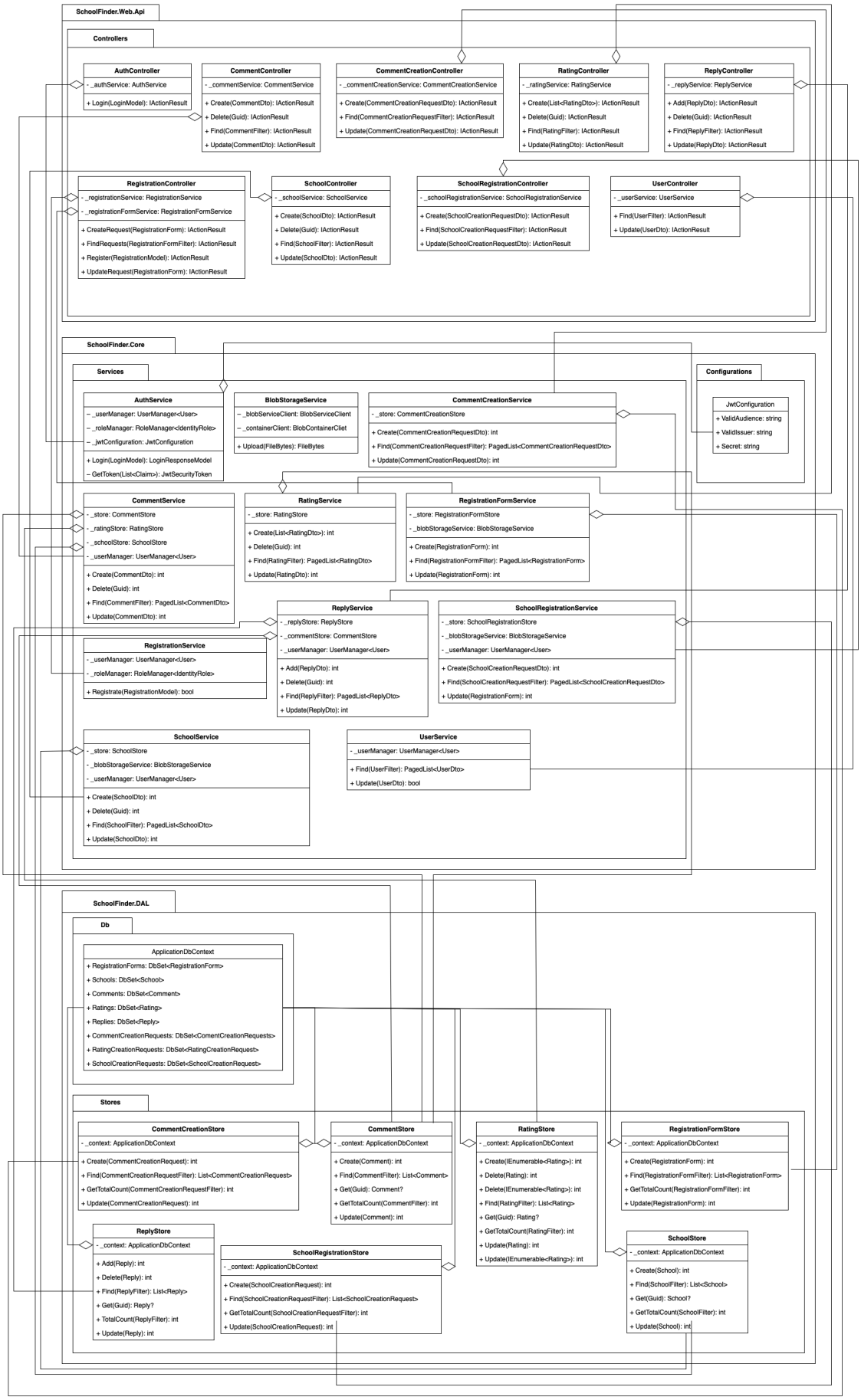

Рисунок 3.3 – Діаграма класів серверного застосунку

3.2.2 Опис моделі даних

Огляд створеної моделі даних можна поділити на дві частини, оскільки для було створено основні класи для роботи бізнес логіки та класи для передачі даних на клієнтський застосунок.

Модель даних для роботи серверного застосунку наведено на рисунку 3.4.

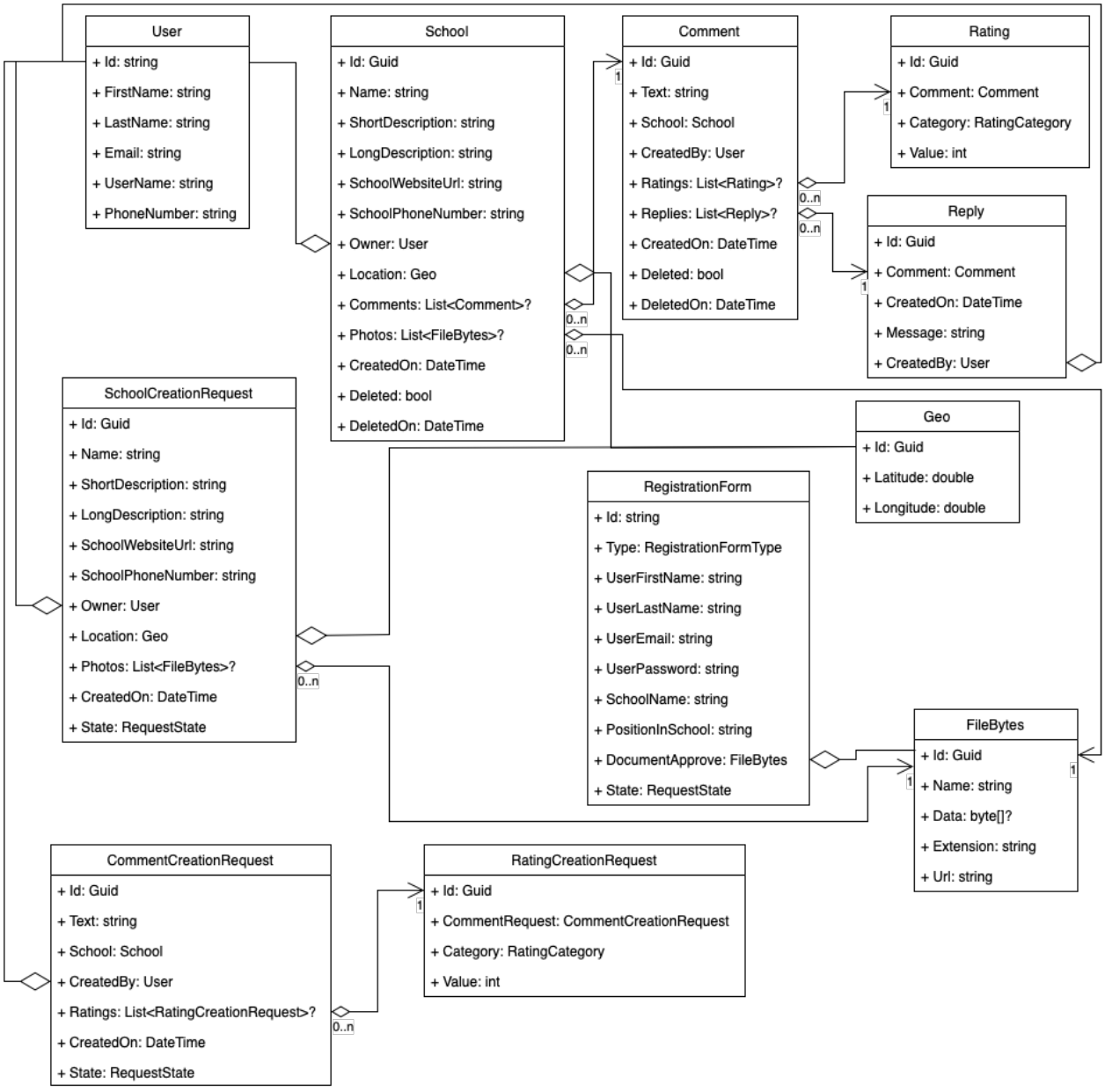

Рисунок 3.4 – Модель даних серверного застосунку

Створено такі класи:

- User – описує акаунт користувача;

- School – містить повну інформацію про школу;
- Comment – описує модель відгуку до закладу. Містить дані про школу до якої залишений, оцінки додані разом з ним, відповіді залишені до нього;
- Rating – клас, що описує оцінку, містить інформацію про те до якого відгуку вона відноситься, про категорію залишеної оцінки та її значення;
- Reply – містить інформацію про відповідь до відгуку залишеного до школи;
- Geo – описує місце розташування закладу на планеті;
- FileBytes – описує завантажені файли до системи;
- SchoolCreationRequest – клас, що описує запит на додавання школи до системи;
- CommentCreationRequest – містить опис запиту на додавання коментаря до закладу до системи;
- RatingCreationRequest – описує оцінку, яку містить запит на створення відгуку;
- RegistrationForm – клас, що описує запит на створення акаунту. Необхідний для певних ролей, акаунти яких потребують огляду перед додаванням до системи.

Також було створено модель даних об’єктів DTO, які використовуються у клієнтському застосунку та передаються з серверного застосунку. Модель даних зображена на рисунку 3.5.

Створені класи для опису DTO об’єктів мають менший об’єм інформації, враховуючи його не потрібність для розробки клієнтського застосунку. Таким чином покращується безпека та швидкодія застосунку. Прикладом цього є класи User та UserDto. Перший має доволі великий обсяг полів необхідних для роботи бізнес-логіки, проте не потрібної для клієнтського. Натомість для клієнтського застосунку було б зручно мати ролі у класі користувача.

Створеними класами DTO є: UserDto, SchoolDto, CommentDto, RatingDto, ReplyDto, SchoolCreationRequestDto, CommentCreationRequestDto, RatingCreationRequestDto. Створення DTO для інших класів не має сенсу, оскільки це не вплине на роботу системи, але додасть копіювання коду, що є поганою практикою.

