БАКАЛАВРСЬКА РОБОТА

БР.КІ-31.00.00.000 ПЗ

Група КІ-21-2

Дуда Назар

2025

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій

Кафедра комп'ютерних систем і мереж

Дуда Назар Вікторович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Сайт для адміністрування та обліку даними наукового містечка "Нова енергія" з використанням ASP .NET Core MVC, entity framework та jquery

Комп'ютерна інженерія
(назва освітньої програми)

123 – Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього ступеня _____ *Дуда Н.В.* _____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ *Гарасимів Т.Г., асист.* _____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри
д.т.н., професор /С. І. Мельничук/
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025 рік

Івано-Франківський національний технічний університет нафти і газу

Інститут *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ:

Зав. кафедрою КСМ

С.І. Мельничук

« 05 » травня 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ.

Дуда Назару Вікторовичу

1. Тема проекту (роботи*) Сайт для адміністрування та обліку даними наукового містечка "Нова енергія" з використанням ASP .NET Core MVC, entity framework та jquery*

керівник проєкту (роботи) *Гарасимів Тарас Григорович, асистент*.

затверджені наказом вищого навчального закладу від *05. 05. 2025 №275/7*

2. Строк подання студентом проекту (роботи) *12 червня 2025р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1.Інформаційне забезпечення інтернет сервісу для адміністрування та обліку даних 2. Розробка структурних рішень веб-сайту для адміністрування та обліку даних 3. Розробка та реалізація веб-сайту для адміністрування та обліку даних*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____.

6. Консультанти розділів роботи

7. Дата видачі завдання *29 січня 2025 р.* .

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Збір інформації, вивчення літератури та пошук додаткової інформації*	*Лютий 2025р*	
2	*Інформаційне забезпечення інтернет-сервісу продажу квитків*	*Березень 2025р*	
3	*Розробка структурних рішень веб-сайту продажу квитків*	*Квітень 2025р*	
4	*Реалізація графічного інтерфейсу та бази даних веб-сайту продажу квитків*	*Травень 2025р*	
5	*Оформлення додатків, дипломної роботи*	*Червень 2025р*	

Студент _____ *Дуда Н. В.*

Керівник роботи _____ *Гарасимів Т. Г.*

АНОТАЦІЯ

Бакалаврська робота присвячена проєктуванню та розробці веб-сайту для адміністрування та обліку даних наукового містечка "Нова Енергія".

Метою роботи є створення інформаційної, структурної, алгоритмічної та програмної складових веб-застосунку для управління послугами музею та їх обліку із застосуванням C#, ASP.NET Core MVC, Entity Framework та jQuery.

У процесі виконання бакалаврської роботи було проаналізовано специфіку роботи музею і які послуги він може надати, на основі цього сформовано технічне завдання із замовниками. Створено структуру бази даних, таблиці та діаграми, що ілюструють логіку роботи веб-застосунку. Розроблено інтерфейс сайту і також всі необхідні сторінки з згідно завдання.

В результаті роботи створено повноцінний веб-сайт для ефективного обліку, контролю та адміністрування сервісів наукового містечка "Нова Енергія".

Ключові слова: ASP.NET Core MVC, Entity Framework, облік даних, об’єкт.

ANNOTATION

The bachelor's thesis is devoted to the design and development of a website for the administration and accounting of the "New Energy" science campus.

The purpose of the work is to create information, structural, algorithmic and software components of a web application for managing museum services and their accounting using C#, ASP.NET Core MVC, Entity Framework and jQuery.

In the process of performing the bachelor's thesis, the specifics of the museum's work and what services it can provide were analyzed, based on this, a technical task was formed with customers. A database structure, tables and diagrams were created that illustrate the logic of the web application. The site interface was developed, as well as all the necessary pages according to the task.

As a result of the work, a full-fledged website was created for effective accounting, control and administration of the "New Energy" science campus services.

Keywords: ASP.NET Core MVC, Entity Framework, data accounting, object

ЗМІСТ

ВСТУП

У сучасному світі все більшу роль відіграє впровадження цифрових технологій та автоматизованих рішень у процеси управління, обліку та організації роботи. Особливо це актуально для освітніх та наукових закладів, де зростає потреба у створенні зручних веб-сервісів для управління внутрішніми процесами та обліку наданих послуг.

Актуальність роботи: обумовлена потребою створити зручний веб-сервіс для адміністрування та обліку сервісів наукового містечка "Нова Енергія". Існуючі системи часто є складними для використання в невеликих організаціях. Запропонований веб-сайт на базі ASP.NET Core MVC, Entity Framework буде простим, швидким, вимагатиме мінімальних ресурсів і орієнтується виключно на адміністративний персонал та працівників.

Об’єктом дослідження: процес створення веб-сервісу для обліку та адміністрування сервісів наукового містечка "Нова Енергія".

Предметом дослідження: розробка, оптимізація та впровадження інформаційної системи обліку і аудиту послуг музею.

Метою роботи: проєктування та реалізація інформаційної, структурної, алгоритмічної та програмної складових веб-застосунку для зберігання, обліку, управління та аналізу даних про послуги, що надаються науковим містечком "Нова Енергія".

Методи дослідження: теорія інформації, теорія реляційних баз даних, методи проєктування веб-інтерфейсів, аналіз вимог та моделювання програмних систем.

Практичне значення: роботи полягає у створенні веб-сайту для зручного та ефективного адміністрування і аудиту послуг наукового містечка "Нова Енергія", що відповідає сучасним вимогам до швидкості, зручності та безпеки.

1 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕРНЕТ СЕРВІСУ ДЛЯ АДМІНІСТРУВАННЯ ТА ОБЛІКУ ДАНИХ

1.1 Опис інформаційного процесу

У межах даної роботи досліджується веб-сайт для адміністрування та обліку сервісів наукового містечка "Нова Енергія" як приклад галузевої інформаційної системи, що автоматизує процес управління та аудиту наданих послуг. Система реалізує інформаційний процес, який охоплює:

– збирання інформації адміністратором про послуги які буде надавати музей;

– адміністрування доступу працівників до системи;

– забезпечення зручного інтерфейсу для перегляду, редагування та аналізу інформації про послуги;

– ведення аудиту та контролю за наданими послугами.

Функціонування подібних інформаційних систем має відповідати чинному законодавству України та загальноприйнятим технічним стандартам розробки програмного забезпечення. У межах роботи були проаналізовані основні нормативні документи, що регулюють обробку даних, інформаційну безпеку та якість програмних продуктів:

1. Закон України «Про захист персональних даних» — регулює порядок збирання, зберігання та обробки персональних даних працівників та адміністраторів системи.

2. Закон України «Про інформацію» — визначає основні принципи доступу, використання та захисту інформації у цифрових системах.

3. Закон України «Про електронні комунікації» — регламентує функціонування інформаційних ресурсів в Інтернеті, зокрема у сфері обміну та зберігання даних.

4. Рекомендації OWASP та методичні матеріали щодо побудови безпечних інформаційних систем — охоплюють принципи безпеки, розмежування доступу, захисту від зовнішніх загроз та організації структур баз даних.

У процесі розробки та експлуатації веб-сайту ці документи створюють правову, організаційну та технічну основу для реалізації інформаційних процесів.

Головні інформаційні характеристики системи, що забезпечують її роботу, включають:

Для користувача:

– ім’я та прізвище: користувач коли реєструється на сайті повинен вказати своє прізвище та ім’я, для того щоб знати ідентифікувати в реальному житті чий це акаунт;

– номер телефону: користувач також повинен вказати номер телефону, цей збір інформації робиться в цілях компанії для комунікації із користувачем/працівником;

– email: такий самий принцип як і у номера телефону, для взаємодії із користувачем/працівником;

– день народження: інформація щоб компанія могла привітати працівника і цим покращити його ефективність;

– логін та пароль: для ідентифікації і доступу певного користувача.

Для послуги:

– дата/час початку: кожна послуга(у подальшому заходи) має дату і час початку;

– час закінчення: кожна активність має час початку, залежить він від того скільки триває активність;

– тип: кожна активність ділиться на типи наприклад: Кріо-шоу/Тесла і тд;

– зал: місце де проводиться активність;

– к-сть людей: скільки людей записалось на відвідування активності туди ще входять к-сть дорослі/дітей і к-сть чоловіків/жінок.

– вік: вік відвідувачів, або якщо група то середнє значення;

– ціна: ціна за активність розрахована на відповідну к-сть людей;

– відповідальна особа: вибирається серед працівників;

– додаткові дані: сюди відноситься номер сертифікату, якщо відвідувач його має, і примітка.

Для клієнта:

– назва організації: тут може бути як школа так і інші заклади(не буде якщо це персональне відвідування);

– контактна особа: представник організації який буде комунікувати з музеєм;

– телефон: телефон контактної особи.

Ці параметри є основними інформаційними характеристиками, які забезпечують ведення обліку, організацію діяльності та ефективне адміністрування сервісів музею через веб-сайт.

1.2 Постановка завдання на проєкт

Оскільки поставлене завдання є вузькоспеціалізованим і передбачає вирішення конкретних, чітко окреслених задач, пошук аналогічних сайтів-конкурентів не має значної доцільності. Після обговорення деталей із замовником веб-додатку та ознайомлення з орієнтовним макетом (рис. 1.1), я розпочав розробку сайту.

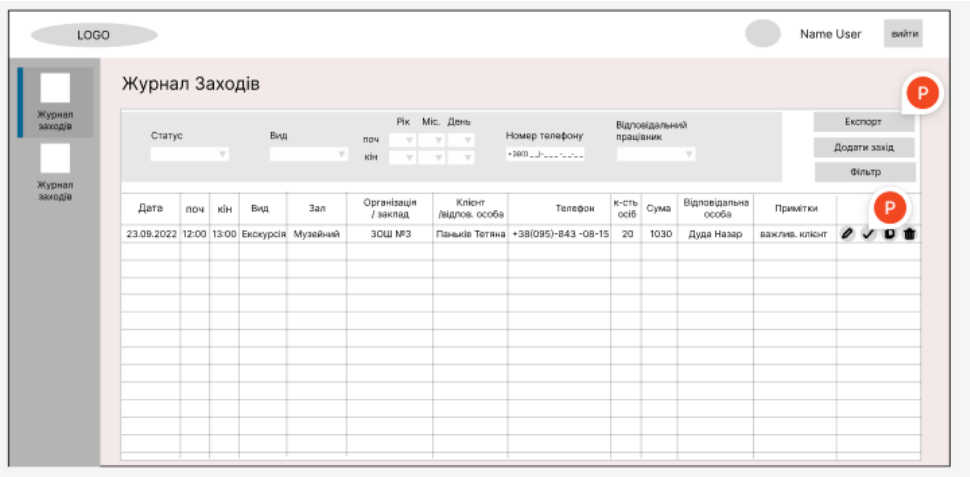


Рисунок 1.1 – Першочерговий макет сторінки “Журнал заходів”

Отже для розробки даного сайті потрібно:

– опираючись на макет завдання, розробити схожий дизайн майбутнього сайту;

– описати логічну модель бази даних, визначити структуру таблиць та взаємозв’язки між ними;

– розробити структуру бази даних та її основні елементи.

Задачі, які повинні вирішуватись:

– реалізувати можливість реєструватися і робити вхід в обліковий запис для користувачів;

– для користувачів із роллю “Адміністратор” можливість міняти ролі у інших користувачів, також додавати/віднімати нові зали проведення заходів і типи заходів;

– для користувачів із роллю “Адміністратор” можливість створювати нові записи у журналі заходів, редагувати, видаляти, копіювати та підтверджувати їх відповідно а також експортувати дані із сторінки “Журнал заходів” у Excel;

– для користувачів із роллю “Персонал” та “Адміністратор” можливість переглядати журнал заходів і здійснювати фільтрацію даних.

Сайт налічує такі основні елементи та сторінки:

1. шапка сайту:

– лого музею;

– сторінки ("Журнал заходів", "Користувачі", "Додати(зал/типу заходу)");

– ім’я та прізвище користувача;

– кнопка Вийти/Увійти в залежності чи користувач зайшов у профіль.

2. Сторінка “Журнал заходів”, містить:

– сектор фільтрації даних, де можна відфільтрувати по даті, відповідальній людині, номеру телефону та інші. записи;

– кнопки щоб додати запис і експортувати записи у excel таблицю;

– всі дані про заходи і кнопки взаємодії із кожним.

3. Сторінка “Користувачі”, містить:

– перелік користувачів;

– роль або ролі відповідних користувачів;

– можливість додавати/видаляти ролі для відповідних користувачів.

4. Сторінки “Зали”, “Спосіб оплати”, “Послуги” є схожими:

– перелік всіх залів, способів оплати та послуг відповідно до кожної сторінки;

– кнопки для додавання і видалення записів.

5. Сторінка “Профіль”:

– всі дані користувача, які він вводив при реєстрації та список ролей.

6. Сторінка входу в акаунт :

– поле логіну користувача, якщо ж користувача із даним логіном немає у базі даних, то він отримає помилку, що такого користувача не існує;

– поле пароля, хеш пароля має відповідати хешу пароля, який користувач вводив під час реєстрації, інакше буде помилка;

– посилання на реєстрацію, якщо користувач немає ще акаунту.