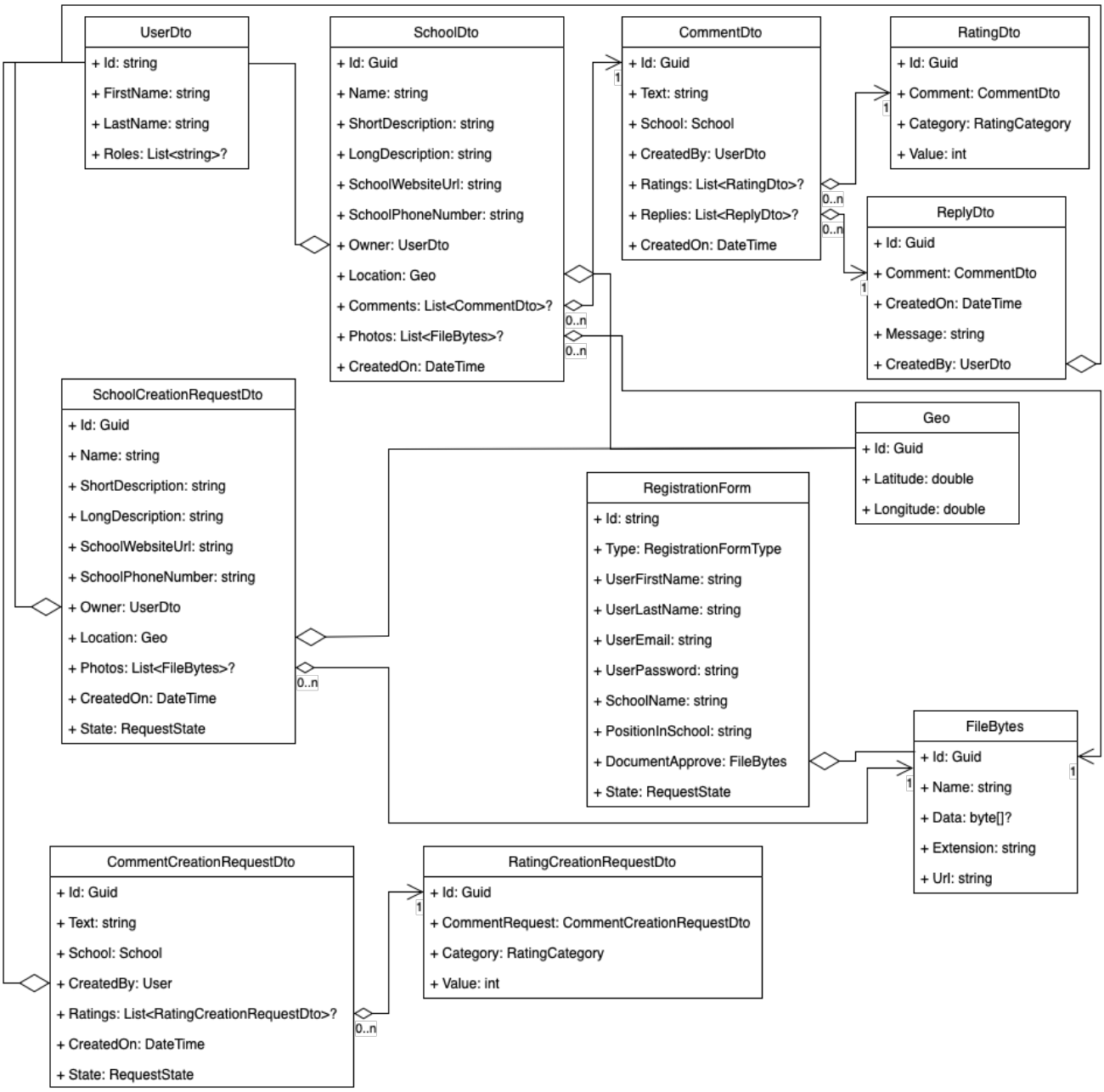

Рисунок 3.5 – Модель даних DTO об’єктів

3.2.3 Опис бази даних

Створення бази даних відбувалося із підходом “CodeFirst”, тобто після створення об’єктів у системі було висунуто чітку вимоги до структури бази даних. На рисунку 3.6 наведено логічну діаграму для створення бази даних.

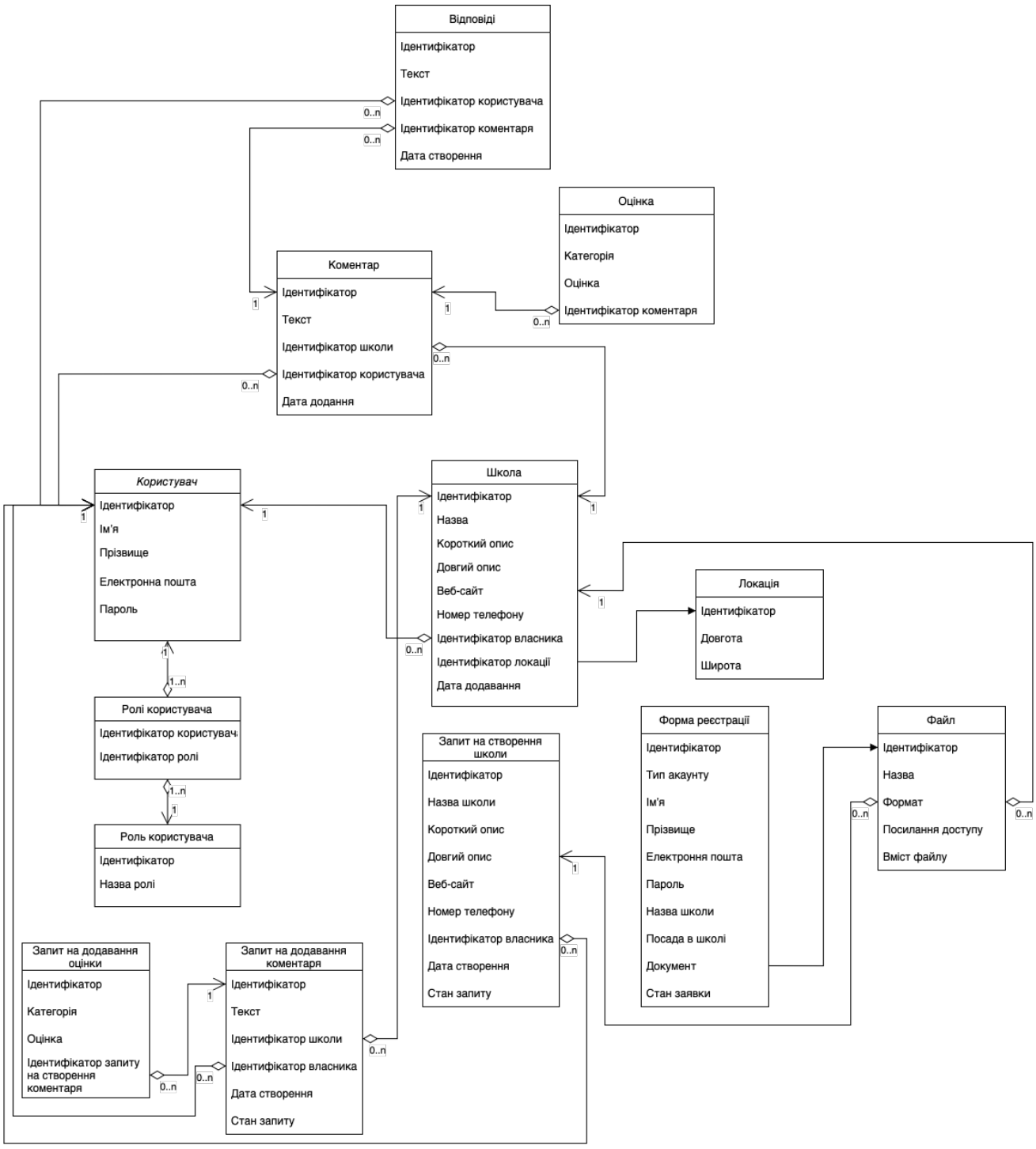

Рисунок 3.6 – Логічна модель даних

Враховуючи логічну модель даних, було створено базу даних під керуванням СКБД Microsoft SQL Server. Для спрощення процесу створення бази даних, було використано технологію Entity Framework Core. Вона дозволила автоматизувати процес створення, оскільки скрипт було автоматично сгенеровано враховуючи модель даних. На рисунку 3.7 зображено структуру готової бази даних.

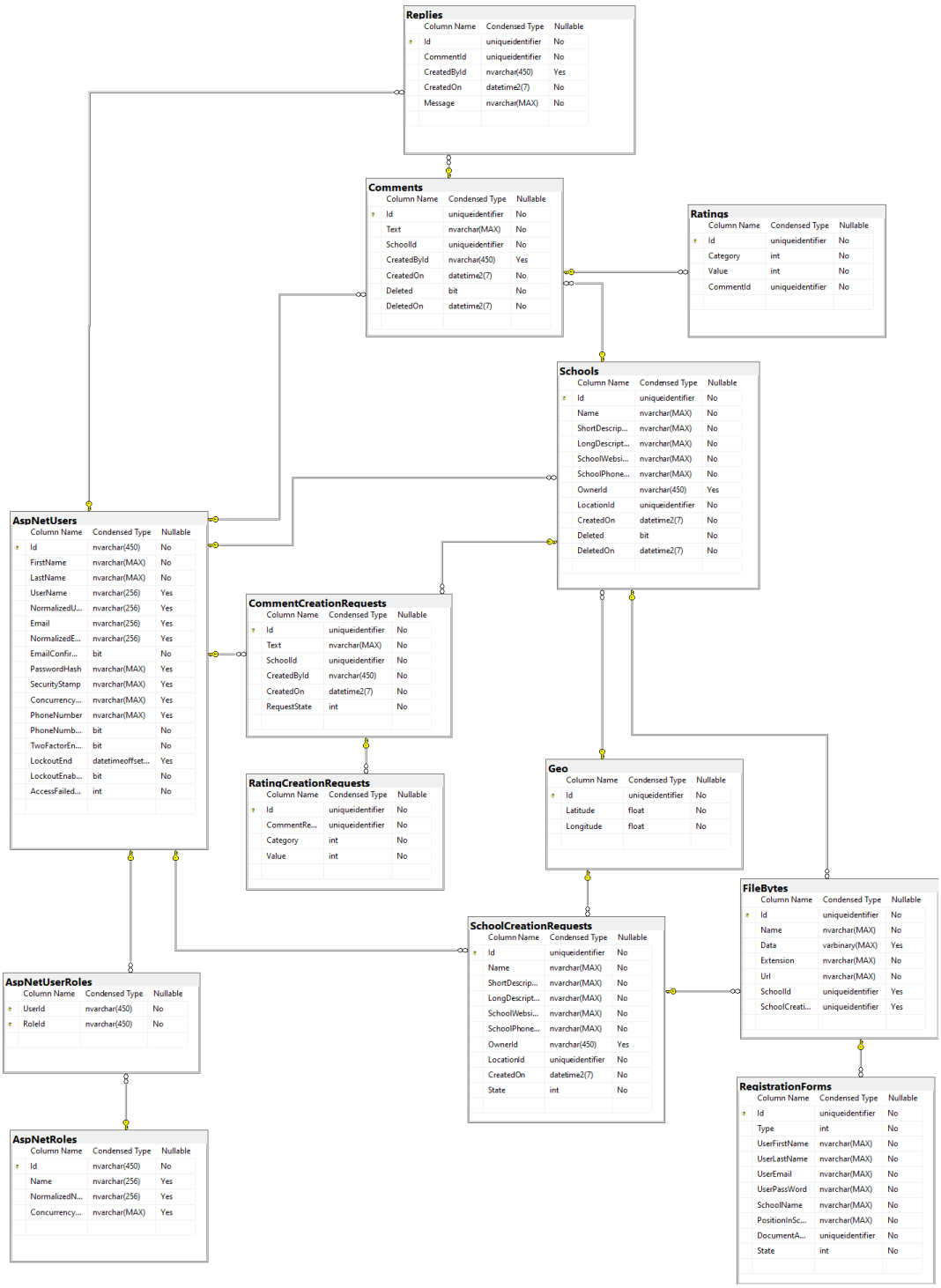

Рисунок 3.7 – Діаграма створеної бази даних

Таблиці створеної бази даних можна умовно поділити на три групи: пов’язані зі зберіганням користувачів та ролей, ті що зберігають різноманітні запити та ті, що містять інформацію про школу.

До першої групи відносяться таблиці «AspNetUsers», «AspNetRoles», «AspNetUserRoles». Перша містить інформацію про акаунт користувача, детальна

інформація про таблицю наведена у таблиці 6.

Таблиця 6 – Опис таблиці «AspNetUsers»

Назва поля	Тип	Обмеження	Опис
Id	nvarchar(450)	До 450 символів, не порожне	Унікальний ідентифікатор користувача
FirstName	nvarchar(MAX)	Не порожне	Ім’я користувача
LastName	nvarchar(MAX)	Не порожне	Прізвище користувача
Email	nvarchar(256)	До 256 символів	Електронна пошта користувача
PasswordHash	nvarchar(MAX)		Пароль користувача
UserName	nvarchar(256)	До 256 символів, не порожне	Ідентифікатор користувача

Таблиця «AspNetRoles» зберігає існуючі у системі ролі, що дозволяє реалізувати розподіл функціоналу між ролями у системі. Структура цієї таблиці описана у таблиці 7.

Таблиця 7 – Опис таблиці «AspNetRoles»

Назва поля	Тип	Обмеження	Опис
Id	nvarchar(450)	До 450 символів, не порожне	Унікальний ідентифікатор ролі
Name	nvarchar(256)	До 256 символів	Назва ролі
NormalizedName	nvarchar(256)	До 256 символів	Нормалізована назва ролі

Останньою таблицею цієї групи є «AspNetUserRoles». Вона надає можливість додавати користувачу більше за одну роль. Опис цієї таблиці наведено у таблиці 8.

Таблиця 8 – Опис таблиці «AspNetUserRoles»

Назва поля	Тип	Обмеження	Опис
UserId	nvarchar(450)	До 450 символів, не порожне	Ідентифікатор користувача
RoleId	nvarchar(450)	До 450 символів, не порожне	Ідентифікатор ролі

До групи, що зберігають запити відносяться таблиці: «RegistrationForms», «SchoolCreationRequests», «CommentCreationRequests», «RatingCreationRequests».

Таблиця «RegistrationForms» містить запити на реєстрацію користувачів, що повинні бути розглянуті адміністратором. Структура таблиці описана у таблиці 9.

Таблиця 9 – Опис таблиці «RegistrationForms»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор запиту
Type	int	Не порожне	Тип акаунту, що треба буде створити
UserFirstName	nvarchar(MAX)	Не порожне	Ім’я
UserLastName	nvarchar(MAX)	Не порожне	Прізвище
UserEmail	nvarchar(MAX)	Не порожне	Електронна пошта
UserPassword	nvarchar(MAX)	Не порожне	Пароль
SchoolName	nvarchar(MAX)	Не порожне	Назва школи
PositionInSchool	nvarchar(MAX)	Не порожне	Посада у школі
DocumentApprove	uniqueidentifier	Не порожне	Документ наданий для підтвердження особистості
State	int	Не порожне	Стан запиту

Таблиця «SchoolCreationRequests» містить запити на додавання шкіл до системи і має структуру наведену у таблиці 10.

Таблиця 10 – Опис таблиці «SchoolCreationRequests»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор запиту
Name	nvarchar(MAX)	Не порожне	Назва закладу
ShortDescription	nvarchar(MAX)	Не порожне	Короткий опис закладу
LongDescription	nvarchar(MAX)	Не порожне	Розширений опис школи
SchoolWebSite	nvarchar(MAX)	Не порожне	Веб-сайт
SchoolPhoneNumber	nvarchar(MAX)	Не порожне	Номер телефону

Продовження таблиці 10

OwnerId	nvarchar(450)	До 450 символів	Ідентифікатор користувача, хто створив запит
LocationId	uniqueidentifier	Не порожне	Ідентифікатор місця розташування
CreatedOn	datetime2(7)	Не порожне	Дата створення
State	int	Не порожне	Стан запиту

Таблиця «CommentCreationRequests» містить запити на додавання відгуків до шкіл до системи і має структуру наведену у таблиці 11.

Таблиця 11 – Опис таблиці «CommentCreationRequests»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор запиту
Text	nvarchar(MAX)	Не порожне	Текст коментаря
SchoolId	uniqueidentifier	Не порожне	Ідентифікатор школи до якої залишено відгук
CreatedById	nvarchar(450)	До 450 символів, не порожне	Ідентифікатор користувача, хто створив запит
CreatedOn	datetime2(7)	Не порожне	Дата створення
RequestState	Int	Не порожне	Стан запиту

Останньою таблицею другої групи є «RatingCreationRequests». Вона містить оцінки, які залишені у відгуку, що поки проходить модерацію. У таблиці 12 наведено її структуру.

Таблиця 12 – Опис таблиці «RatingCreationRequests»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор запиту
CommentRequestId	uniqueidentifier	Не порожне	Ідентифікатор відгуку до якого відноситься оцінка

Продовження таблиці 12

Category	int	Не порожне	Категорія до якої відноситься оцінка
Value	int	Не порожне	Оцінка

Остання, третя група містить такі таблиці: «Schools», «Comments», «Ratings», «Replies», «Geo», «FileBytes». Перша таблиця зберігає інформацію про школи у системі та її опис наведено у таблиці 13.

Таблиця 13 – Опис таблиці «Schools»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор школи
Name	nvarchar(MAX)	Не порожне	Назва закладу
ShortDescription	nvarchar(MAX)	Не порожне	Короткий опис закладу
LongDescription	nvarchar(MAX)	Не порожне	Розширений опис школи
SchoolWebSite	nvarchar(MAX)	Не порожне	Веб-сайт
SchoolPhoneNumber	nvarchar(MAX)	Не порожне	Номер телефону
OwnerId	nvarchar(450)	До 450 символів	Ідентифікатор власника
LocationId	uniqueidentifier	Не порожне	Ідентифікатор місця розташування
CreatedOn	datetime2(7)	Не порожне	Дата створення
Deleted	bit	Не порожне	Показник чи видалено заклад
DeletedOn	datetime2(7)	Не порожне	Дата видалення школи із системи

Таблиця «Comments» зберігає відгуки залишені до шкіл. Структуру таблиці описано у таблиці 14.

Таблиця 14 – Опис таблиці «Comments»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор
Text	nvarchar(MAX)	Не порожне	Текст коментаря

Продовження таблиці 14

SchoolId	uniqueidentifier	Не порожне	Ідентифікатор школи до якої залишено відгук
CreatedById	nvarchar(450)	До 450 символів	Ідентифікатор користувача, хто створив коментар
CreatedOn	datetime2(7)	Не порожне	Дата створення
Deleted	bit	Не порожне	Показник чи видалено коментар
DeletedOn	datetime2(7)	Не порожне	Дата видалення коментаря

Таблиця «Ratings» містить оцінки шкіл, структуру наведену у таблиці 15.

Таблиця 15 – Опис таблиці «Ratings»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор
CommentId	uniqueidentifier	Не порожне	Ідентифікатор відгуку до якого відноситься оцінка
Category	int	Не порожне	Категорія до якої відноситься оцінка
Value	int	Не порожне	Оцінка

Наступною є таблиця «Replies», вона має на меті зберігання відповідей залишених до коментарів. Опис таблиці наведено у таблиці 16.

Таблиця 16 – Опис таблиці «Replies»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор
CommentId	uniqueidentifier	Не порожне	Ідентифікатор відгуку
CreatedById	nvarchar(450)	До 450 символів	Ідентифікатор користувача, хто лишив відповідь
CreatedOn	datetime2(7)	Не порожне	Дата створення
Message	nvarchar(MAX)	Не порожне	Текст відповіді

Таблиця «Geo» зберігає локації шкіл. Структуру таблиці описано у таблиці 17.

Таблиця 17 – Опис таблиці «Geo»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор
Latitude	float		Широта позиції
Longitude	float		Довгота позиції

Останньою є таблиця «FileBytes», створена для зберігання інформацію про завантажені до системи файли та доступ до них. Її структура наведена у таблиці 18.

Таблиця 18 – Опис таблиці «FileBytes»

Назва поля	Тип	Обмеження	Опис
Id	uniqueidentifier	Не порожне	Ідентифікатор
Name	nvarchar(MAX)	Не порожне	Назва файлу
Data	varbinary(MAX)		Вміст файлу
Extension	nvarchar(MAX)	Не порожне	Формат файлу
Url	nvarchar(MAX)	Не порожне	Посилання для доступу
SchoolId	uniqueidentifier		Ідентифікатор школи
SchoolCreationRequestId	uniqueidentifier		Ідентифікатор запиту

3.3 Опис клієнтського застосунку

Розробка клієнтського застосунку відбувалась із використанням технології Blazor WebAssembly. Для роботи застосунку необхідно було створити компоненти та сервіси, які будуть виконувати наступні задачі:

- надсилати запити до серверного застосунку;
- переходити між сторінками клієнтського застосунку;
- зберігати стан системи.

На рисунку 3.8 наведено діаграму класів сервісів та компонентів, що

вирішують ці задачі для клієнтського застосунку.

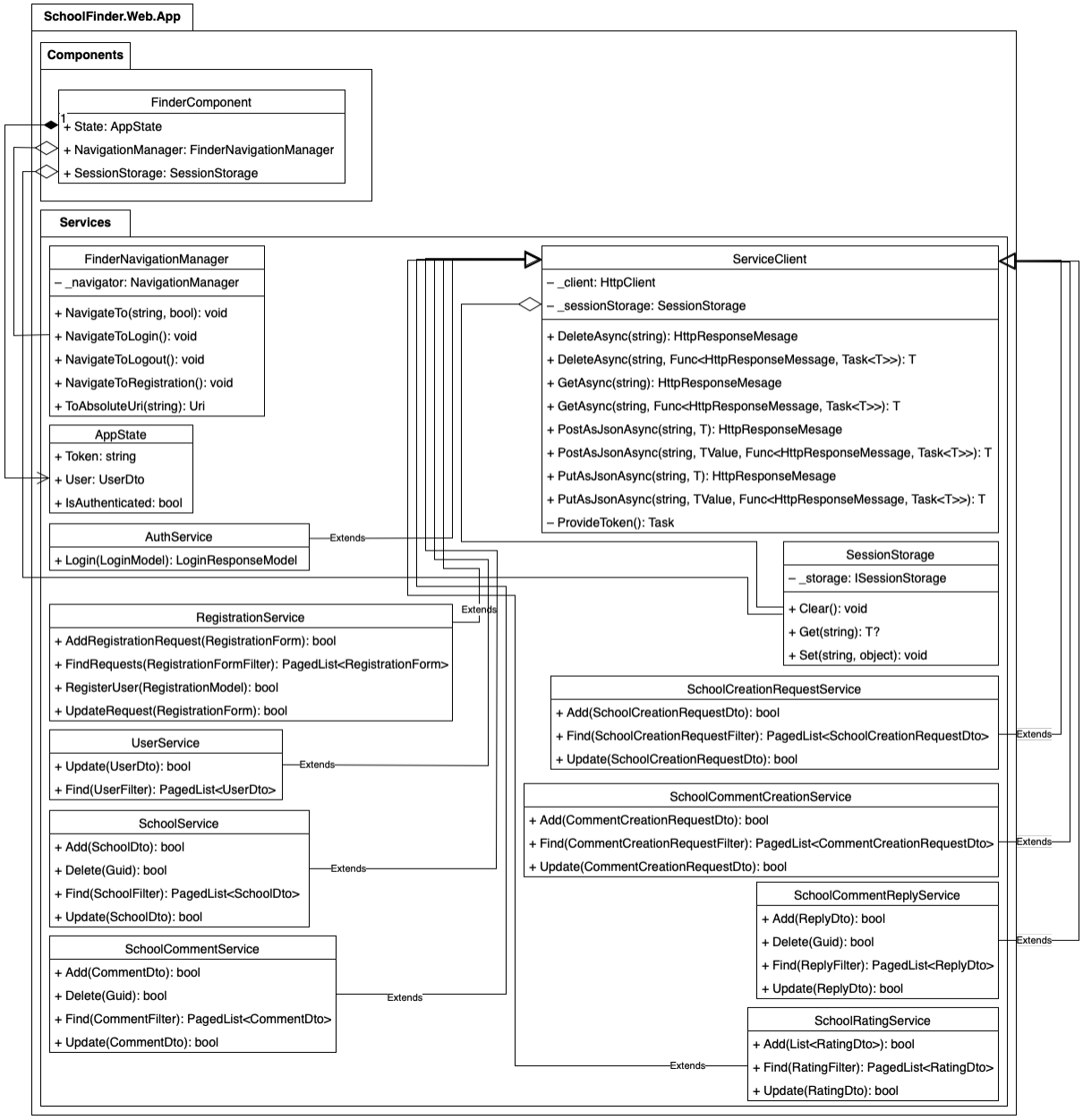

Рисунок 3.8 – Діаграма класів клієнтського застосунку

Виконання першого завдання зумовило створення сервісів для надсилання запитів до описаних в розділі 3.2 ендпоінтів серверного застосунку. Для спрощення цього процесу було створено клас ServiceClient, який надає функціонал надсилання різноманітних запитів і став батьківським для усіх сервісів. Було створено такі сервіси: AuthService, RegistrationService, UserService, SchoolService, SchoolCommentService, SchoolRatingService, SchoolCommentReplyService, SchoolCreationRequestService, SchoolCommentCreationService.

Для швидкої навігації по клієнтському застосунку було створено клас FinderNavigationManager, який за основу має вбудований в Blazor WebAssembly NavigationManager.

Завдання зберігання стану системи зумовило створення компонентів FinderComponent та AppState. FinderComponent є базовим компонентом для усіх інших компонентів та сторінок застосунку, він має об’єкт класу AppState, що зберігає стан системи. Під час ініціалізації компоненту, створюється новий об’єкт класу AppState, який виконує читання стану системи із sessionStorage за допомогою сервісу SessionStorage.

3.4 Опис головного функціоналу

Розроблена веб-система повинна забезпечувати користувачів простим, в той же час доволі широким функціоналом. До основних функцій системи варто віднести пошук шкіл відповідно до заданих користувачем параметрів та оцінювання закладів.