7. Сторінка реєстрації :

– поле логіну, який повинен містити принаймні 4 символи;

– поле паролю, пароль має містити хоча б 5 символів, одну велику літера та одну цифру;

– поле підтвердження пароля, дані у цьому полі повинні співпадати із даними з попереднього поля;

– поля Ім’я та Прізвища користувача;

– поле телефону, тут повинен бути номер телефону у певному форматі +38… , інакше буде помилка;

– поле пошти, тут також потрібен даний формат із @ та “.”;

– поле дати народження, де за допомогою календаря користувач обирає дату.

2 РОЗРОБКА СТРУКТУРНИХ РІШЕНЬ ВЕБ-САЙТУ ДЛЯ АДМІНІСТРУВАННЯ ТА ОБЛІКУ ДАНИХ

2.1 Розробка структури та функціоналу сайту “Нова Енергія”

У рамках розробки сайту для адміністрування та обліку даних наукового містечка "Нова Енергія" була обрана павутинна структура навігації. Її особливістю є те, що користувач має змогу в будь-який момент перейти з однієї сторінки на іншу без необхідності повертатися на головну сторінку.

Усі основні розділи сайту розташовані у шапці профілю та постійно доступні, що забезпечує зручну, швидку та гнучку навігацію для працівників і адміністратора. Такий підхід дозволяє мінімізувати кількість кроків для доступу до потрібної інформації та підвищує ефективність роботи з системою.

Важливо зазначити, що доступ до більшості функцій сайту можливий тільки для авторизованих працівників. Неавторизовані користувачі не мають можливості переглядати дані чи взаємодіяти з системою крім як реєструватися. У разі спроби доступу до закритих розділів, користувач отримає повідомлення що доступ закрито через недостачу прав.

Рисунок 2.1 - структура сайту наукового містечка “Нова Енергія”

Використання бази даних є необхідною умовою для реалізації такого функціоналу, адже саме вона зберігає ключову інформацію — облікові дані працівників, клієнтів, відомості про послуги, замовлення та інші важливі елементи, що забезпечують ефективне адміністрування системи.

2.2 Розробка таблиць та структури бази даних для сайту

Для реалізації автентифікації/авторизації потрібно створити таблиці Users та таблиці Roles, так як один користувач може мати декілька ролей і одна роль може бути у декількох користувачів то відповідно зв'язок між таблицями Users і таблицею Roles є багато до багатьох а отже потрібно створити додаткову таблицю UserRoles.

Таблиця 2.1 – Дані про користувачів

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор користувача	Первинний ключ
Login	Nvarchar(30)	Логін користувача	-
Password	Nvarchar(30)	Пароль користувача	-
FirstName	Nvarchar(30)	Ім’я користувача	-
LastName	Nvarchar(40)	Прізвище користувача	-
PhoneNumber	Nvarchar(20)	Номер користувача	-
EmailString	Nvarchar(max)	Пошта користувача	-
DataOfBirthday	Date	Дата народження користувача	-

Оскільки проєкт виконується за допомогою фреймворку EntityFramework, то описувати запити на створення таблиці не потрібно, все що потрібно це створити модель і на кожне поле використати відповідний атрибут як у коді нижче:

```
namespace MyApp.DB.Entities
{
    public class User
    {
        [Key]
        public int Id { get; set; }
        [MaxLength(30)]
        public string Login { get; set; }
        [MaxLength(30)]
        public string Password { get; set; }
        [MaxLength(30)]
        public string FirstName { get; set; }
        [MaxLength(40)]
        public string LastName { get; set; }
        [MaxLength(20)]
        public string PhoneNumber { get; set; }
        public string EmailString { get; set; }
        [Column(TypeName = "date")]
        public DateTime DateOfBirthday { get; set; }
        public ICollection<UserRole> Roles { get; set; }
        [NotMapped]
        public string FullName => $"{FirstName} {LastName}";
    }
}
```

Оскільки sql запит не потрібно прописуватися бо він виконується “під капотом” у подальшому я не буду його вказувати для кожної таблиці як і модель, але вигляд sql запиту для даної таблиці буде виглядати ось так:

```
CREATE TABLE [Users] (
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
```

```
    [Login] NVARCHAR(30) NOT NULL,
    [Password] NVARCHAR(30) NOT NULL,
    [FirstName] NVARCHAR(30) NOT NULL,
    [LastName] NVARCHAR(40) NOT NULL,
    [PhoneNumber] NVARCHAR(20) NULL,
    [EmailString] NVARCHAR(MAX) NULL,
    [DateOfBirthday] DATE NOT NULL
);
```

Наступну таблицю, яку слід створити це таблиця з ролями.

Таблиця 2.2 – Дані про ролі

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор ролі	Первинний ключ
Name	Nvarchar(40)	Назва ролі	-

Таблиця є максимально простою і містить тільки поле Id, що є первинним ключем, та поле name, що є власне назвою ролі.

Далі щоб з’єднати ті дві таблиці і утворити зв'язок багато до багатьох створив наступну таблицю.

Таблиця 2.3 – Таблиця UserRole

Назва поля	Тип поля	Опис	Ключ
UserId	Int	Ідентифікатор користувача	Первинний і зовнішній ключ
RoleId	Int	Ідентифікатор ролі	Первинний і зовнішній ключ

Можна побачити що ці два поля є водночас первинним і зовнішнім ключем, бо обидва поля разом унікально ідентифікують запис у таблиці та кожне поле посилається на відповідну таблицю (Users і Roles).

Далі потрібно створити три простенькі таблиці, таблиця із залами, де є тільки ідентифікатор який виступає первинним ключем та назва залу. Таблиці способи оплати і типи заходів є аналогічними до таблиці зали.

Таблиця 2.4 – Дані про зали

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор залу	Первинний ключ
Name	Nvarchar(30)	Назва залу	-

Таблиця 2.5 – Дані про типи заходів

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор типу заходу	Первинний ключ
Name	Nvarchar(30)	Назва типу заходу	-

Таблиця 2.6 – Дані про методи оплати

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор методу оплати	Первинний ключ
Name	Nvarchar(30)	Назва методу оплати	-

Таблиця 2.7 – Дані про заходи

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор заходу	Первинний ключ
StartDateTime	DateTime	Дата і час початку	-
EndTime	Time	Час закінчення	-
TypeOfServiceId	Int	Ідентифікатор типу заходу	Зовнішній ключ
HallId	Int	Ідентифікатор залу	Зовнішній ключ
CountOfPeople	Int	К-сть людей	-
CountOfChildren	Int	К-сть дітей	-
CountOfAdults	Int	К-сть дорослих	-

Кінець таблиці 2.7

CountOfFemaleG	Int	К-сть людей жіночої статі	-
CountOfMaleG	Int	К-сть людей чоловічої статі	-
AgeGroup	Int	Вікова група	-
Sum	Int	Вартість заходу	-
ResponsiblePerson	Nvarchar(50)	Відповідальний працівник	-
FindFrom	Nvarchar(Max)	Звідки дізналися про музей	-
NumberOfCertificate	Int	Номер сертифікату	-
Note	Nvarchar(Max)	Примітка	-
IsConfirmed	Bit	Чи підтверджений захід	-
ActivityRecordId	Int	Ідентифікатор запису заходу	Зовнішній ключ

Далі приступив до реалізації основних таблиць в даній роботі, а саме: таблиця заходів, таблиця клієнтів і таблиця записів заходів. Прийняв рішення зробити зв’язки один до багатьох між таблицями клієнтів та записів заходів, бо один клієнт може мати багато записів заходів, але кожен запис заходу належить тільки одному клієнту. І також зв’язок один до багатьох між таблицями заходи і записи заходів, оскільки один запис заходу може включати багато конкретних захід, але кожен захід належить тільки одному запису. Це стандартна структура, коли, клієнт може робити багато замовлень і одне замовлення може включати багато різних активностей (послуг).

Далі показано як виглядає таблиця із клієнтами.

Таблиця 2.8 – Дані про клієнтів

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор клієнта	Первинний ключ
Organization	Nvarchar(Max)	Назва організації/закладу	-

Кінець таблиці 2.8

ContactPerson	Nvarchar(Max)	Контактна оссоба/представник	-
CategoryOfContactPerson	Nvarchar(Max)	Категорія приватної особи	-
Phone	Nvarchar(20)	Телефон контактної особи	-

Щоб з’вязок правильно працював потрібно ще створити одну таблицю із записами заходів. Зв’язна таблиця між заходами, клієнтами та типом оплати показано нижче.

Таблиця 2.9 – Дані про записи заходів

Назва поля	Тип поля	Опис	Ключ
Id	Int	Ідентифікатор запису	Первинний ключ
PaymentMethodId	Int	Ідентифікатор типу оплати	Зовнішній ключ
ClientId	Int	Ідентифікатор клієнта	Зовнішній ключ

На основі розроблених таблиць була сформована реляційна база даних, яка являє собою набір взаємопов’язаних таблиць. Структуру цієї бази представлено на рисунку 2.2.

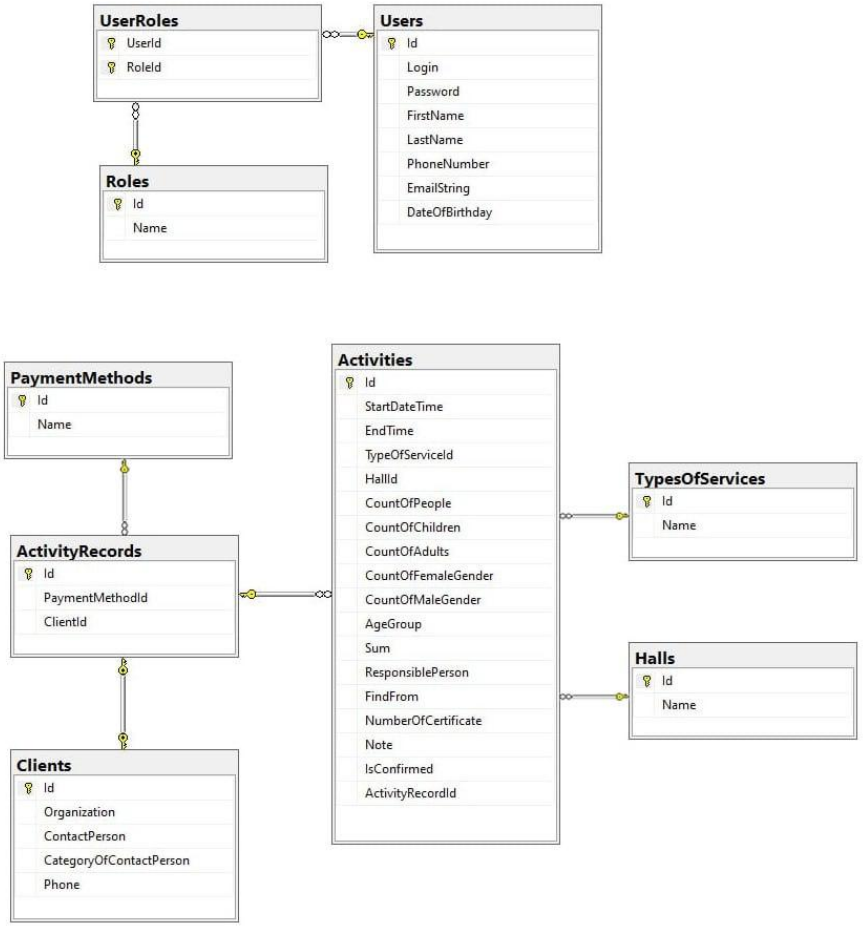


Рисунок 2.2 - Структура бази даних для сайту “Нова Енергія”

2.3 Розробка діаграм послідовності взаємодії компонентів сервісу

У процесі створення веб-сайту для адміністрування та обліку послуг музею важливо проаналізувати, як користувач взаємодіє із системою на різних етапах її використання. У цьому проєкті розглядаються декілька основних сценаріїв дій користувача і системи.

Взаємодія системи і користувача під час реєстрації показано на рисунку 2.3

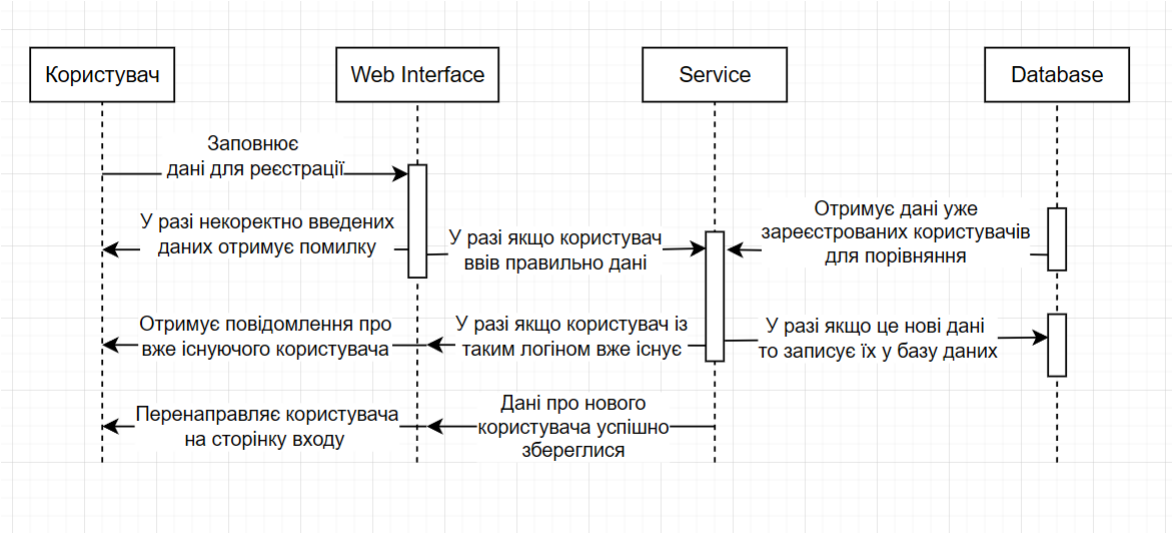


Рисунок 2.3 – Діаграма послідовності реєстрації користувача

Отже коли користувач уперше заходить на сайт, то він бачить тільки кнопку “Увійти”, при натисканні на цю кнопку він перенаправляється на форму для входу, але якщо користувач немає облікового запису, він буде бачити знизу кнопку “Зареєструватися”, після натискання він потрапляє на форму реєстрації. У діаграмі показано, що спочатку він вводить дані, якщо дані не валідний і web частина це бачить вона повідомляє його, що трапилася помилка при заповненні даних. У разі якщо все добре дані відправляються до сервісу, який безпосередньо працює з даними користувачів, він робить запит до бази даних, щоб отримати дані про логіни усіх користувачів, щоб порівняти чи дані від користувача не зустрічаються у базі даних. Якщо логін користувача вже є у базі даних, то сервіс повідомить про це web частину сайту і вона відобразить цю помилку. Якщо дані є новим то сервіс зробить запит у базу даних, щоб записати нові вхідні дані, після чого повідомить web частину, що все пройшло успішно і дані збережено, користувач у свою чергу буде вже автоматично перенаправлено на сторінку Входу.