Для рейтингування шкіл було визначено дев’ять головних критеріїв, що визначають загальний стан закладу, а саме:

- складність навчання;
- вимогливість вчителів;
- успіхи учнів;
- індивідуальний підхід;
- інклюзивність будівлі;
- стан ремонту;
- зручність розташування;
- стан інфраструктури довкола;
- місце для паркування.

Рейтинг навчального закладу складається з оцінок залишених користувачами і відображається, як загальна оцінка, так і оцінка по кожній з категорій. Діаграма

послідовності оцінки закладів наведена на рисунку 3.9.

Для додавання відгуку, користувачу необхідно перейти на сторінку шкіл, після чого знайти бажаний заклад та відкрити вікно повної інформації про нього. У відкритому вікні є розділ з коментарями, де користувачу необхідно обрати опцію додавання відгуку. Після цього до інтерфейсу додається форма для встановлення оцінок до категорій та поле текстового коментаря. Заповнивши форму, користувач підтверджує додавання відгуку. Далі до системи додається запит на створення коментаря, це зроблено для уникнення додавання хибної інформації та дотримання політик системи. На наступному етапі модератор переглядає запити на додавання коментаря. У випадку відповідності відгуку правилам та політикам системи, запит підтверджується і відгук додається до системи і відображається на сторінці.

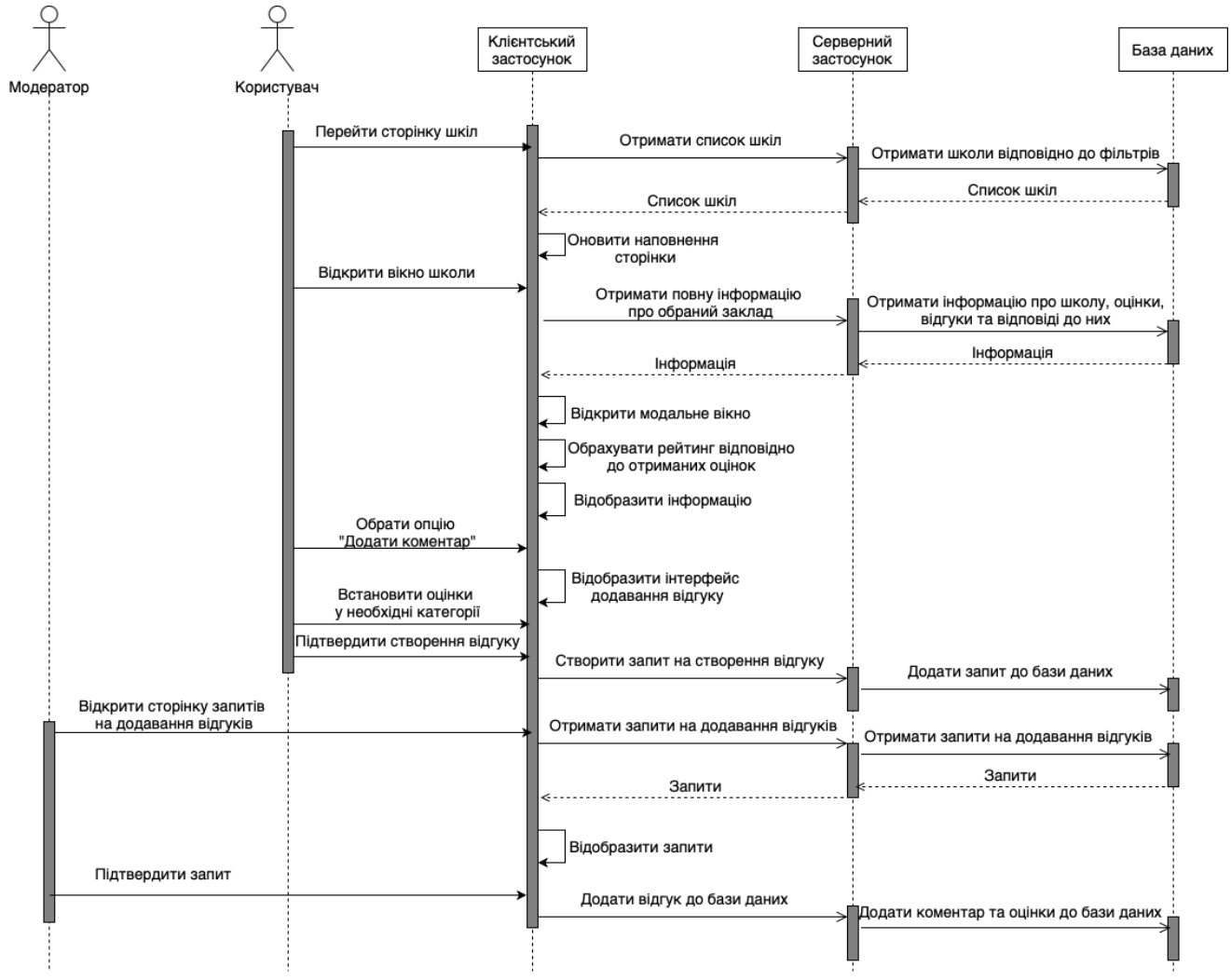

Рисунок 3.9 – Діаграма послідовності рейтингування навчального закладу

Процес пошуку шкіл включає встановлення фільтрів відповідно до власних потреб користувача. Такими потребами є текстовий пошук за ключовими словами, сортування за певною інформацією, встановлення обмежень на відстань до навчального закладу та встановлення обмежень на загальний рейтинг та на необхідні категорії.

На рисунку 3.10 наведено діаграму послідовності для пошуку шкіл. Для пошуку необхідно перейти на сторінку шкіл. На сторінці задати необхідні параметри фільтри пошуку. Можна задати обмеження на всі категорії оцінок, на загальну оцінку, відстань до закладу та текстовий пошук, відповідно до потреб. Після чого виконати пошук. Система оновить список відповідно до заданих фільтрів.

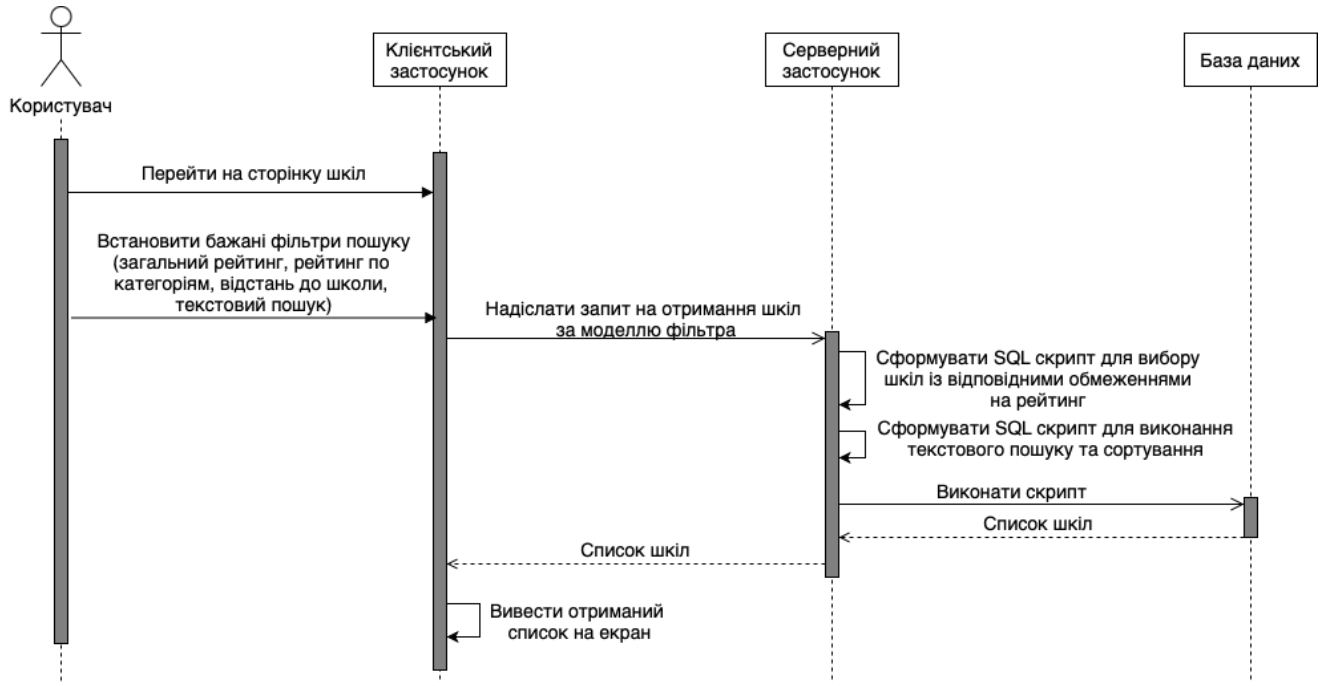

Рисунок 3.10 – Діаграма послідовності пошуку шкіл

Отриманий функціонал пошуку шкіл дозволяє доволі гнучко підходити до питання знаходження школи, оскільки має багато різних варіацій використання і кожен хто його використовує може це робити згідно власних потреб.

4 РОБОТА РОЗРОБЛЕНОЇ ВЕБ-СИСТЕМИ

Все що необхідно для користування системою – мати доступ до мережі Інтернет. Користувач авторизується у системі і отримує доступ до функціоналу визначеного за його роллю.

4.1 Вимоги до використання та запуску системи

Вимоги до запуску системи варто розподілити на запуск клієнтського застосунку та серверної частини.

Для локального запуску проєкту на комп’ютері необхідне виконання таких вимог:

- операційна система Windows, оскільки обрана система керування базами даних Microsoft SQL Server накладає обмеження на тип операційної системи;
- встановлений SDK який підтримує ASP.NET Core та Blazor WASM, для цього проєкту це .NET 6. Для зручності рекомендовано встановити IDE який одразу завантажує SDK та надає зручний інтерфейс для запуску проєкту;
- веб-браузер із підтримкою WebAssembly, наприклад: Google Chrome, Microsoft Edge, Safari, Firefox.

Варто зауважити, що локальний запуск проєкту має сенс лише при розробці продукту або для демонстрації отриманих результатів. Для правильної роботи проєкту необхідно виконати його хостинг. Це означає, що він буде запущений на сервері і для того аби його використати необхідно буде лише інтернет з’єднання та браузер. Для хостингу може бути використаний раніше згаданий сервіс Azure.

4.2 Функціонал застосунку

Огляд функціоналу розробленого застосунку важливо розділити на окремі частини у яких буде розглянуто функції доступні користувачу відповідно до встановленої йому ролі.

4.2.1 Огляд функціоналу доступного неавторизованому користувачу

Розпочнемо з неавторизованого або незареєстрованого користувача. На рисунку 4.1 зображено головний екран який бачить неавторизований або незареєстрований користувач, з нього вони можуть перейти на сторінки авторизації (логін) або реєстрації.

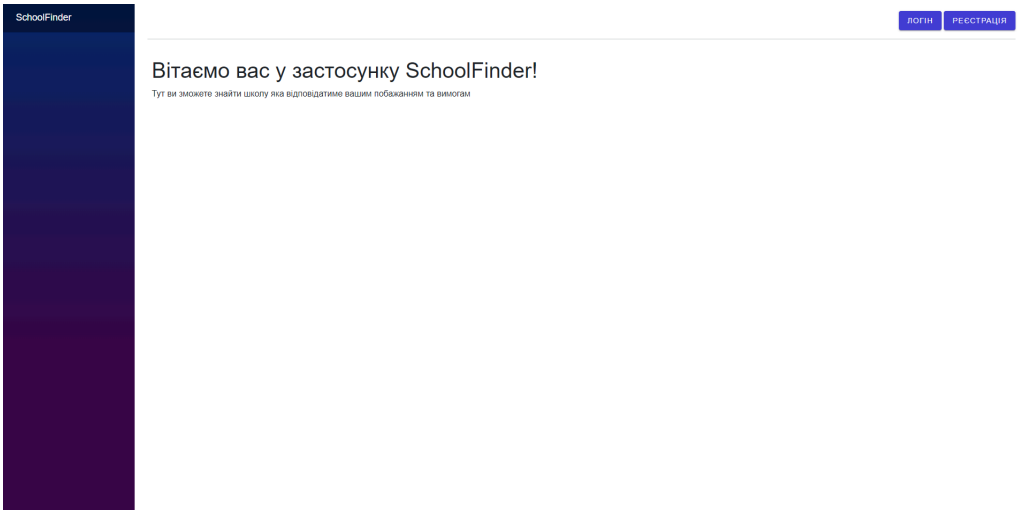

Рисунок 4.1 – Зображення головного екрану для незареєстрованого або неавторизованого користувача

На рисунку 4.2 можемо бачити форму авторизації користувача, при правильно введених даних відкривається головний екран застосунку із відповідним до ролі функціоналом. У випадку невірно введених даних відображується відповідне сповіщення. Також варто зазначити що з цієї сторінки можна перейти на сторінку реєстрації.