Далі розглянемо діаграму входу, яка є дещо схожою із попередньою.

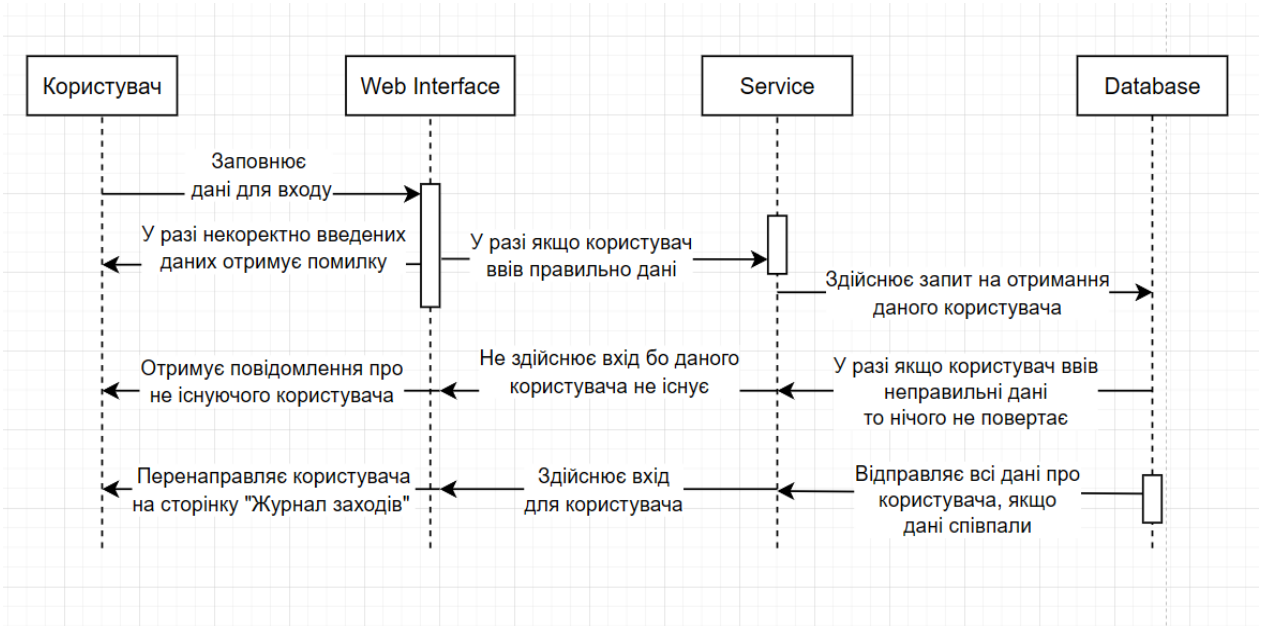


Рисунок 2.4 – Діаграма послідовності входу користувача

На діаграмі можна побачити, що вона дійсно дуже схожа до попередньої, тільки тут, ще по даним які ввів користувач робить запит, якщо відповіді немає тоді можна зрозуміти, що якісь дані введені користувачем не є правильними і слід повідомити користувача, що такого користувача не існує. У разі якщо даний користувач є у базі даних то здійсниться вхід і користувач буде перенаправлено на сторінку із журналом заходів.

Наступним, що потрібно розглянути це сценарній отримання відфільтрованих даних користувачем, тобто процес фільтрування через спеціальне меню (рис. 2.5). Спочатку користувач у спеціальній формі вибирає по яким параметрах йому потрібно відобразити дані, далі сервіс на основі цього формує запит і після отримання даних з бази даних відправляє на web частину, щоб вона вже відобразила потрібні записи.

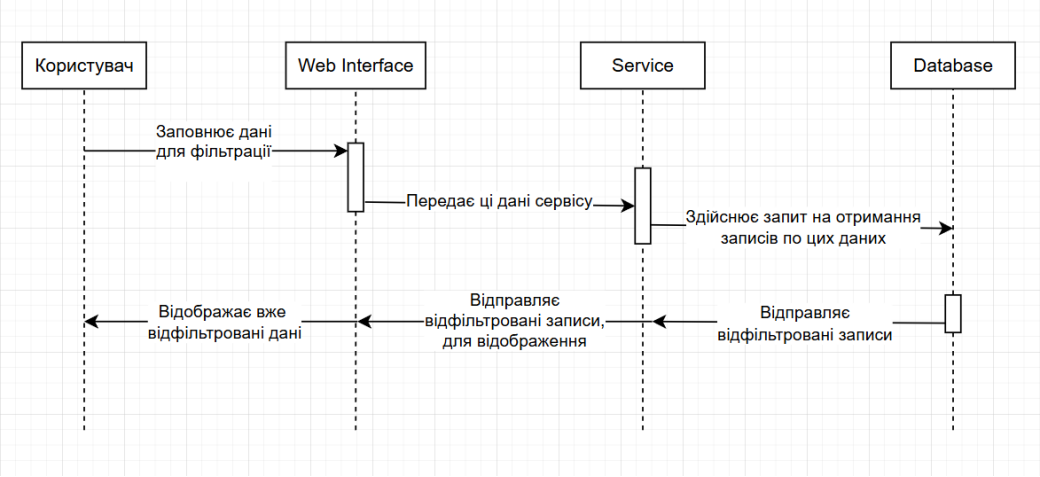


Рисунок 2.5 – Діаграма послідовності фільтрації даних

Взаємодії описані вище доступні користувачам із ролями “Administrator” та “Personnel”. Наступні ж будуть вже доступні тільки адміністратору. На сторінці “Журнал заходів” можна загружати Excel таблицю із відфільтрованими записи, які цікавлять користувача. Процес схожий до фільтрації тільки сервіс автоматично записує дані у створену таблицю і завантажує її на комп’ютер користувача. Діаграму послідовності експорту даних у вигляді Excel таблиці показано на рисунку 2.6.

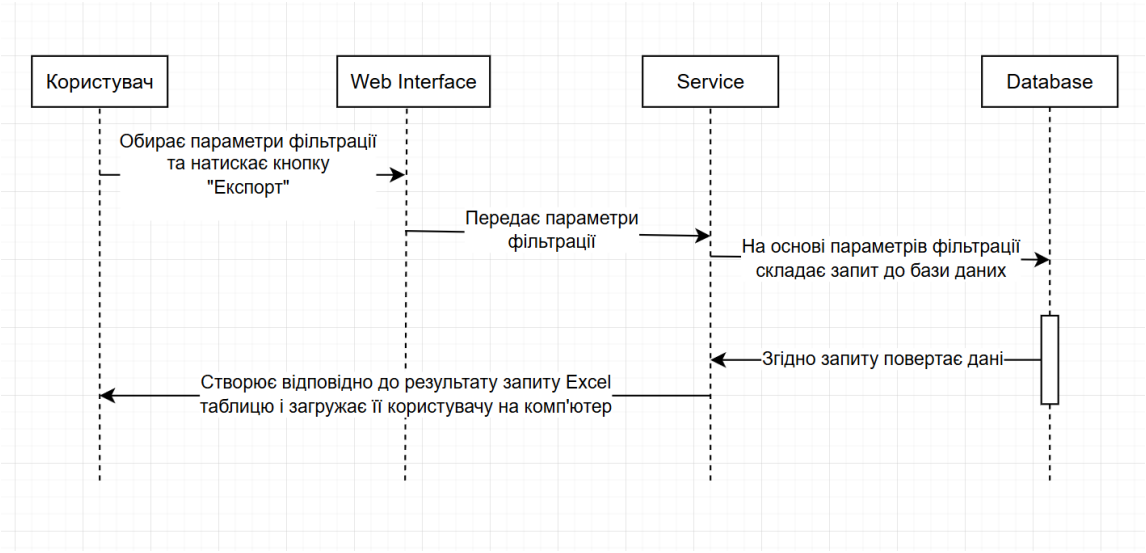


Рисунок 2.6 – Діаграма послідовності експорту даних

Після того як адміністратор перейду у вкладку “Користувачі” він побачить список користувачів, їх ролі та можливість видаляти та додавати ролі для певних

користувачів. Щоб додати роль потрібно навпроти користувача натиснути на посилання із надписом “Додати роль”, далі сайт повинен показати перелік усіх можливих ролей, і користувач повинен вибрати, далі ця інформація потрапляє у сервіс який вже додає записи у базу даних (рис. 2.7).

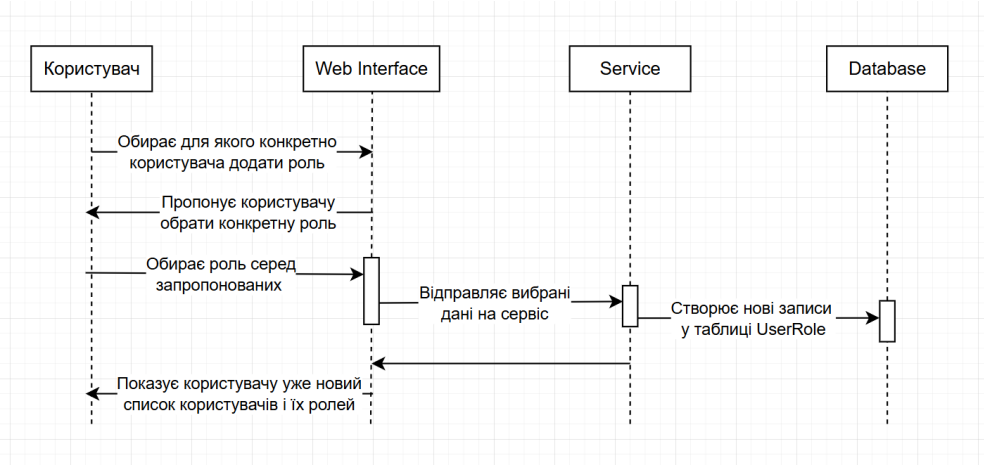


Рисунок 2.7 – Діаграма послідовності додавання ролі для користувача

Також адміністратор може прибрати роль для певного користувача, процес дуже схожий тільки із запропонованих ролей для видалення буду відображатися тільки ролі користувача, а не усі можливі (рис. 2.8).

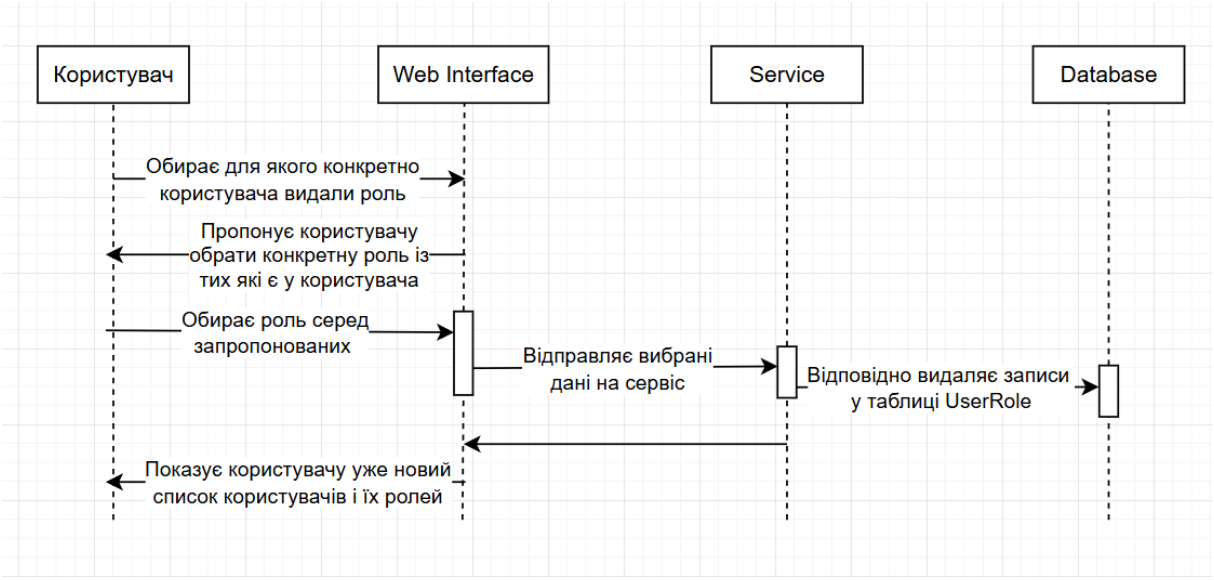


Рисунок 2.8 – Діаграма послідовності видалення ролі для користувача

Отже, розробка зазначених діаграм послідовностей, що ілюструють взаємодію користувача із системою, є необхідною умовою для забезпечення повноцінного функціонування веб-сайту.