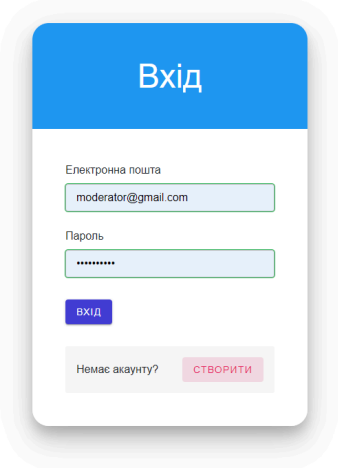

Рисунок 4.2 – Зображення сторінки авторизації користувача

У випадку коли користувач ще не має акаунту, він переходить на сторінку реєстрації, де відповідно до власних потреб заповнює необхідну форму. На рисунку 4.3 ми бачимо форму реєстрації учня, після заповнення цієї форми створюється запит який буде розглянуто адміністратором і, у випадку відповідності даних, користувач буде доданий до системи.

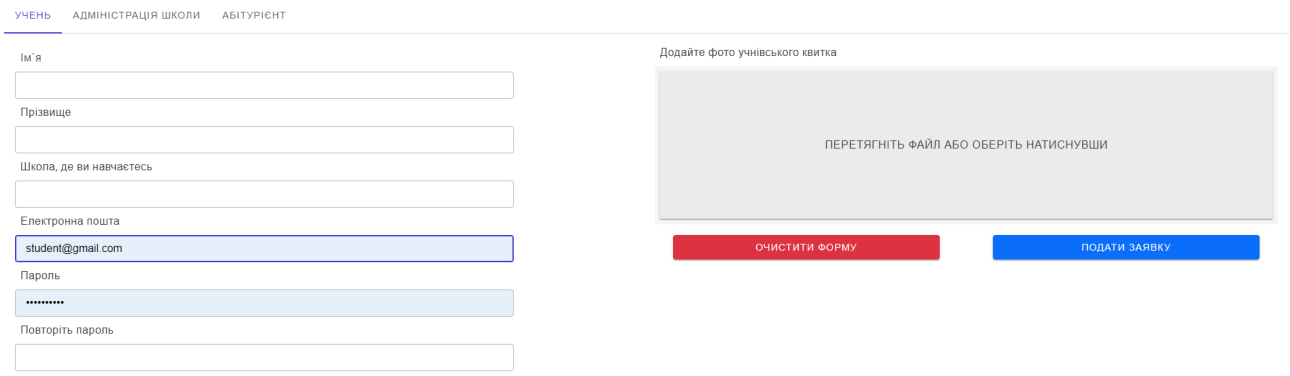

Рисунок 4.3 – Форма реєстрації акаунту учня

Для створення акануту для адміністратора школи необхідно заповнити форму, яка зображена на рисунку 4.4, та додати фото документа підтверджуючого його посаду у даній школі. Далі створюється запит на реєстрацію який розглядається адміністратором і у випадку відповідності даних створюється акаунт.

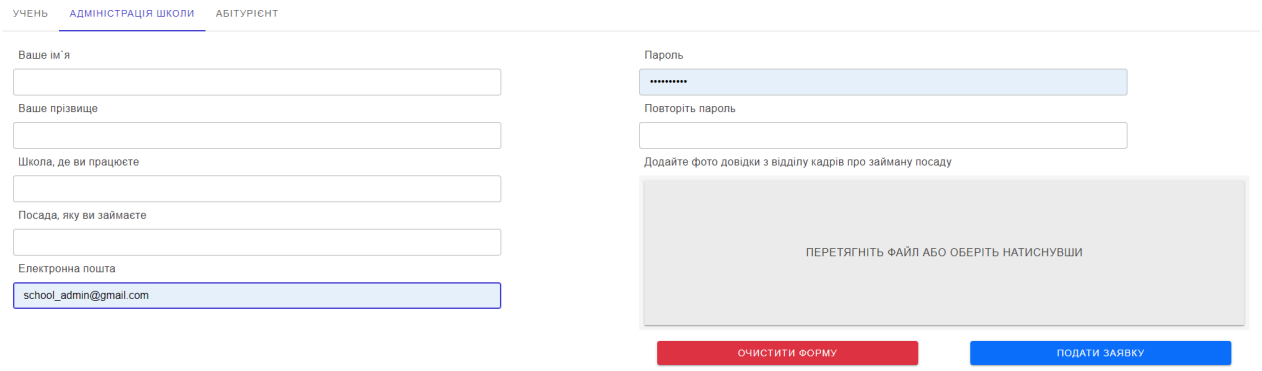

Рисунок 4.4 – Форма реєстрації адміністратора школи

Останньою опцією на екрані реєстрації є створення акаунта абітурієнта, форма для цього зображена на рисунку 4.5. Користувач із такою роллю має доступ тільки до перегляду інформації, тому створення такого акаунту відбувається одразу.

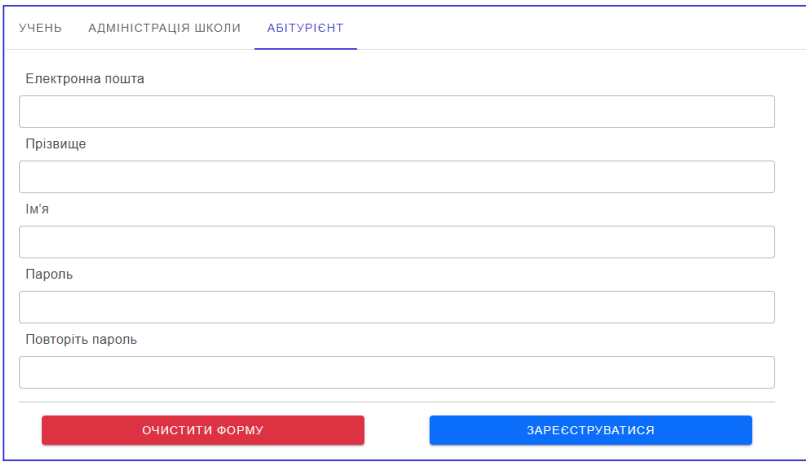

Рисунок 4.5 – Форма реєстрації абітурієнта

4.2.2 Огляд функціоналу доступного абітурієнту

Як було зазначено вище, користувач має право тільки на перегляд інформації, тому у меню йому доступна тільки сторінка «Школи». Вигляд меню наведено на рисунку 4.6.

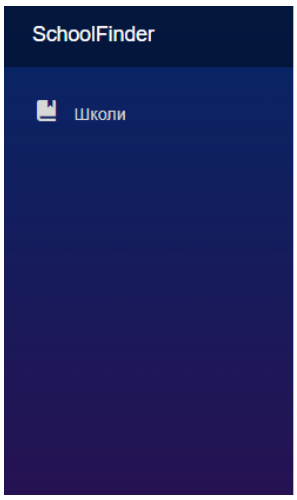

Рисунок 4.6 – Вигляд меню для користувача із роллю «Абітурієнт»

При переході на сторінку «Школи», зображення якої наведено на рисунку 4.7, ми бачимо, що користувачу доступний перегляд інформації про школи.

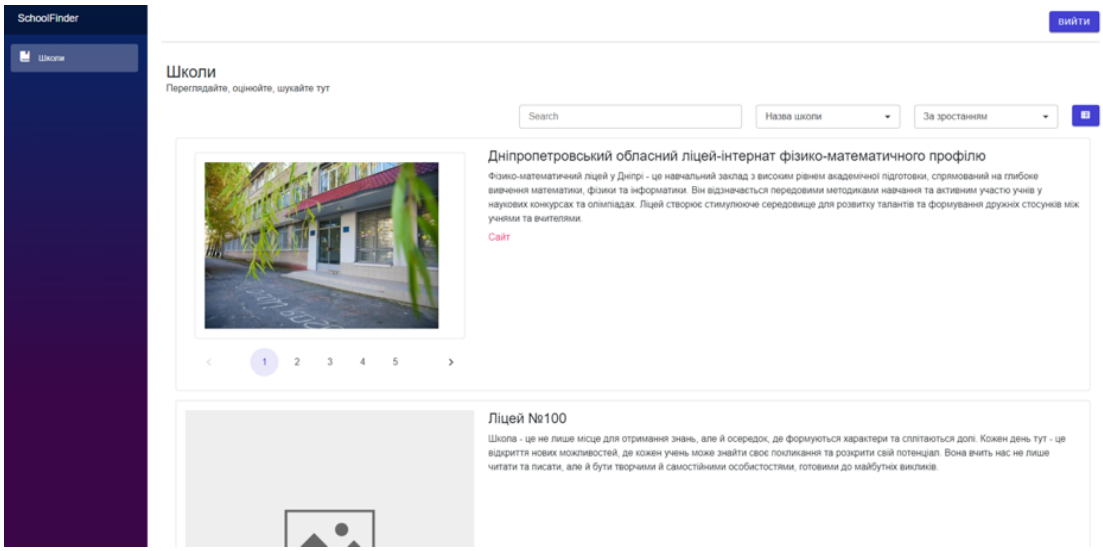

Рисунок 4.7 – Вигляд сторінки «Школи»

Для пошуку бажаного закладу, користувач має можливість встановити фільтри відповідно до його потреб. На фото 4.8 зображено інтерфейс для встановлення фільтрів.

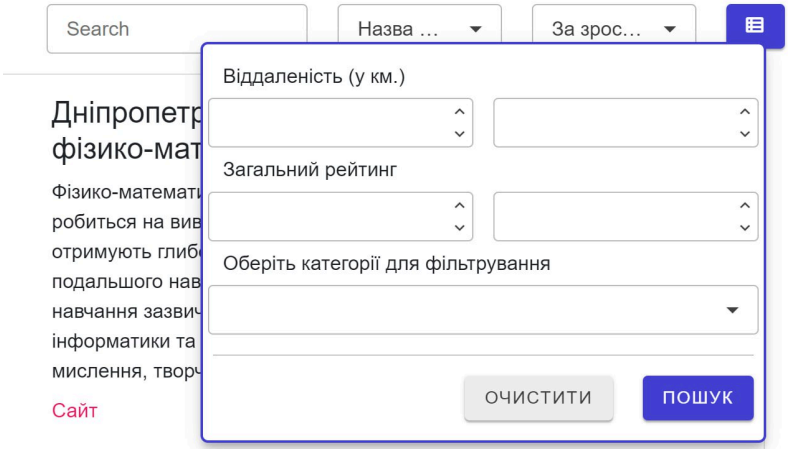

Рисунок 4.8 – Інтерфейс встановлення фільтрів для пошуку шкіл

Розширена інформація про школи відкривається у додатковому вікні, його зображено на рисунку 4.9, тут користувач може переглядати фотографії надані адміністрацією школи, прочитати розгорнуту інформацію про заклад, переглянути локацію, статистику про заклад та коментарі від учнів.