2.4 Розробка USE-CASE діаграми взаємодії з користувачами

Виходячи зі структури веб-сайту для адміністрування та обліку послуг музею, побудованої бази даних та діаграм послідовностей взаємодії з користувачем, було розроблено діаграму USE-CASE, яка відображає можливі сценарії взаємодії з функціоналом веб-сайту, який доступний для кожного з існуючих акторів. У таблиці нижче показано дії, які може виконувати користувач залежно від його ролі.

Таблиця 2.10 - Опис варіантів використання

Назва	Опис
Реєстрація	Функція реєстрації акаунта
Вхід(Авторизація Автентифікація)	Функція що розрізняє користувача і надає йому права
Додавання залів/типів заходів/методів оплати	Функція доступна адміністратору для додавання нових залів/типів заходів та методів оплати
Редагування ролей для інших користувачів	Функція доступна адміністратору для додавання нових і видалення старих ролей користувачу
Додавання нових заходів у журнал	Функція доступна адміністратору для додавання нових заходів у журнал заходів.
Редагування/Видалення/ Копіювання/Підтвердження запису	Функція доступна адміністратору для редагування, видалення, копіювання, підтвердження запису у журналі
Експорт у Excel	Функція доступна адміністратору щоб отримати Excel таблицю із певних записів згідно параметрів фільтрації

Так як сайт не надає ніяких можливостей взаємодії неавторизованому

користувачі, крім реєстрації, то нижче показано use-case діаграму для користувача із роллю “Персонал” (рис. 2.9), даний користувач може користуватися тільки фільтрацією і переглядати журнал заходів. Користувач із роллю

Рисунок 2.9 – Use-case діаграми функцій користувач із роллю “Персонал”

Користувач із роллю “Персонал”:

– перегляд записів у журналі заходів;
– фільтрація записів у таблиці журнал заходів.

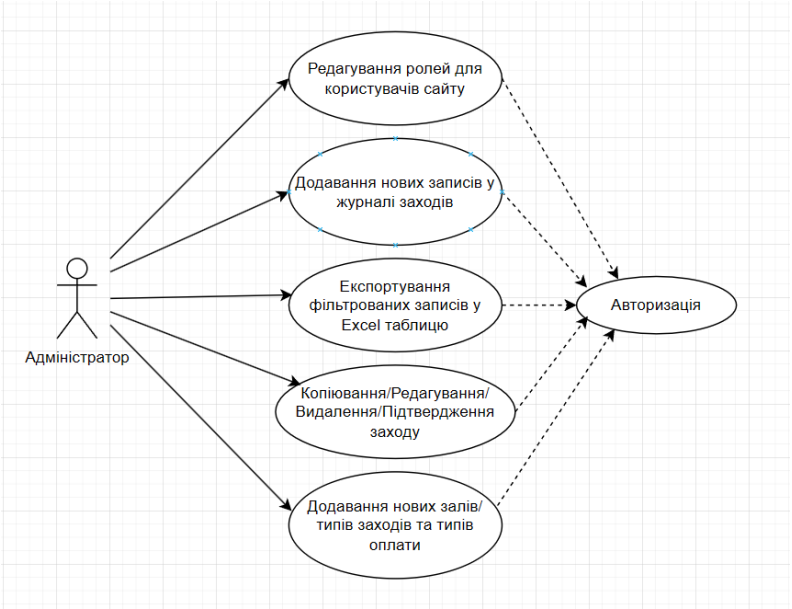


Рисунок 2.10 - Use-case діаграми функцій користувач із роллю “Адміністратор”

Адміністратор:

1. Перегляд записів у журналі заходів;
2. Фільтрація записів у таблиці журнал заходів;
3. Редагування ролей для користувачів;
4. Додавання нових записів у журнал заходів;
5. Експортування фільтрованих записів у Excel таблицю;
6. Копіювання/Редагування/Видалення/Підтвердження заходів;
7. Додавання нових залів, типів заходів та типів оплат.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБ-САЙТУ ДЛЯ АДМІНІСТРУВАННЯ ТА ОБЛІКУ ДАНИХ

3.1 Розробка та реалізація веб-сайту наукового містечка “Нова Енергія”

У межах розробки інформаційної системи для наукового містечка "Нова Енергія" було створено веб-сайт, основною метою якого є забезпечення зручного та ефективного адміністрування послуг і ведення обліку діяльності музею. Веб-сайт розроблено з урахуванням особливостей внутрішніх бізнес-процесів, де основними користувачами системи є адміністратори та працівники наукового містечка. Для звичайних відвідувачів система не передбачає можливості взаємодії, що дозволяє зосередити весь функціонал на внутрішньому управлінні.

Для реалізації проєкту була застосована багаторівнева (n-layer) архітектура, яка забезпечує розподіл відповідальностей між окремими компонентами. Архітектура проєкту включає такі складові: App.Db (рівень доступу до бази даних), App.Service (рівень бізнес-логіки) та App.Web (рівень представлення) (рис. 3.1). Такий підхід дозволяє створити гнучку, зрозумілу та легко підтримувану систему, в якій кожен рівень виконує окрему функцію та мінімально залежить від інших.

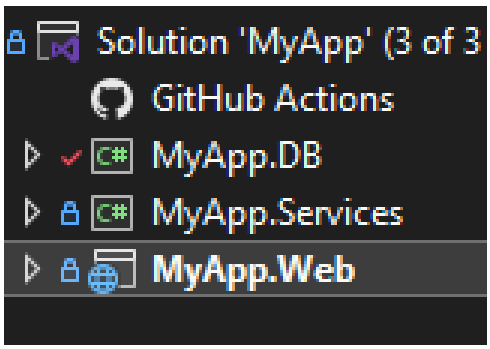


Рисунок 3.1 – Загальна структура проєкту

Проєкт App.Db виконує функцію доступу до бази даних і є основним

шаром зберігання даних у розробленій системі (рис. 3.2). У його структурі ключове місце займає папка Entities, яка містить усі необхідні моделі (сутності), що описують таблиці бази даних та їх властивості. Ці моделі є основою для побудови зв’язків між різними об'єктами системи.

У папці Migrations зберігаються файли, які відповідають за створення, оновлення та зміну структури бази даних у процесі розробки. Завдяки використанню механізму міграцій, база даних може автоматично синхронізуватися зі змінами в моделях без необхідності ручного втручання.

Клас AppDbContext є центральною частиною цього рівня, оскільки саме він забезпечує зв'язок між сутностями та фізичною базою даних, а також дозволяє виконувати запити, збереження та обробку інформації.

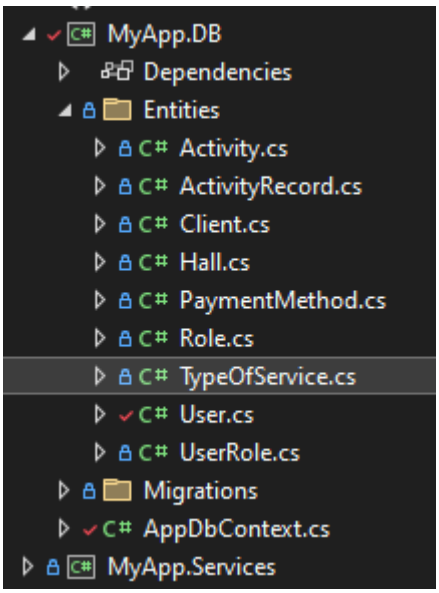


Рисунок 3.2 – Структура проєкту App.Db

У проєкті MyApp.Service реалізовано бізнес-логіку веб-сайту, яка відповідає за обробку даних та взаємодію між базою даних і веб-інтерфейсом (рис. 3.3). Цей рівень забезпечує чітке розділення відповідальностей та сприяє зручній підтримці та масштабуванню проєкту.

У папці Interfaces зосереджені інтерфейси, які визначають контракти для сервісів. Вони дозволяють описати, які методи повинні бути реалізовані у відповідних класах, що забезпечує зручність для подальшого розширення

функціоналу та полегшує тестування. Завдяки використанню інтерфейсів у проєкті реалізовано принцип Dependency Injection — сервіси впроваджуються через конструктори класів, що дозволяє легко замінювати їх реалізації, підвищує модульність коду та сприяє кращій підтримці й тестуванню системи.

Папка Models містить допоміжні моделі, які використовуються для обміну інформацією між різними рівнями програми або для обробки даних, які не зберігаються безпосередньо в базі.

У папці Services реалізовані конкретні сервіси, які містять бізнес-логіку програми, методи обробки запитів, взаємодію з базою даних через репозиторії або контекст, а також виконують основні функції системи відповідно до вимог замовника.

Завдяки чіткій структурі цього рівня, система стає зрозумілою, логічно побудованою та зручною для подальшої підтримки й розвитку.

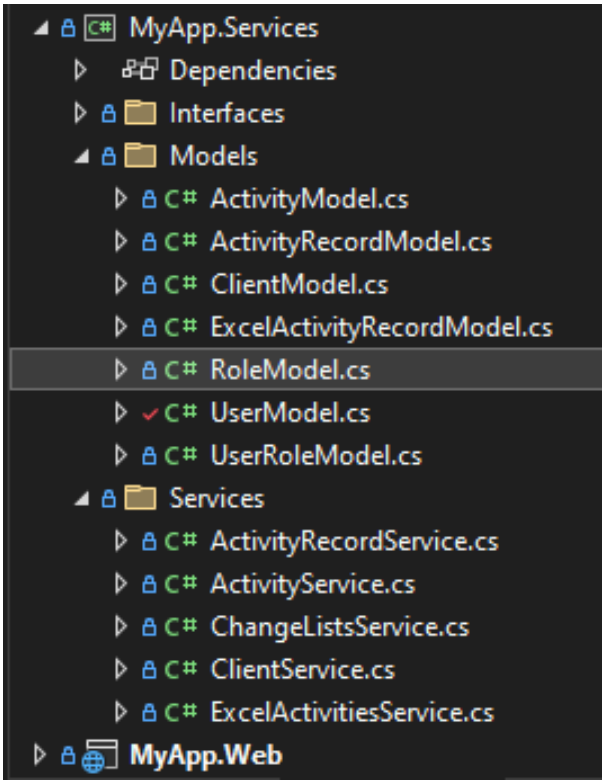


Рисунок 3.3 – Структура проєкту App.Services

Проєкт App.Web є основним рівнем представлення, через який відбувається взаємодія користувачів із системою(рис. 3.4). У його структурі

ключове місце займають папки Controllers, які відповідають за обробку HTTP-запитів, керують маршрутизацією та координують взаємодію між інтерфейсом користувача і бізнес-логікою, реалізованою у сервісах. Контролери забезпечують прийом даних від користувача, їх перевірку, виклик відповідних сервісів для обробки та формування відповідей.

Папка Infrastructure містить допоміжні класи та сервіси, що підтримують основну функціональність веб-сайту. Тут зокрема розміщені механізми для безпечного хешування паролів, що гарантує захист користувацьких даних, а також компоненти для маппінгу моделей — наприклад, реалізація маппера для трансформації даних між різними об’єктами, що значно спрощує процес реєстрації та автентифікації користувачів.

У папці ViewModels зберігаються моделі представлення, які адаптовані для передачі даних від контролерів до виглядів (Views). Вони містять лише ті властивості, які потрібні для відображення та взаємодії у користувацькому інтерфейсі, що сприяє підтримці принципу розділення відповідальностей і підвищує зручність розробки.

Папка Views містить файли з шаблонами інтерфейсу користувача, реалізовані з використанням Razor — технології, яка дозволяє поєднувати HTML з C# кодом для динамічного генерування сторінок.

Крім цього, проєкт включає такі важливі конфігураційні файли, як Program.cs, який відповідає за запуск і ініціалізацію веб-додатку, а також файл Startup.cs, де налаштовуються сервіси, підключаються middleware-компоненти, конфігурується маршрутизація та безпека додатку. Файл SampleData використовується для наповнення бази даних тестовими або початковими даними, що необхідно для відлагодження та демонстрації роботи системи.

Завдяки такій структурі проєкт MyApp.Web є зручним для розширення, підтримки та забезпечує високу якість взаємодії користувачів із системою.