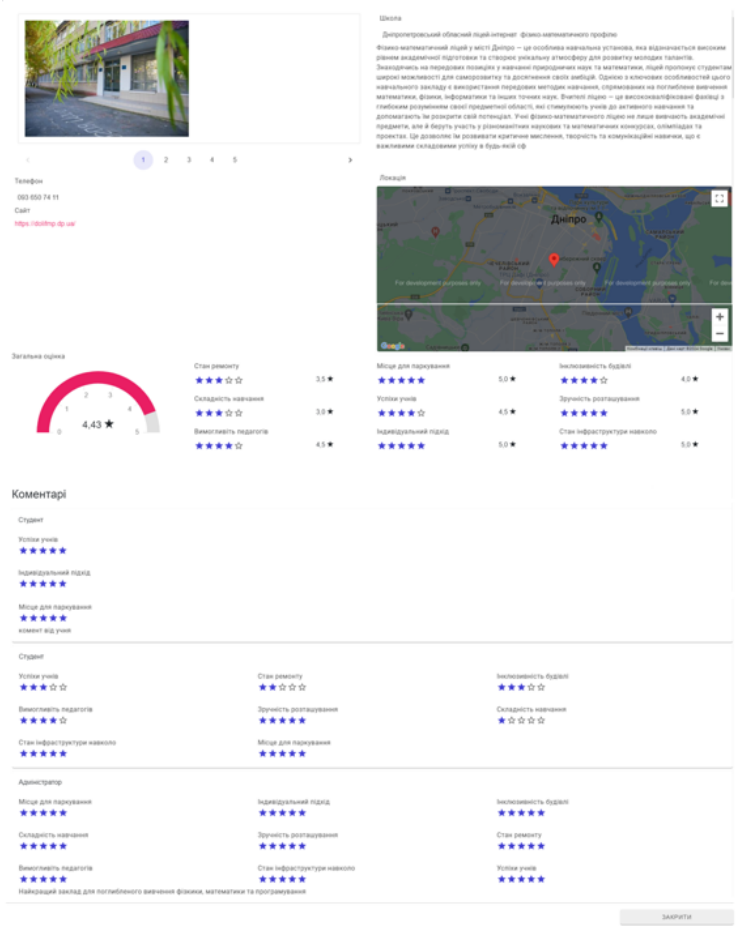

Рисунок 4.9 – Зображення вікна із інформацією про школу

4.2.3 Огляд функціоналу доступного учню

Дана роль розширює можливості попередньої, тому розглянемо саме відмінності.

На рисунку 4.10 зображено частину вікна із інформацією про школу, як бачимо, для даної ролі доступна кнопка для додавання відгуку про заклад, натиснувши на неї відкривається форма де учень встановлює рейтинги до необхідних категорій та додає коментар. Після заповнення, коментар спочатку проходить модерацію і потім вже відображається на сторінці.

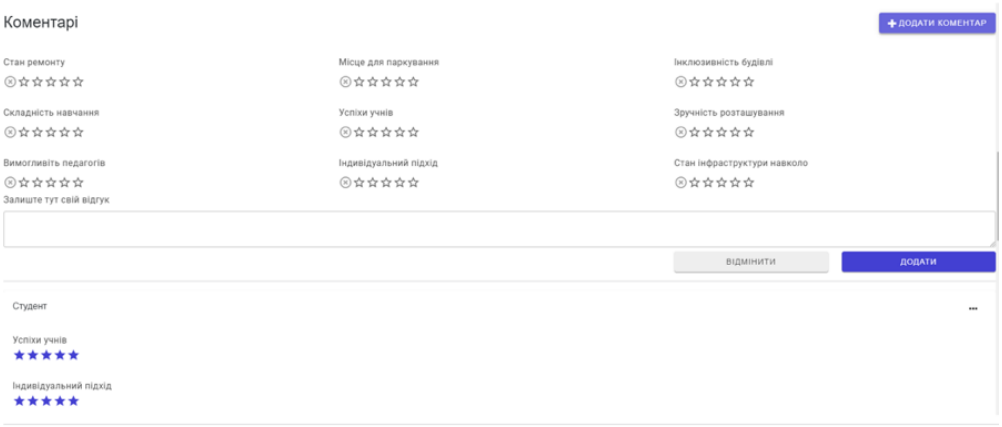

Рисунок 4.10 – Вікно із інформацією про школи для користувача із роллю «Учень»

Також для залишених користувачем коментарів доступні опції редагування та видалення. На рисунку 4.11 наведено зображення використання цього функціоналу.

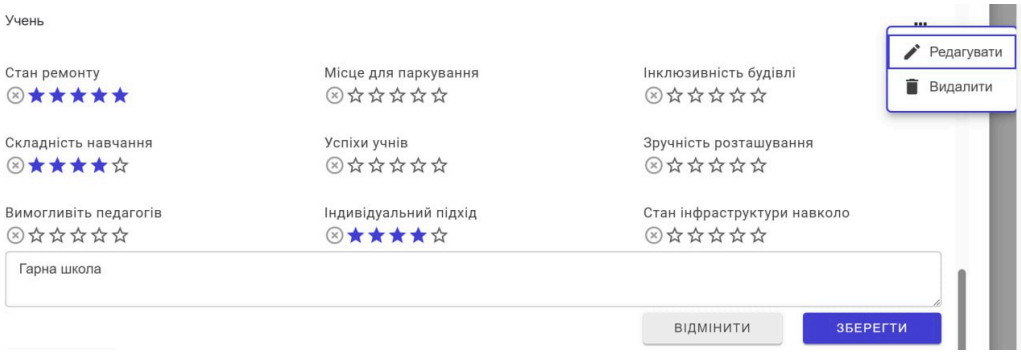

Рисунок 4.11 – Інтерфейс редагування та видалення коментаря

4.2.4 Огляд функціоналу доступного шкільному адміністратору

Для користувачів із роллю «Шкільний адміністратор», меню має розширений вид, вони можуть переглядати окремо заклади створені ними. Розширене меню та сторінка створених ними закладів зображене на рисунку 4.12.

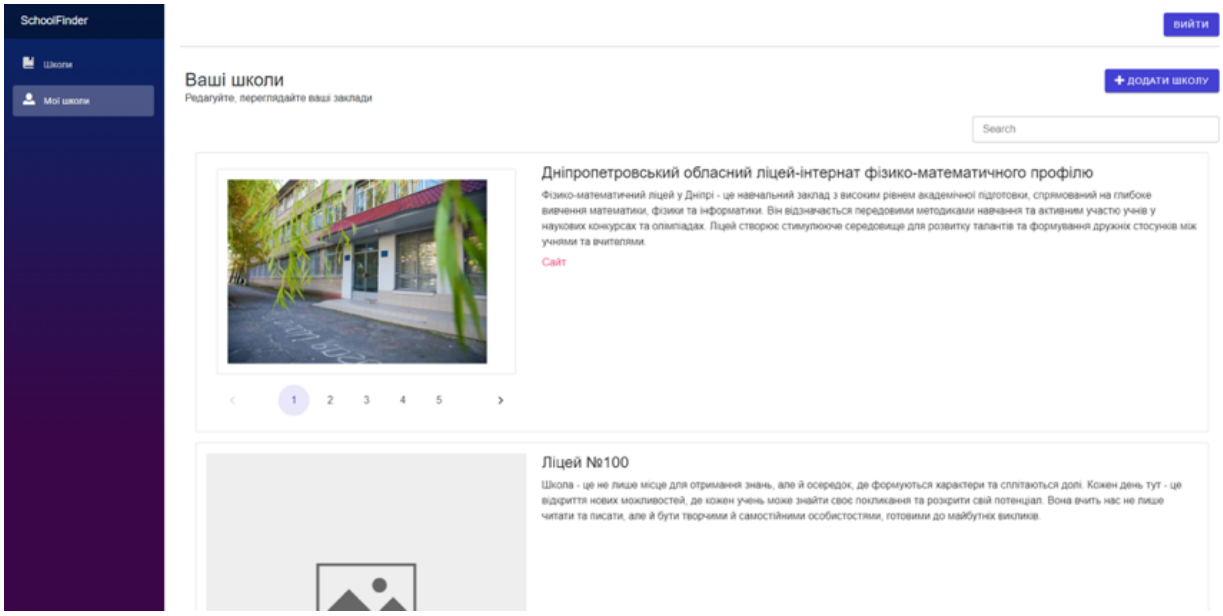

Рисунок 4.12 – Сторінка створених користувачем закладів та вигляд розширеного меню

Для користувачів із цією роллю вікно з інформацією має дещо інший вигляд, воно відкривається в режимі редагування, як це відображено на рисунку 4.13.

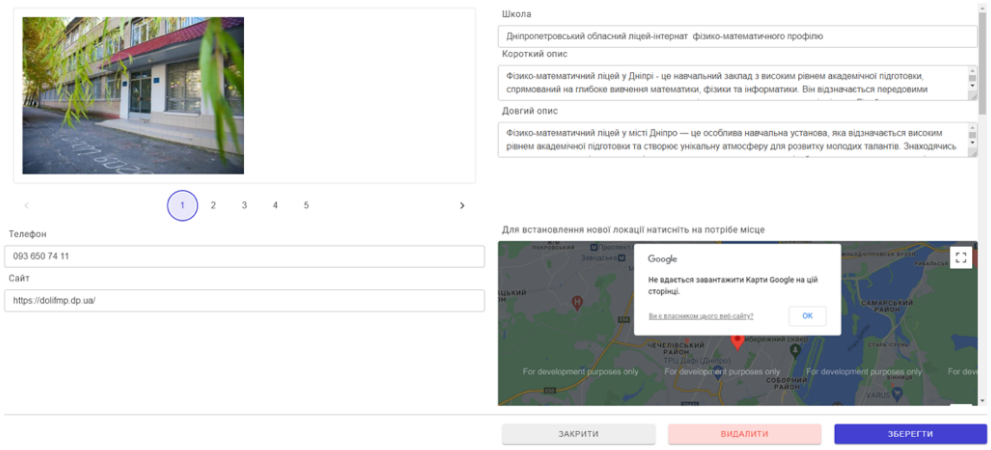

Рисунок 4.13 – Вікно з інформацією про школу в режимі редагування

Також шкільний адміністратор може подавати заявки на створення нових закладів, для цього є відповідна кнопка «Подати запит». Натиснувши її відкривається форма, зображена на рисунку 4.14, після заповнення якої створюється запит. У разі підтвердження адміністратором, додається новий заклад із наданої інформацією.

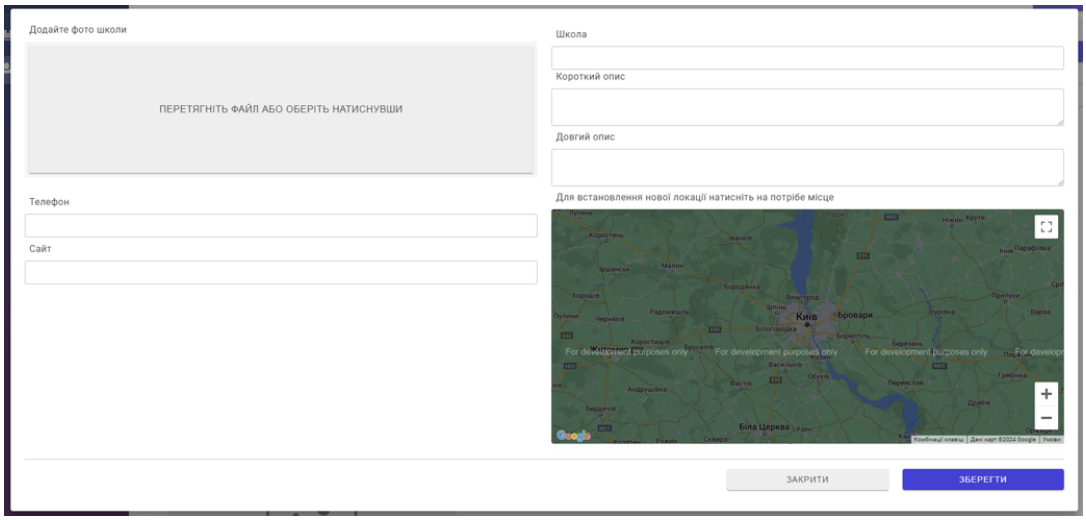

Рисунок 4.14 – Форма для створення нового закладу

Також адміністратор школи може відповідати на коментарі залишені до його закладу, тим самим підтримуючи зв’язок із коментаторами.

4.2.5 Огляд функціоналу доступного модератору

Ролі, розглянути далі, є службовими ролями, тобто тими що відповідають за контроль над системою. До таких відносяться: модератор, адміністратор та супер адміністратор.