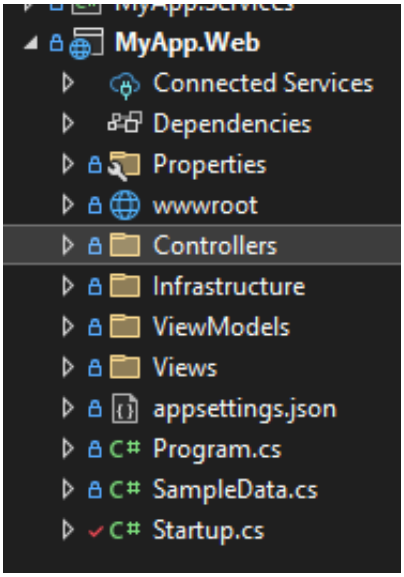


Рисунок 3.4 – Структура проєкту App.Wb

У кожному рівні проєкту використовуються власні моделі, які оптимізовані для виконання специфічних завдань на відповідному шарі. Наприклад, у рівні доступу до даних (MyApp.Db) це сутності (Entities), що відображають структуру таблиць бази даних, у бізнес-логіці (MyApp.Service) — моделі, які відповідають за обробку даних, а у веб-рівні (MyApp.Web) — ViewModels, що призначені для відображення інформації користувачам.

Для спрощення трансформації даних між цими різними моделями в проєкті застосовується бібліотека AutoMapper (рис. 3.5). Вона автоматизує процес копіювання властивостей з однієї моделі в іншу, що значно зменшує кількість повторюваного коду і знижує ризик помилок при ручній конвертації. AutoMapper налаштовується через спеціальні профілі, де визначаються правила відповідності полів між моделями різних шарів. Це дозволяє підтримувати чисту архітектуру і забезпечує прозору передачу даних між рівнями системи.

Використання AutoMapper покращує модульність і масштабованість коду, оскільки зміни у структурі однієї моделі автоматично відображаються в процесі мапінгу без необхідності вносити суттєві зміни у інші частини програми. Таким чином, AutoMapper є важливим інструментом у реалізації принципу розділення відповідальностей та підтримці чистоти архітектури веб-сайту "Нова Енергія".

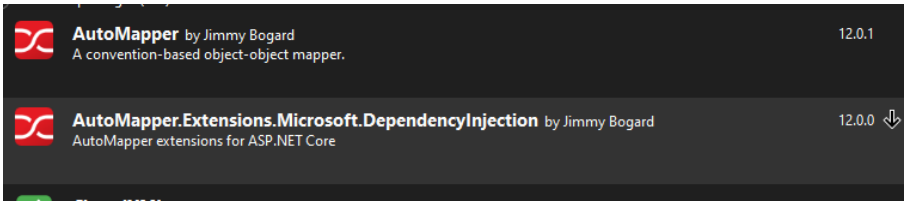


Рисунок 3.5 – Бібліотеки AutoMapper

Також для роботи з базою даних було встановлено EntityFrameworkCore, який забезпечую зручну роботу з базою даних.

Щоб реалізувати авторизацію і автентифікацію спочатку потрібно створити таблиці користувачів і ролей та допоміжну таблицю, як буде виступати зв'язною для реалізації зв'язку багато до багатьох.

Коди до Entity User показано нижче, всі інші також реалізовано згідно таблиць у розділі 2.2.

```
public class User
{
    public int Id { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string PhoneNumber { get; set; }
    public string EmailString { get; set; }
    public DateTime DateOfBirthday { get; set; }
    public ICollection<UserRole> Roles {  get; set; }
}
```

Розуміючи що інші шари програми будуть працювати зі своїми моделями потрібно у файлі MapperRegistration.cs додати дані рядки:

```
public MapperRegistration()
{
    CreateMap<User, UserModel>(MemberList.Destination);
    CreateMap<UserModel, User>(MemberList.Destination);
```

```
        CreateMap<UserModel,UserViewModel>(MemberList.Destination);
        CreateMap<UserViewModel, UserModel>(MemberList.Destination);

        CreateMap<UserRole,UserRoleModel>(MemberList.Destination);
        CreateMap<UserRoleModel,UserRole>(MemberList.Destination);

        CreateMap<Role, RoleModel>(MemberList.Destination);
        CreateMap<RoleModel, Role>(MemberList.Destination);

        CreateMap<RoleModel, RoleViewModel>(MemberList.Destination);
        CreateMap<RoleViewModel, RoleModel>(MemberList.Destination);
}
```

Далі приступив до написання Інтерфейсу для сервісу взаємодії з користувачем і ролями IIdentityService.cs:

```
public interface IIdentityService
{
        IEnumerable<UserModel> Users { get; }
        IEnumerable<RoleModel> Roles { get; }
        Task AddRoleToUser(int userId, int roleId);
        Task RemoveRoleFromUser(int userId, int roleId);
        Task<UserModel> GetUserByLogin(string login);
        Task<UserModel> GetUserById(int? id);
        Task<UserModel> AddUser(UserModel userModel);
}
```

Клас IdentityService реалізовує інтерфейс IidentityService:

```
public class IdentityService : IIdentityService
```

```
{
     private readonly AppDbContext _appDbContext;
     private readonly IMapper _mapper;

     public IdentityService(AppDbContext appDbContext, IMapper
     mapper)
     {
          _appDbContext = appDbContext;
         _mapper = mapper;
     }
}
```

Прописується конструктор даного класу, і після нього реалізація всіх методів інтерфейсу такі як додати користувача(AddUser), методи для додавання та видалення ролі для користувача(AddRoleToUser/RemoveRoleFromUser), отримання користувача по ід/логіну(GetUserById/GetUserByLogin) і також реалізація отримання ролей і користувачів із Бази Даних.

У проєкті використовується механізм впровадження залежностей (Dependency Injection) для зручного управління життєвим циклом об'єктів та для забезпечення слабкого зв'язку між компонентами. Для реєстрації сервісів у системі використовується файл Startup.cs, який містить метод ConfigureServices. Саме в цьому методі відбувається додавання залежностей до контейнера сервісів. Наприклад, для підключення сервісу авторизації потрібно зареєструвати інтерфейс та його реалізацію наступним чином:

```
services.AddScoped<IIdentityService, IdentityService>();
```

Також для налаштування механізму автентифікації та авторизації у веб-додатку необхідно додати відповідні сервіси у методі ConfigureServices класу Startup. Зокрема, потрібно підключити cookie-автентифікацію, яка дозволяє зберігати інформацію про вхід користувача у вигляді кукі-файлів. Для цього у контейнер сервісів необхідно додати наступний фрагмент коду:

```
services.AddAuthentication(CookieAuthenticationDefaults.Authentica
tionScheme).AddCookie(options =>
{
     options.LoginPath = new
     Microsoft.AspNetCore.Http.PathString("/Account/Login");
});
```

Ця конфігурація вказує, що у проєкті використовується cookie як основний спосіб автентифікації, а шлях до сторінки входу встановлюється як /Account/Login. Це означає, що у випадку спроби доступу до захищеного ресурсу без автентифікації користувача буде автоматично перенаправлено на сторінку входу. Окрім цього, для коректної роботи системи автентифікації необхідно вказати порядок обробки запитів у методі Configure класу Startup. Тут потрібно додати виклики:

```
app.UseAuthentication();
app.UseAuthorization();
```

Метод app.UseAuthentication() підключає middleware, яке перевіряє наявність кукі у запиті та, за необхідності, встановлює контекст користувача. Метод app.UseAuthorization() дозволяє контролювати доступ до окремих частин веб-додатку відповідно до налаштованих правил авторизації. Використання цих компонентів є обов'язковим для захисту ресурсу та правильної обробки прав доступу в системі. Завдяки такій конфігурації реалізується безпечний механізм входу в систему, захист сторінок та контроль над доступом відповідно до рівня прав користувачів.

Клас UserViewModel буде дещо відрізнятися від схожих йому класів:

```
public class UserViewModel
{
    public int Id { get; set; }
    [Display(Name = "Логін")]
    public string Login { get; set; }
    [Display(Name = "Ім'я")]
    [Required(ErrorMessage = "Вкажіть Ім'я")]
    public string FirstName { get; set; }
    [Display(Name = "Прізвище")]
    [Required(ErrorMessage = "Вкажіть Прізвище")]
```

```
    public string LastName { get; set; }
    [Display(Name = "Телефон")]
    [DataType(DataType.PhoneNumber)]
    [RegularExpression(@"^(\+38)\d{10}$", ErrorMessage = "Невірний
формат номера телефону(+38)")]
    [Required(ErrorMessage = "Вкажіть номер телефону")]
    public string PhoneNumber { get; set; }
    [Display(Name = "Пошта")]
    [Required(ErrorMessage = "Вкажіть пошту")]
    [DataType(DataType.EmailAddress)]
    public string EmailString { get; set; }
    [Display(Name = "Дата народження")]
    [Required(ErrorMessage = "Вкажіть дату-народження")]
    [DataType(DataType.Date)]
    public DateTime DateOfBirthday { get; set; }
    [Display(Name = "Ролі")]
    public ICollection<UserRoleViewModel> Roles {  get; set; }
}
```

Даний клас крім все тих же гетерів/сетерів містить атрибути. Атрибут в C# — це метадані, які додаються до класів, методів, властивостей, параметрів або інших елементів коду, щоб надати додаткову інформацію про них. Атрибути використовуються для: валідації даних, налаштування поведінки коду під час виконання, документування коду, взаємодії з рефлексією.

Уданому випадку використовуються:

[Display] — це атрибут анотацій даних (Data Annotation), який використовується в ASP.NET MVC і Blazor для налаштування відображення властивостей моделі у формах, представленнях (Views) та повідомленнях про помилки.

[Required] – цей атрибут вимагає, щоб поле не було порожнім (обов'язкове для заповнення). ErrorMessage = "текст" – текст помилки, який буде виведений, якщо користувач не введе значення.

Атрибут [DataType(DataType.PhoneNumber)] використовується в ASP.NET MVC для визначення типу даних та покращення їх обробки у формах.

Атрибут [DataType(DataType.EmailAddress)] використовується для вказівки, що поле є електронною поштою.

Атрибут [DataType(DataType.Date)] вказує, що поле містить дату без часу.

Це атрибут валідації [RegularExpression], який перевіряє, чи значення в полі відповідає заданому регулярному виразу.

Маючи все вище перераховане можна переходити до створення контролеру, він буде мати назву Account. Код даного класу показано нижче:

```
     public class AccountController : Controller
{
     private readonly IIdentityService _identity;
     private readonly IMapper _mapper;

     public AccountController(IIdentityService identity, IMapper
mapper)
     {
          _identity = identity;
            _mapper = mapper;
     }
…
}
```

Отже він наслідується від загального зласу Controller, має конструктор і два поля типу мапер та сервісу для даного контролера. Використовує він всього два типи запитів це Get і Post.

Метод GET використовується для отримання даних із сервера. Усі параметри передаються через URL у вигляді рядка запиту (QueryString). Це означає, що дані можна побачити в адресному рядку браузера, що робить метод менш безпечним для передачі конфіденційної інформації. GET-запити кешуються браузером, а також мають обмеження на довжину URL. Важливо, що GET є ідемпотентним, тобто повторне виконання запиту не змінює стан сервера.

Метод POST, на відміну від GET, використовується для надсилання даних на сервер, наприклад, при заповненні форми. Усі параметри передаються у тілі запиту, що робить цей метод більш безпечним для передачі конфіденційної інформації, оскільки дані не відображаються в URL. POST-запити не кешуються браузером, а їхній розмір не обмежений, що дозволяє передавати великі обсяги інформації. Проте POST не є ідемпотентним – повторне надсилання запиту може, наприклад, створити дублікат запису в базі даних.

Тут вони реалізовані за допомогою атрибутів HttpGet/HttpPost. Перший екшин який має цей контролер це Profile, він шукає по логіну, який отримує з клеймів, в далі під цього користувача повертає View.

Наступний екшин це Login типу get, який просто генерує сторінку логування. Login типу post має ряд перевірок щоб після успішного їх проходження перекинути на метод Authenticate, який буде описаний нище.

Екшин Register є схожим до попереднього оскільки виконується і get і post запитом. Але у post тут вже створюється і додається запис у бд. Метод Authenticate працює з кукі та клаймами, які є унікальними для кожного користувача.

Щоб сайт функціонував добре нам потрібно View для взаємодії з користувачем, кожен проєкт mvc містить Razor сторінку _Layout.cshtml. Razor-сторінка – це вигляд (View) в ASP.NET, який використовує Razor-синтаксис для поєднання HTML і C#-коду. Вона дозволяє створювати динамічні веб-сторінки, що взаємодіють із сервером.

Коли користувач пробує зареєструватися він потрапляє на дану форму входу (рис. 3.6).

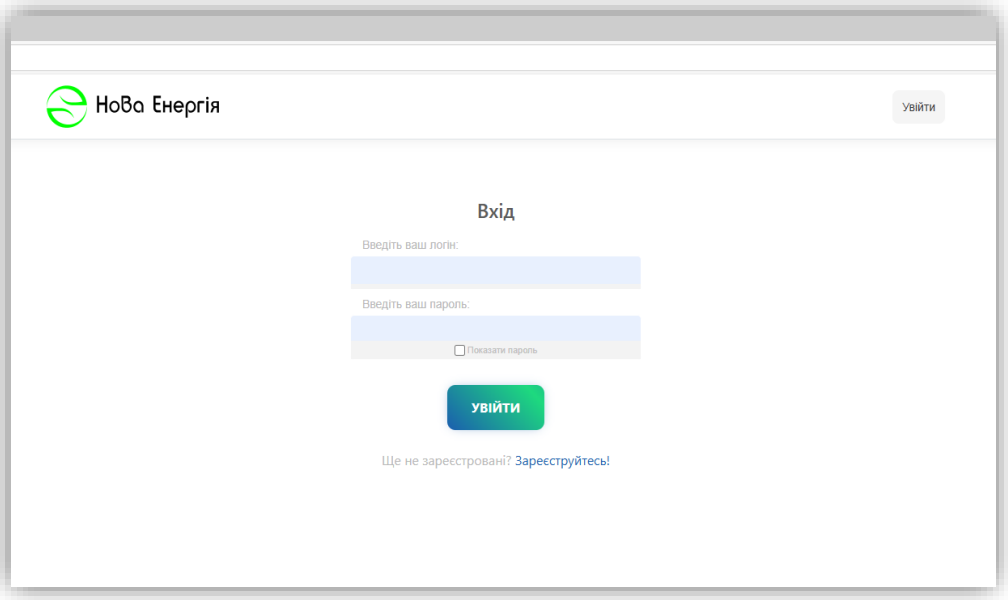


Рисунок 3.6 – Форма входу

Тут з використанням java script, який робить кнопку неактивною якщо всі поля не заповнені, та checkbox, який приховує або показує пароль для зручності, показано роботу на рисунку 3.7.

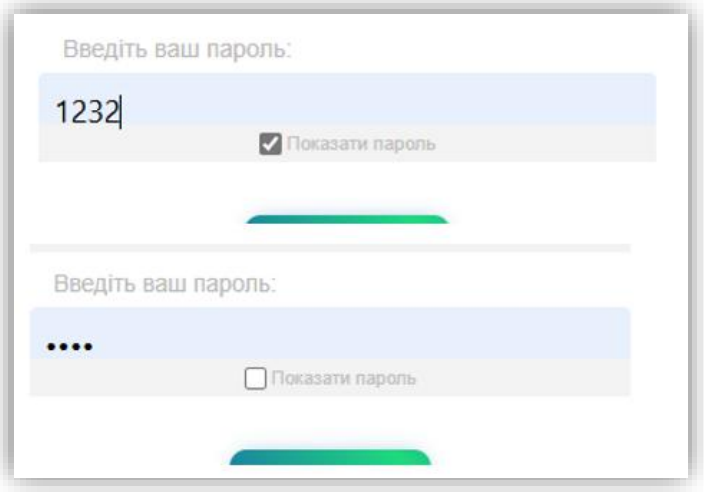


Рисунок 3.7 – Робота checkbox показати пароль

Це виконано з використанням даного js скрипту:

```
let inputLogin = document.getElementById("inputLogin");
let inputPass = document.getElementById("inputPass");
let btnSubmit = document.getElementById("btnSubmit");
function checkInputs(){
     if(inputLogin.value.trim() !== ""
          && inputPass.value.trim() !== ""){
          btnSubmit.disabled = false;
     }else{
     btnSubmit.disabled = true;
     }
}
inputLogin.addEventListener("input", checkInputs);
inputPass.addEventListener("input", checkInputs);
function ShowPassForLoginPage() {
     let el = document.getElementById('inputPass');
     if (el.type === "password") {
          el.type = "text";
     } else {
        el.type = "password";
     }
}
```

Далі спробую зайти використавши логін користувача якого не існує, результат на рисунку 3.8. Тут користувач або помилився із логіном або пробує використати логін від не існуючого користувача. Із паролем буде аналогічно.

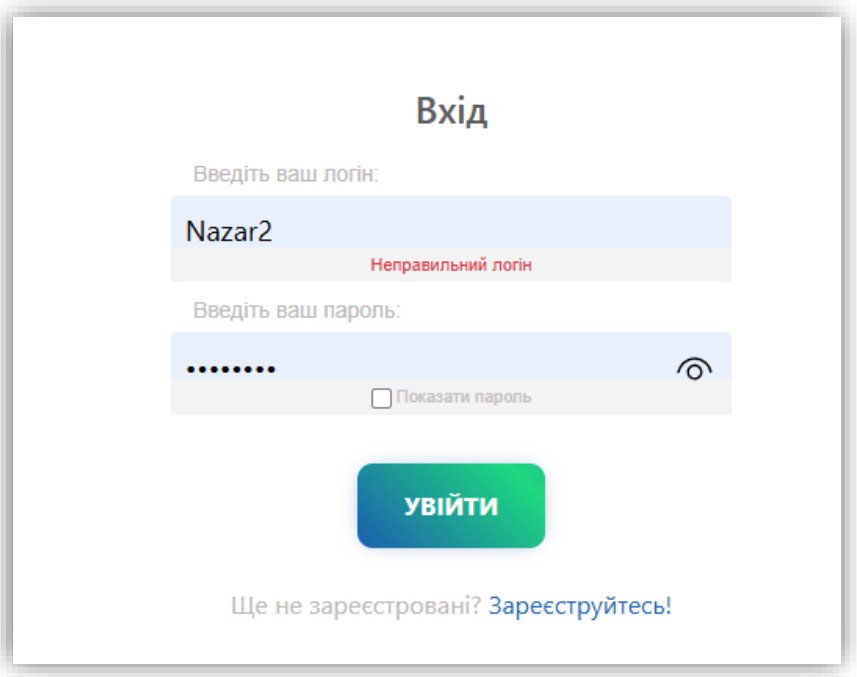


Рисунок 3.8 – Спроба входу з невірним логіном

Якщо користувач правильно введене дані свого зареєстрованого акаунта то він побачить, що кнопка “Увійти” змінилась “Вийти”, лівіше пише його прізвище та ім’я з можливістю натиснути на них і перейти у профіль, і також йому стали доступні розділи меню відносно його ролі (рис. 3.9).

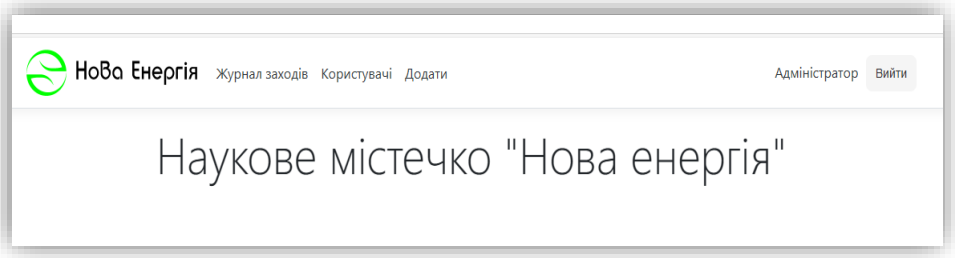


Рисунок 3.9 – Сторінка після успішного входу

У разі якщо користувач ще немає, наявного акаунта на сторінці входу я надпис “Зареєструйтесь!”, натиснувши на який, користувач попадає на сторінку

реєстрації (рис. 3.10).

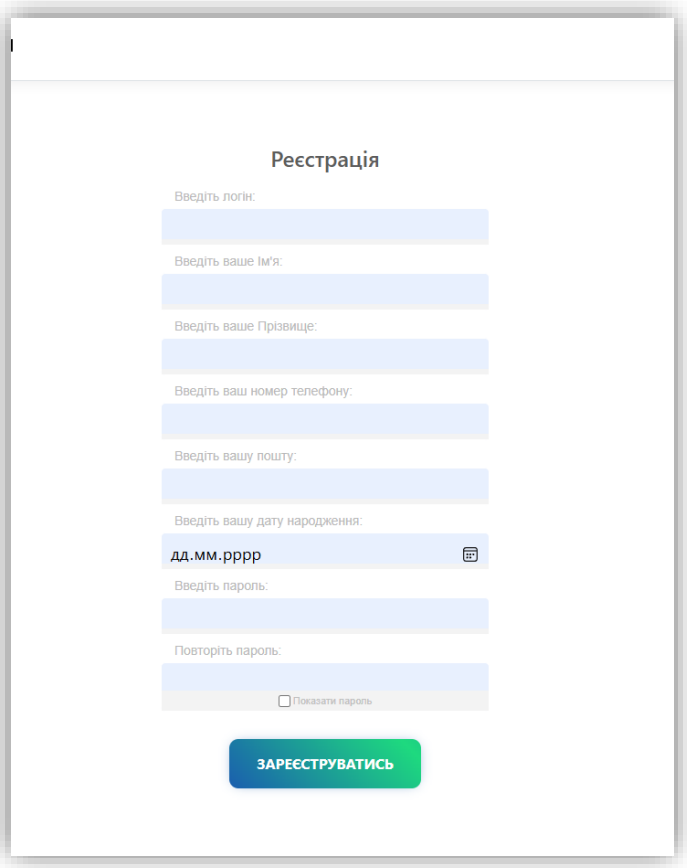


Рисунок 3.10 – Форма реєстрації

Дана форма є дещо схожою із формою входу, проте тут більше полів де потрібно вводити правильні дані, щоб програма пропустила далі користувача і зареєструвала його, також присутній js скрипт, який робить неактивною кнопку “Зареєструватись” якщо є хоча б одне не заповнене поле. Також присутній прапорець, з його допомогою можна відобразити пароль для зручності.

Після спроби ввести некоректні дані, форма повідомить користувачу, що або дані не у вірному форматі або користувач із таким логіном є у базі даних, це продемонстровано на рисунку 3.11.

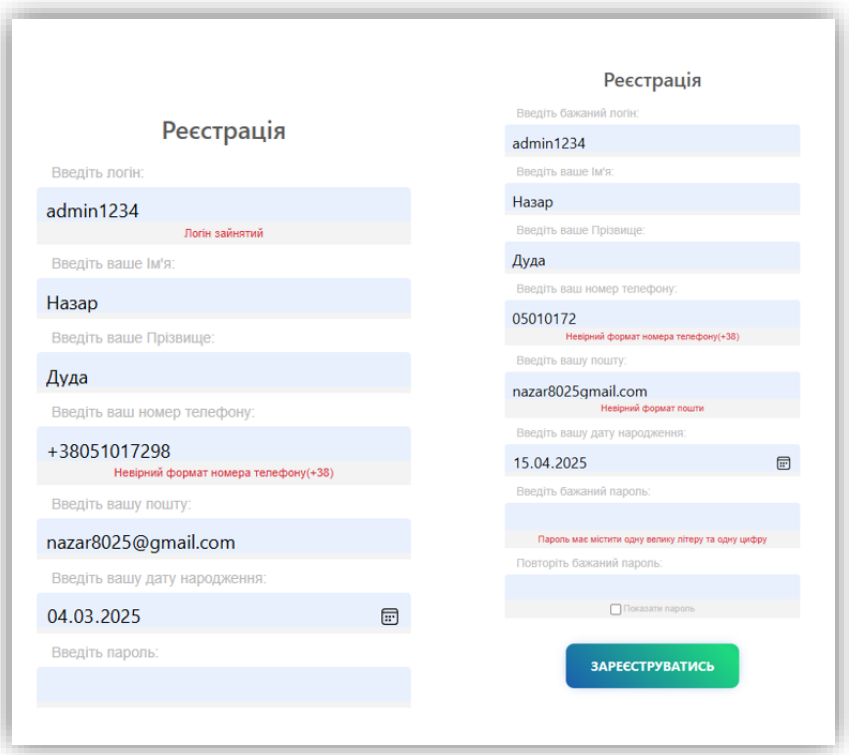


Рисунок 3.11 – Помилки під час реєстрації

У одному варіанті, користувач просто не правильно заповнив поля, а у іншому використав логін, що зайнято і відповідно система не зареєструвала його і повідомила його про це.

Після успішної реєстрації, користувача автоматично зареєструє і буде доданим у базу даних він зможе тільки переглядати свій профіль і виходити з акаунта, оскільки більше можливостей він немає, поки йому не дадуть права (рис. 3.12).

Рисунок 3.12 – Мінімалістичний профіль користувача

У разі якщо користувач через використання URL зможе зайти на сторінку доступу до якої у нього немає, то він отримає таке повідомлення (рис. 3.13). Це реалізовано з використанням атрибута [Authorize(Roles = "admin")] над контролером який дає змогу користуватися ним тільки певним користувачам.

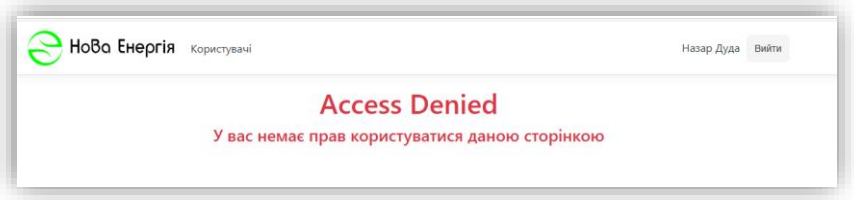


Рисунок 3.13 – Обмеження прав користувача

Отже коли розібрали усі аспекти і тонкощі роботи ролей у даному сайті переходимо до сторінки, де користувач із роллю “Адміністратор” може додавати або видаляти ролі у певних користувачів. Сторінка “Користувачі” показано на рисунку 3.14

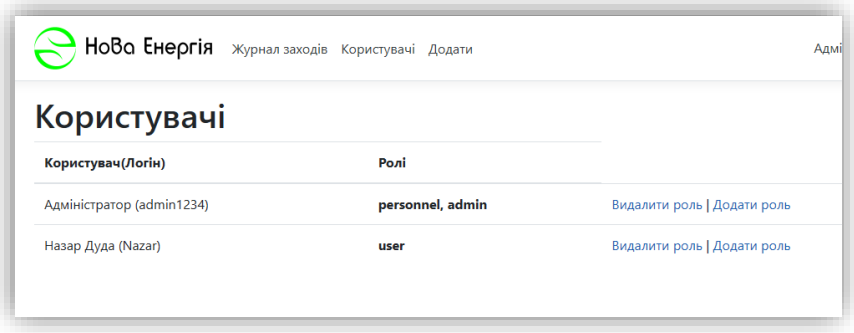


Рисунок 3.14 – Сторінка “Користувачі”

Інтуїтивно зрозумілий інтерфейс де показано ім’я та прізвище, дужках логін користувача, далі ролі через кому бо їх може бути декілька і посилання управління ролями для користувачів(додати роль або забрати).