Для ролі модератор доступне редагування будь-якого контенту в системі. Він може змінювати або видаляти інформацію про школи, коментарі, оцінки тощо. Також модератор відповідальний за розгляд запитів на додавання коментарів, для цього створено відповідний пункт в меню та сторінку на якій відображено інформацію. На рисунку 4.15 зображено цю сторінку.

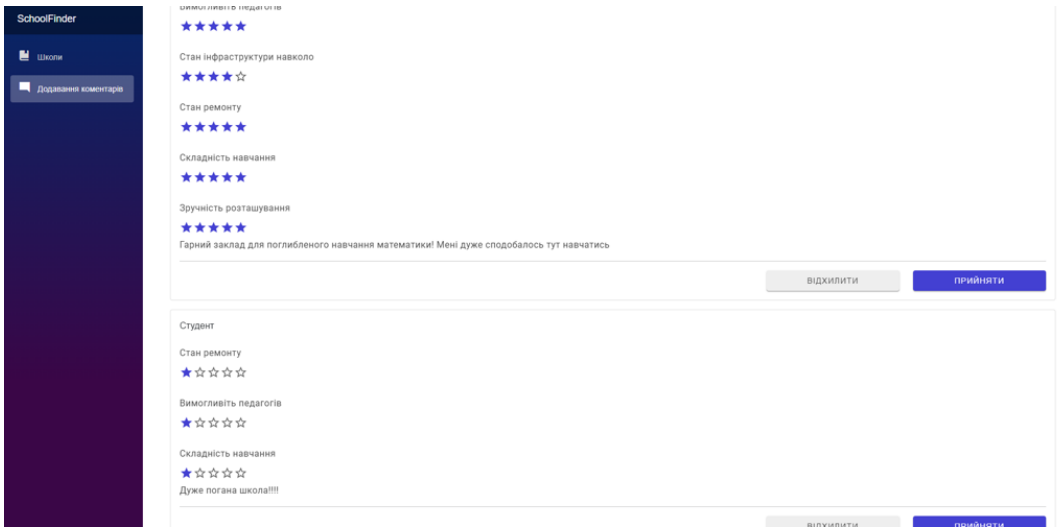

Рисунок 4.15 – Сторінка обробки запитів на додавання коментарів

4.2.6 Огляд функціоналу доступного адміністратору

Для ролі «Адміністратор» створено розширене меню що включає додаткові пункти для сторінок із запитами на створення акаунтів та шкіл. На рисунку 4.16 зображено це меню.

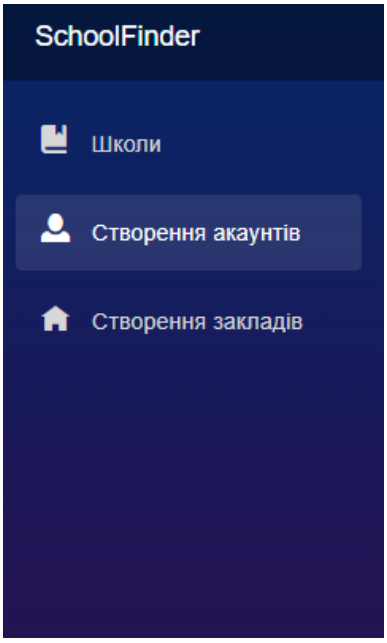

Рисунок 4.16 – Вигляд розширеного меню для ролі «Адміністратор»

Адміністратор відповідальний за обробку запитів на створення акунтів, функціонал створений для цього зображено на рисунку 4.17.

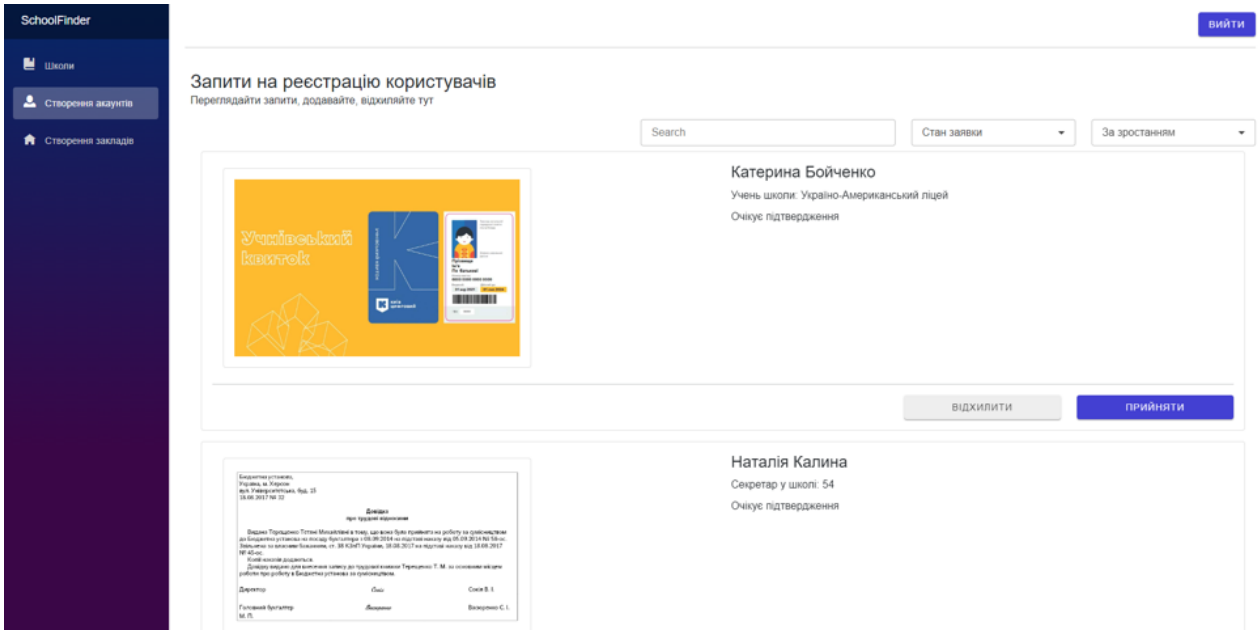

Рисунок 4.17 – Сторінка обробки запитів на створення акаунтів

Також адміністратор повинен обробляти запити на створення закладів, для цього створена відповідна сторінка, вона зображена на рисунку 4.18.

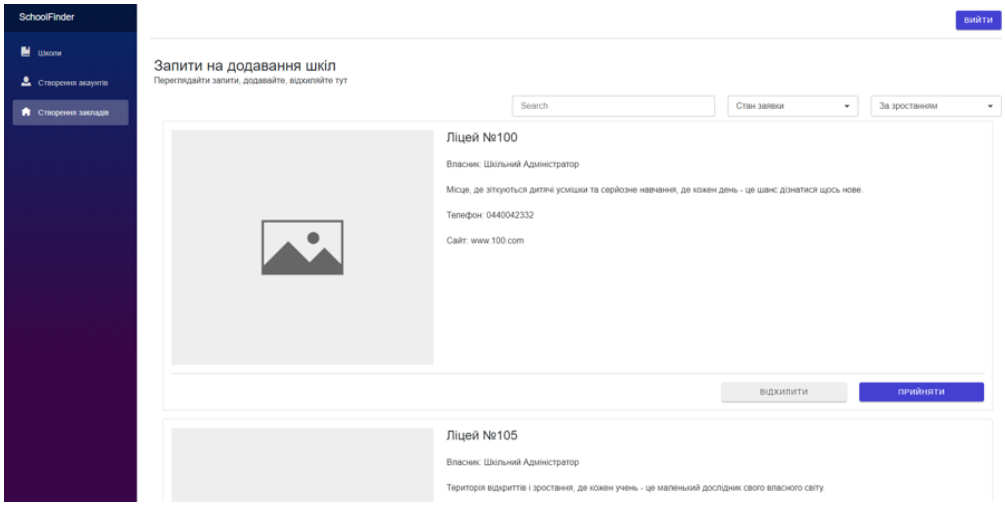

Рисунок 4.18 – Сторінка обробки запитів на створення закладів

Ще однією функціональною можливістю адміністратора є створення шкіл для будь-якого користувача, для цього на сторінці «Школи» є необхідна кнопка яка відкриває вікно зображене на рисунку 4.14, єдина відмінність у тому що є додаткове поле у якому адміністратор обирає користувача для якого створюється

заклад. Після заповнення форми школа одразу додається до системи, минаючи етап створення запиту.

4.2.7 Огляд функціоналу доступного супер адміністратору

Останньою роллю є «Супер адміністратор», вона створена для встановлення або зміни ролей користувачів. Для цього створено відповідну сторінку де відображаються користувачі та їх роль. Її зображено на рисунку 4.19.

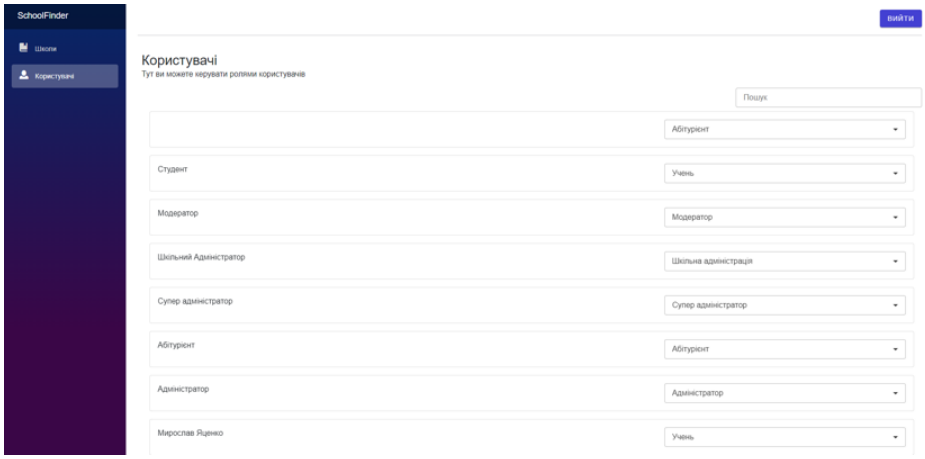

Рисунок 4.19 – Сторінка зміни ролей користувачів

У випадку необхідності зміни ролі, користувач обирає нову із випадаючого списку після чого вона автоматично оновлюється.

ВИСНОВКИ

В ході виконання завдання дипломної роботи було розроблено веб-систему для пошуку та рейтингування навчальних закладів. Даний веб-застосунок має на меті покращення та спрощення процесу вибора навчальних закладів школярами.

1. На основі аналізу підходів, методів виконання поставленої задачі обґрунтовано використання підходу до розробки Agile, методології Scrum та мануального тестування отриманих результатів. Даний вибір було зроблено оскільки такий підхід дозволяє гнучко та швидко реагувати на потреби розробки, а обрана методологія поділяє процес розробки на спринти, під час яких можна більш досконало виконувати поставлені задачі. Вибір мануального тестування був зумовлений тим, що у розробленому застосунку його використання займе менше часу та буде більш інформативним у випадку виявлення проблем.

2. На основі аналізу програмних засобів обґрунтовано використання продуктів від компанії Microsoft, а саме: технологія ASP.NET Core, Blazor, Entity Framework Core та Azure. Було зроблено такий вибір оскільки ці засоби є одними з передових на сьогоднішній день, а враховуючи те що їх розробником є одна компанія це означає виникне менше проблем під час їх інтеграції одне з одним.

3. В результаті виконання поставленого завдання було розроблено програмну веб-систему, що покращує школярам вибір навчального закладу. Даний застосунок має розподіл ролей користувачів. Встановлено такі ролі та доступний функціонал для користувачів:

- незареєстрований користувач, має змогу подати заявку на створення акаунта учня або шкільного адміністратора додавши фото необхідного документа або створити акаунт абітурієнта і одразу користуватися системою.
- абітурієнт, може переглядати заклади та їх рейтинг;
- учень, може те саме що і абітурієнт і додатково залишати відгуки про заклади;
- шкільний адміністратор, може додавати або видаляти школи, оновлювати про них інформацію. Є представником від школи;

- модератор, людина яка може редагувати усю інформацію в системі, слідкує за відповідністю контенту до політики системи;
- адміністратор, додатково до можливостей модератора відповідає за розглядання заявок на створення акаунтів та закладів;
- супер адміністратор, людина відповідальна за розподілення ролей серед користувачів.