Після натискання на кнопку “Додати роль”, адміністратору буде продемонстровано модальне вікно (рис. 3.15), де йому буде уточнено для якого конкретно користувача він додає роль і запропоновано випадаючий список, де буде перелік усіх можливих ролей. Після натискання “Додати” адміністратор побачить уже нові зміни в списку (рис. 3.16).

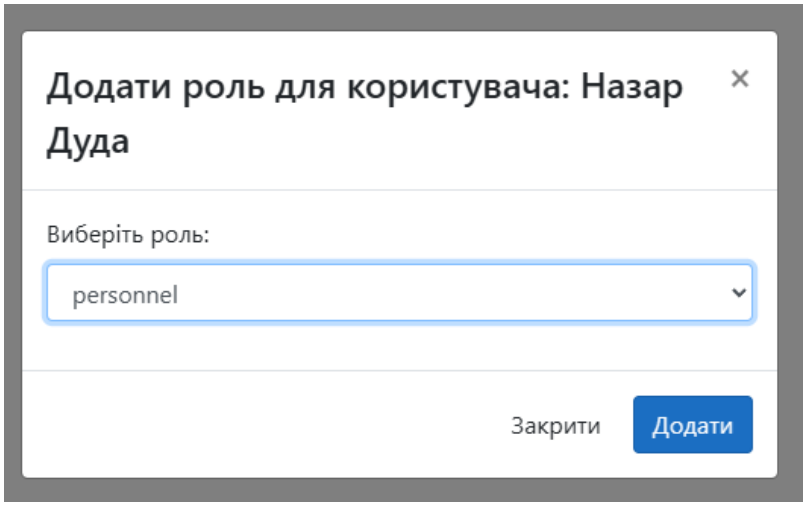


Рисунок 3.15 – Модальне вікно додавання ролі

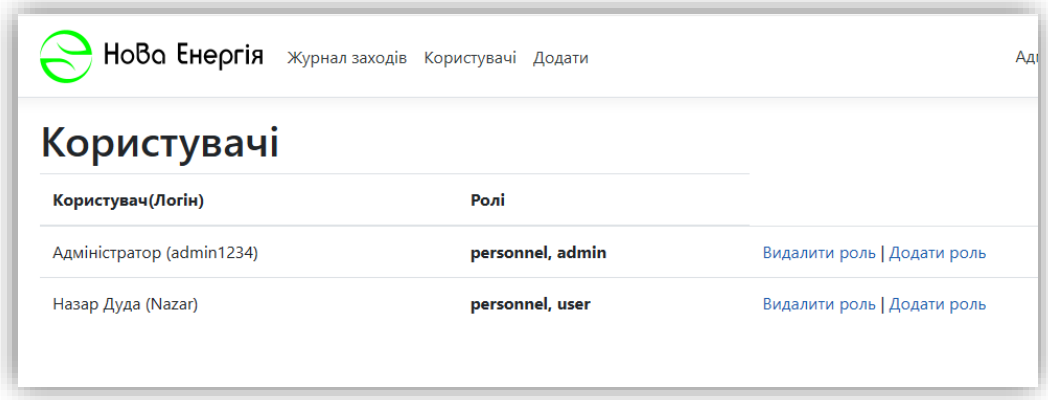


Рисунок 3.16 – Список користувачів після змін

Модальне вікно реалізовано за допомогою jquery на js. На сайті присутнє це вікно, але воно заховане і відповідно при спробі додавання спрацьовує даний js скрипт:

```
function setUserId(event) {
    let userId = event.target.getAttribute("data-user-id");
    let userName = event.target.getAttribute('data-user-name');
    let modalTitle = document.getElementById('userName');
    modalTitle.textContent = 'Додати роль для користувача: ' +
userName;
    $("#myModal").data("userId", userId);
}
```

```
function submitForm() {
    let roleId = document.getElementById("RoleName").value;
    let userId = $("#myModal").data("userId");
    $.ajax({
        url: "/Administration/AddUserRole",
        type: "POST",
        data: { roleId: roleId, userId: userId},
        success: function () {
            $("#myModal").modal("hide");
            window.location.reload();
        }
    });
}
```

У випадку якщо потрібно прибрати певну роль у користувача, то адміністратор натискає “Видалити роль” і його перенаправляє на спеціальну сторінку, де пише що це за користувач та які ролі він має, це продемонстровано на рисунку 3.17. Далі навпроти кожної ролі адміністратор бачить іконку кошика, після натискання на котру він видалить роль у даного користувача і знову його буде переслано на сторінку усіх користувачів, де він моментально може спостерігати зміни, як на рисунку 3.16.

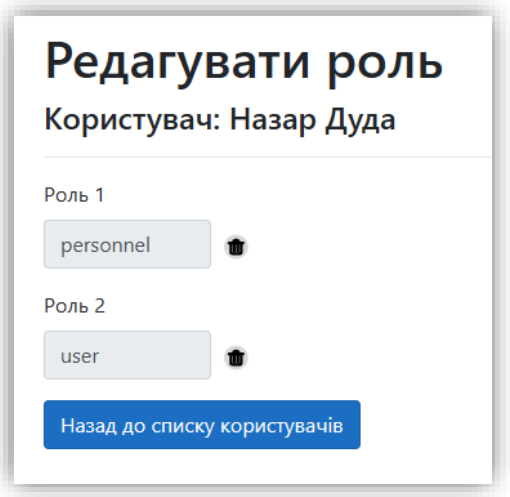


Рисунок 3.17 – Сторінка де пише ролі користувача

Коли адміністратор надав роль персоналу, то я можу зайти із свого акаунта і побачити, що мені відкрився доступ до сторінки “Журнал заходів”. Тут можна побачити форму фільтрації даних (рис. 3.18).

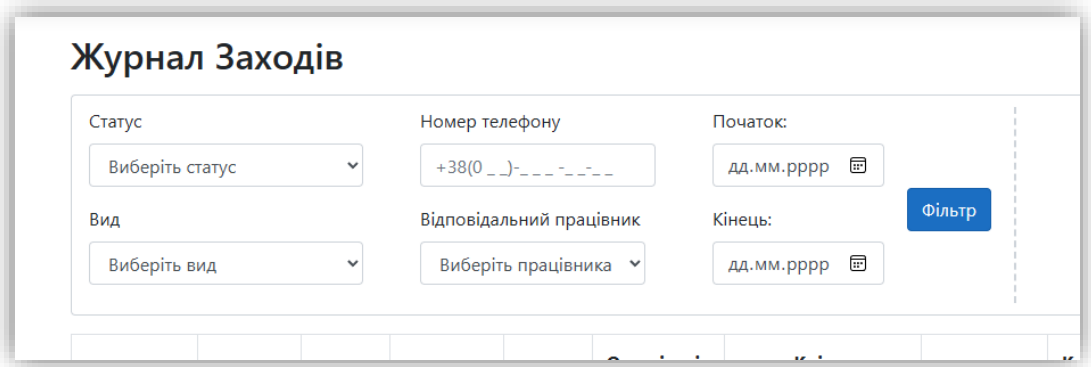


Рисунок 3.18 – Форма фільтрації даних

Отже тут можна фільтрувати по статусу, захід може бути підтвердженим або не підтвердженим, це залежить від того чи після того як клієнти записалися на певний захід, він прийшов відвідали захід і розрахувався, так зване бронювання.

Вид – це тип заходу до прикладу: Кріо-шоу, Тесла, Алхімія, Персональне відвідування, Екскурсія та інші.

Номер телефону – це відповідно номер контактної особи для зв’язку із нею.

Відповідальний працівник – це перелік працівників/персоналу які будуть відповідальну за проведення того чи іншого заходу. Тут буде випадаючий список із користувачами, які мають роль персонал.

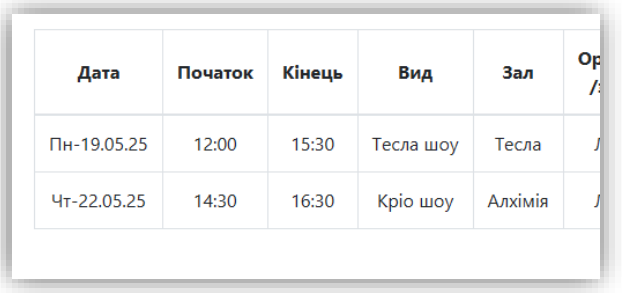
Дата початку і кінця – це відповідно з якої по яку дату відображати заходи.

Ці всі параметри можуть вибиратися разом так і по одинці. У випадку якщо запису який відповідає параметрам фільтрації немає то результатом буде пуста таблиця.

Поетапно почнемо розбирати стовпці таблиці. На рисунку 3.19 показано частину стовпців. Бачимо, що є стовпець дата, відображається він так що показує день тижня, зроблено це за використанням форматування строки а саме:

```
@el.StartDateTime.ToString("ddd-dd.MM.yy", new CultureInfo("uk-UA"))
```

Час для двох наступних стовпців дістається відповідно через форматування дати у строку і виводі необхідних значень. Два наступних стовпця Вид і Зал є простими і інтуїтивно зрозумілими.



Дата	Початок	Кінець	Вид	Зал	Ор /:
Пн-19.05.25	12:00	15:30	Тесла шоу	Тесла	Л
Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Л

Рисунок 3.19 – Перша частина таблиці заходів

Отже наступна частина таблиці (рис. 3.20) показано такі стовпці:

– організація/заклади – назви навчальних заклидів та організацій;

– клієнт – клієнт або представник організації чи закладу;

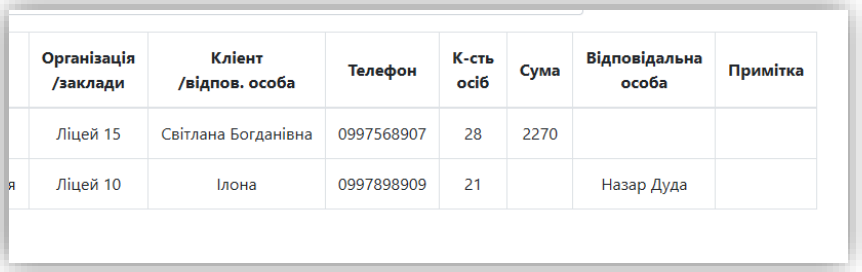
– телефон – контактний телефон відповідальної особи;

– к-сть осіб – загальна сума осіб що прийшла на захід;

– сума – оплата за послуга, це поле ще може бути пустим, якщо послуга безкоштовна;

– відповідальна особа – відповідальна особа із працівників представників персоналу, береться вона із бази даних користувачів хто має роль персонал, поле може бути пустим якщо відповідальна особа не потребується;

– примітка – додаткова інформація, може не заповнюватись.



Організація /заклади	Клієнт /відпов. особа	Телефон	К-сть осіб	Сума	Відповідальна особа	Примітка
Ліцей 15	Світлана Богданівна	0997568907	28	2270		
Ліцей 10	Ілона	0997898909	21		Назар Дуда	

Рисунок 3.20 – Друга частина таблиці заходів

Отже прости користувач може користуватися тільки фільтрацією, зроблено це щоб він міг відшукати себе і знав за що відповідальний.

Оскільки користувач із цією роллю більше немає можливостей взаємодіяти із сайтом повернемося до користувача із правами адміністратора. Зайшовши у ту ж сторінку “Журнал заходів” ми бачимо що добавилися кнопки “Додати запис” та “Експорт” (рис. 3.21). Кнопка експорт експортує всі необхідні дані у Excel таблицю та завантажує її користувачу, але про її детальну роботу буде описано пізніше. Кнопка додати запис перекине користувача на форму для створення ного заходу і додавання його у таблицю заходів.

Також біля кожного запису з’явилися іконки інструментів:

– олівець – можливість редагувати запис;

– галочка – переводить запис у стан “Підтверджено”, про який згадувалося раніше;

– файл – повністю копіює даний запис;

– кошик – видаляє даний запис із бази та таблиці.

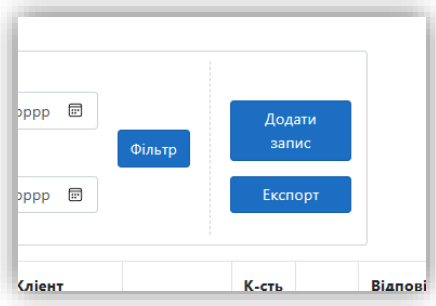


Рисунок 3.21 – Додаткові можливості взаємодії

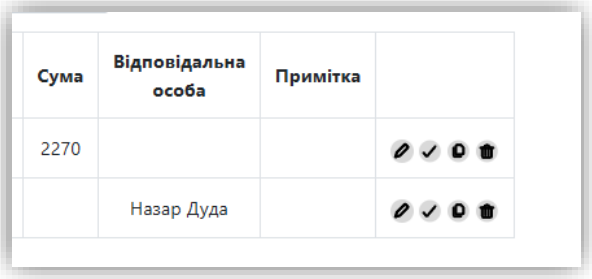


Рисунок 3.22 – Інструменти взаємодії із записами

Отже спробуємо додати запис натиснувши кнопку “Додати запис”.