Такий розподіл ролей дає змогу встановити контроль над системою відповідальними за це людьми.

За допомогою інтеграції необхідних сервісів було досягнуто розширення можливостей застосунку та зроблено його більш простим для використання.

4. На основі проведеного тестування було доведено коректність роботи розробленої веб-системи для пошуку та рейтингування навчальних закладів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коноваленко І.В. Програмування мовою C# 6. – Тернопіль: ТНТУ, 2016. – 227 с.
2. Ian Griffiths. Programming C# 8.0 Build cloud, Web, and Desktop Applications. First Edition. Видавництво O'Reilly Media, 2020.
3. C# documentation. URL: https://docs.microsoft.com/en-us/dotnet/csharp/.
4. Вступ до ASP.NET Core. URL: https://krypton.com.ua/rozdil-1-vstup-do-asp-net-core.
5. ASP.NET Core Documentation. URL: https://docs.microsoft.com/en-us/aspnet/core.
6. Overview of Entity Framework Core. URL: https://learn.microsoft.com/ef/core.
7. Архітектура клієнт-сервер. Сховища даних. URL: http://educational.mariroz.com/InformTechVInfrastrRynku/lect/lect8.pdf.
8. What is Agile? URL: https://www.atlassian.com/agile.
9. Scrum Sprints: Everything You Need to Know. URL: https://www.atlassian.com/agile/scrum/sprints.
10. JWT Authentication And Authorization In .NET 6.0 with Identity Framework. URL: https://www.c-sharpcorner.com/article/jwt-authentication-and-authorization-in-net-6-0-with-identity-framework.
11. What is Blazor vs (Angular, React and Vue). URL: https://medium.com/@devathon_/blazor-vs-react-angular-vue-390c373869b9.
12. Blazor – Build client web apps with C#. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor.
13. Radzen Blazor Components, набір безкоштовних нативних компонентів інтерфейсу. URL: https://www.radzen.com/blazor-components.
14. Azure: Blob storage. URL: https://rtfm.co.ua/azure-blob-storage/#pll_switcher.
15. Документація Google Maps Platform. URL: https://developers.google.com/maps/documentation.
16. Git Documentation. URL: https://git-scm.com/book/en/v2.

17. Рейтинг мов програмування 2024. URL: https://dou.ua/lenta/articles/language-rating-2024/.
18. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch and others. URL: https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/.
19. Comparison between Blazor vs Angular vs React vs Vue. URL: https://www.angularminds.com/blog/comparison-between-blazor-vs-angular-vs-react-vs-vue.

ДОДАТОК А

ТЕКСТ ПРОГРАМНОГО МОДУЛЯ

«РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ТА СЕРВІСІВ ВЕБ-ЗАСТОСУНКУ »

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-01_24Б

Аркушів 7

Київ – 2024

Лістинг програми

До розділу 3.2:

SchoolController.cs

```
public class SchoolService : ServiceClient
   {
      public SchoolService(HttpClient client, SessionStorage sessionStorage) : base(client,
sessionStorage)
      {
      }

      public async Task<bool> Add(SchoolDto school)
      {
         HttpResponseMessage response = await PostAsJsonAsync("create", school);
         return await response.Content.ReadFromJsonAsync<bool>();
      }

      public async Task<bool> Delete(Guid id)
      {
         HttpResponseMessage response = await DeleteAsync($"delete/{id}");
         return await response.Content.ReadFromJsonAsync<bool>();
      }

      public async Task<PagedList<SchoolDto>> Find(SchoolFilter filter)
      {
         HttpResponseMessage response = await PostAsJsonAsync("find", filter);
         return (await response.Content.ReadFromJsonAsync<PagedList<SchoolDto>>())!;
      }

      public async Task<bool> UpdateSchool(SchoolDto school)
      {
         HttpResponseMessage response = await PutAsJsonAsync("update", school);
         return await response.Content.ReadFromJsonAsync<bool>();
      }
   }
```

AuthController.cs

```
public class SchoolService : ServiceClient
   {
      public SchoolService(HttpClient client, SessionStorage sessionStorage) : base(client,
sessionStorage)
      {
      }

      public async Task<bool> Add(SchoolDto school)
      {
         HttpResponseMessage response = await PostAsJsonAsync("create", school);
         return await response.Content.ReadFromJsonAsync<bool>();
      }
```

```
    public async Task<bool> Delete(Guid id)
    {
        HttpResponseMessage response = await DeleteAsync($"delete/{id}");
        return await response.Content.ReadFromJsonAsync<bool>();
    }

    public async Task<PagedList<SchoolDto>> Find(SchoolFilter filter)
    {
        HttpResponseMessage response = await PostAsJsonAsync("find", filter);
        return (await response.Content.ReadFromJsonAsync<PagedList<SchoolDto>>())!;
    }

    public async Task<bool> UpdateSchool(SchoolDto school)
    {
        HttpResponseMessage response = await PutAsJsonAsync("update", school);
        return await response.Content.ReadFromJsonAsync<bool>();
    }
}
```

ReplyService.cs

```
public class ReplyService
{
    private readonly CommentStore _commentStore;
    private readonly ReplyStore _replyStore;
    private readonly UserManager<User> _userManager;

    public ReplyService(CommentStore commentStore, ReplyStore replyStore, UserManager<User>
userManager)
    {
        _commentStore = commentStore;
        _replyStore = replyStore; ;
        _userManager = userManager;
    }

    public async Task<int> Add(ReplyDto dto)
    {
        Reply entity = dto.ToModel();
        Comment? comment = await _commentStore.Get(dto.CommentId);
        User user = await _userManager.FindByIdAsync(dto.CreatedBy.Id);
        entity.Comment = comment!;
        entity.CreatedBy = user;
        return await _replyStore.Add(entity);
    }

    public async Task<int> Delete(Guid Id)
    {
        Reply? entity = await _replyStore.Get(Id);
        if (entity == null)
        {
            return 0;
        }
        return await _replyStore.Delete(entity);
    }
```

```
    public async Task<PagedList<ReplyDto>> Find(ReplyFilter filter)
    {
        List<Reply> replies = await _replyStore.Find(filter);
        int totalCount = await _replyStore.TotalCount(filter);
        return new PagedList<ReplyDto>(replies.Select(r => r.ToDto()), totalCount);
    }

    public async Task<int> Update(ReplyDto dto)
    {
        Reply? entity = await _replyStore.Get(dto.Id);
        if (entity == null)
        {
            return 0;
        }
        entity.Message = dto.Message;
        return await _replyStore.Update(entity);
    }
}
```

AuthService.cs

```
public class AuthService
{
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;
    private readonly JwtConfiguration _jwtConfiguration;

    public AuthService(UserManager<User> userManager, RoleManager<IdentityRole>
roleManager, JwtConfiguration jwtConfiguration)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _jwtConfiguration = jwtConfiguration;
    }

    public async Task<LoginResponseModel> Login(LoginModel model)
    {
        var user = await _userManager.FindByEmailAsync(model.Email);
        if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
        {
            var userRoles = await _userManager.GetRolesAsync(user);

            var authClaims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Email),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            };

            foreach (var userRole in userRoles)
            {
                authClaims.Add(new Claim(ClaimTypes.Role, userRole));
            }
```

```
            var token = GetToken(authClaims);

            return new LoginResponseModel(){
                Token = new JwtSecurityTokenHandler().WriteToken(token),
                ExpireOn = token.ValidTo,
                User = user.ToDto(userRoles.ToList())
            };
        }
        return new LoginResponseModel();
    }

    private JwtSecurityToken GetToken(List<Claim> authClaims)
    {
        var authSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtConfiguration.Secret));

        var token = new JwtSecurityToken(
            issuer: _jwtConfiguration.ValidIssuer,
            audience: _jwtConfiguration.ValidAudience,
            expires: DateTime.Now.AddHours(12),
            claims: authClaims,
            signingCredentials: new SigningCredentials(authSigningKey,
SecurityAlgorithms.HmacSha256)
            );

        return token;
    }
}
```

RegistrationService.cs

```
public class RegistrationService
{
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public RegistrationService(UserManager<User> userManager, RoleManager<IdentityRole>
roleManager)
    {
        _userManager = userManager;
        _roleManager = roleManager;
    }

    public async Task<bool> Registrate(RegistrationModel model)
    {
        var userExists = await _userManager.FindByEmailAsync(model.Email);
        if (userExists != null)
            return false;

        User user = new User()
        {
            FirstName = model.FirstName,
            LastName = model.LastName,
            Email = model.Email,
```

```
            UserName = model.Email,
            SecurityStamp = Guid.NewGuid().ToString()
        };
        var result = await _userManager.CreateAsync(user, model.Password);
        if (!result.Succeeded)
            return false;

        if (!await _roleManager.RoleExistsAsync(model.Role))
            await _roleManager.CreateAsync(new IdentityRole(model.Role));

        await _userManager.AddToRoleAsync(user, model.Role);

        return true;
    }
}
```

До розділу 3.3:

AppState.cs

```
public class AppState
    {
        public string Token { get; set; } = string.Empty;
        public UserDto? User { get; set; }
        public bool IsAuthenticated
        {
            get
            {
                return !string.IsNullOrEmpty(Token);
            }
        }

        public AppState(string token, UserDto user)
        {
            Token = token;
            User = user;
        }
        public AppState() {
        }
    }
```

FinderComponent.cs

```
public class FinderComponent : ComponentBase
    {
        public AppState State { get; set; } = new AppState();
        [Inject]
        public FinderNavigationManager NavigationManager { get; set; } = null!;
        [Inject]
        public SessionStorage SessionStorage { get; set; } = null!;

        protected override async Task OnInitializedAsync()
        {
            State = (await SessionStorage.Get<AppState>("AppState")) ?? new AppState();
```

```
    }
  }
```

FinderNavigationManager.cs

```
public class FinderNavigationManager
  {
    private readonly NavigationManager _navigator;

    public string Uri => _navigator.Uri;

    public FinderNavigationManager(NavigationManager navigator)
    {
      _navigator = navigator;
    }

    public void NavigateTo(string uri, bool forceLoad = false)
    {
      _navigator.NavigateTo(uri, forceLoad);
    }

    public void NavigateToLogin()
    {
      _navigator.NavigateTo("/login");
    }

    public void NavigateToLogout()
    {
      _navigator.NavigateTo("/logout");
    }

    public void NavigateToRegistration()
    {
      _navigator.NavigateTo("/registration");
    }

    public Uri ToAbsoluteUri(string relativeUri)
    {
      return _navigator.ToAbsoluteUri(relativeUri);
    }
  }
```

SessionStorage.cs

```
public class SessionStorage
  {
    private readonly ISessionStorageService _storage;

    public SessionStorage(ISessionStorageService storage)
    {
      _storage = storage;
    }

    public async Task Clear()
    {
```

```
        await _storage.ClearAsync();
    }

    public async Task<T?> Get<T>(string key) where T : class
    {
        if(await _storage.ContainKeyAsync(key))
        {
            return await _storage.GetItemAsync<T>(key);
        }

        return null;
    }

    public async Task Set(string key, object value)
    {
        await _storage.SetItemAsync(key, value);
    }
}
```

SchoolService.cs

```
public class SchoolService : ServiceClient
{
    public SchoolService(HttpClient client, SessionStorage sessionStorage) : base(client, sessionStorage)
    {
    }

    public async Task<bool> Add(SchoolDto school)
    {
        HttpResponseMessage response = await PostAsJsonAsync("create", school);
        return await response.Content.ReadFromJsonAsync<bool>();
    }

    public async Task<bool> Delete(Guid id)
    {
        HttpResponseMessage response = await DeleteAsync($"delete/{id}");
        return await response.Content.ReadFromJsonAsync<bool>();
    }

    public async Task<PagedList<SchoolDto>> Find(SchoolFilter filter)
    {
        HttpResponseMessage response = await PostAsJsonAsync("find", filter);
        return (await response.Content.ReadFromJsonAsync<PagedList<SchoolDto>>())!;
    }

    public async Task<bool> UpdateSchool(SchoolDto school)
    {
        HttpResponseMessage response = await PutAsJsonAsync("update", school);
        return await response.Content.ReadFromJsonAsync<bool>();
    }
}
```