Потрапляємо ми на сторінку створення запису, да форма поділена на дві складові. Першу складову це дані про клієнта показано на рисунку 3.23.

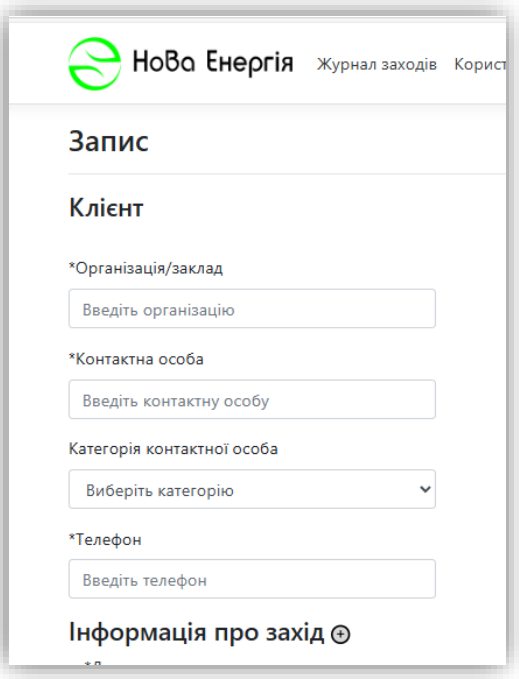


Рисунок 3.23 – Перша частина створення нового запису

Тут поля, які потребують заповнення позначено “*” інакше форма покаже помилку користувачу таку як це показано на рисунку 3.24.

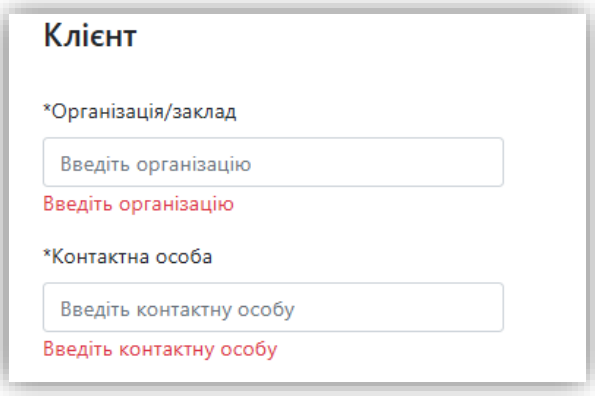


Рисунок 3.24 – Повідомлення про необхідність заповнення

Отже розглянемо ці поля від найлегшого до найважчого у реалізації:

– категорія контактної особи – це випадаючий список де перелічено всі можливі категорії (рис. 3.25);

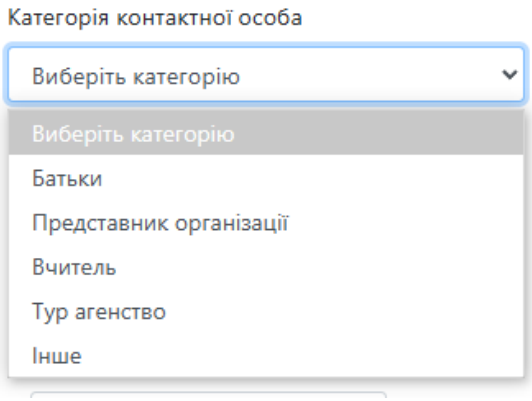


Рисунок 3.25 – Випадаючий список із категоріями

– контактна особа – це власне ім’я особи з якою в разі чого можна контактувати;

– телефон – телефон особи/клієнта, який треба заповнити у правильному форматі інакше буде помилка як на рисунку 3.26, форма приймає як з +38 так і без;

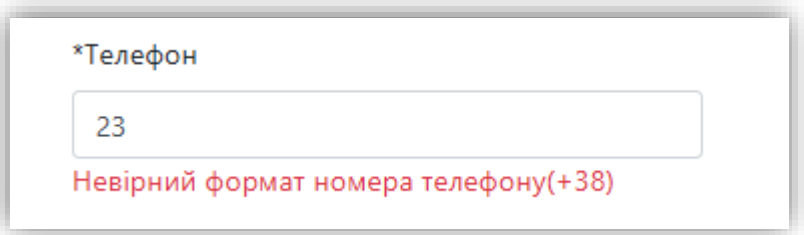


Рисунок 3.26 – Неправильний формат номера телефону

– організація/заклад – тут потрібно ввести організацію, але бували випадки коли приходили клієнти, що вже були і відповідно їх назва вже є у базі даних, то було прийнято рішення реалізувати Ajax запит.

Ajax запит працює таким чином, користувач вводить букву і моментально відправляється запит у базу даних, де запит шукає всі назви організації на цю букву, далі ці дані потрапляють у випадаючий список звідки користувач може вибрати із запропонованих. Код даного запиту із використанням jquery показано нижче:

```
$(function () {
    let autocompleteInput = $('#inputOrganization');
    autocompleteInput.autocomplete({
        source: function (request, response) {
        $.ajax({
        url: '/ActivityRecord/GetSuggestions',
        type: 'GET',
        data: { term: request.term },
        dataType: 'json',
        success: function (data) {
            response(data);
        }
    });
},
    minLength: 1
    });
});
```

На рисунку 3.27 продемонстровано декілька варіантів роботи запиту, якщо ввести перші букви назви яких є у базі даних, і якщо нові. Отже якщо даних не знайдено то допоміжного випадаючого списку не буде.

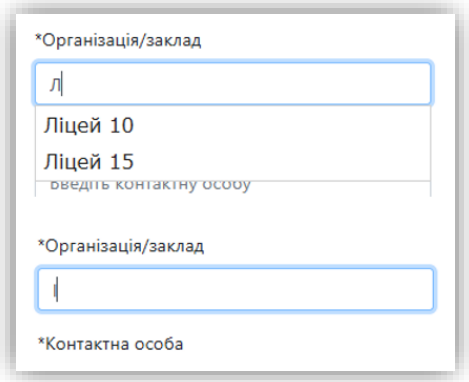


Рисунок 3.27 – Робота Ajax запиту

Наступна складова запису це інформація про сам захід(рис. 3.28), тут користувач заповнює дату і час початку, він має зручний інтерфейс де просто із запропонованого календаря вибирає дату і час. Так само час закінчення. Тип

послуги, подано у вигляді випадаючого списку, дані в який заносять із таблиці TypeOfServices. Аналогічно як з типи так само із залами, є випадаючий список, дані в який заносять із таблиці Halls.

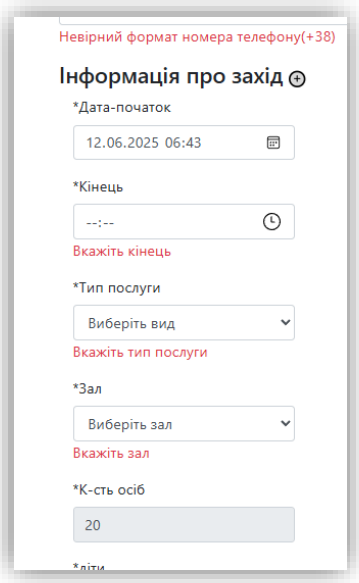


Рисунок 3.28 – Друга частина створення нового запису

Також на рисунку 3.29 показано такі поля як:

– к-сть осіб – це загальна к-сть осіб на заході, це поле не можна заповнювати, бо воно рахується автоматично як сума дорослих і дітей;

– діти/дор. – це поля де вводиться к-сть дітей і дорослих відповідно, і ці дані є основними для поля к-сть осіб;

– Ч/Ж – це к-сть чоловіків і жінок в цій групі людей;

– вікова група – це середній вік групи.

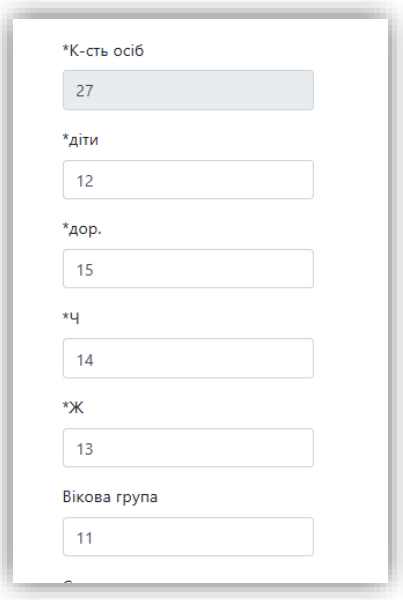


Рисунок 3.29 – Заповнення даних про осіб групи

У випадку якщо сума дітей і дорослих не співпаде із сумою чоловіків і жінок в групі користувач отримає дану помилку, що показано на рисунку 3.30, і відповідно такий запис не відправиться у базу даних. Реалізовано це з використанням jquery та java script:

```
$('#BtnSubmitAll').click(function () {
    let countOfChildren =
parseInt($('#inputCountOfChildren').val());
    let countOfAdults = parseInt($('#inputCountOfAdults').val());
    let countOfMaleGender =
parseInt($('#inputCountOfMaleGender').val());
    let countOfFemaleGender =
parseInt($('#inputCountOfFemaleGender').val());
    if ((countOfChildren + countOfAdults) !== (countOfMaleGender +
countOfFemaleGender)) {
        alert('Сума дітей та дорослих не дорівнює сумі чоловіків
та жінок');
        return false;

    }
});
```

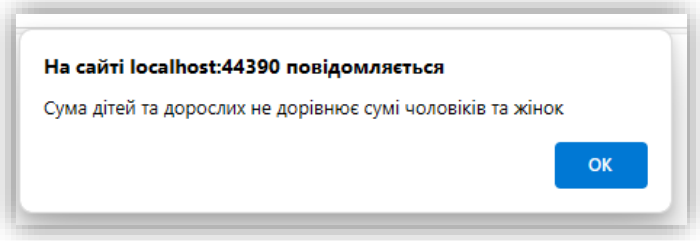


Рисунок 3.30 – Помилка при введені невірних даних

Тепер розглянемо останні поля форми (рис. 3.31). Тут ми бачимо що є поле сума, де приблизно пишеться вартість послуги для к-сть дітей/дорослих заданих полями вище, реалізовано це знову ж таки js, невеликий фрагмент скрипта показано нижче:

```
function sumOfActivity() {
    const valueAdults = parseInt(inputCountOfAdults.value) || 0;
    const valueChildren = parseInt(inputCountOfChildren.value) ||
0;

    let sum;
    switch (selectTypeOfService.value) {
        case "Персональне відвідування":
            sum = (valueAdults * 70) + (valueChildren * 50);
            break;
        case "Тесла шоу":
            sum = (valueAdults * 90) + (valueChildren * 80);
            break;
        …
     }
     inputSum.placeholder = "~ " + sum + " грн";
}
```

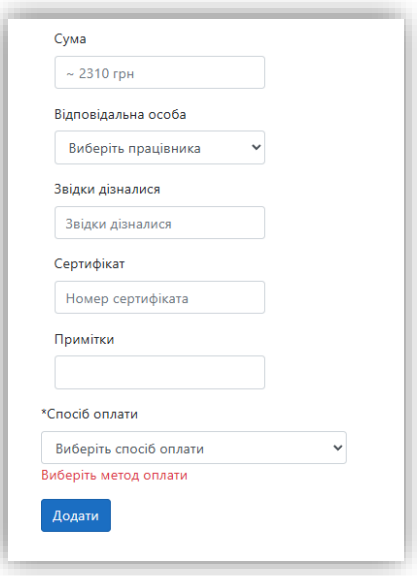
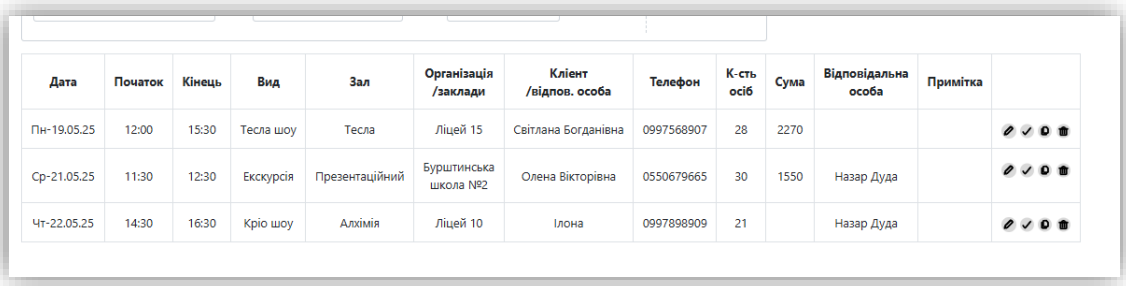


Рисунок 3.31 – Останні поля форми для створення запису

У полі відповідальна особа так як і параметрі фільтрації на сторінці “Журнал заходів” буде показано випадаючий список і користувач повинен вибрати серед зареєстрованих працівників. Поля “Звідки дізналися” , “Сертифікат”, “Примітка” є інформативними і використовуються для бору інформації. Поле “Спосіб Оплати” є схожим до полів “Зал” і “Вид” заходу.

Вірно заповнивши ці дані і натиснувши кнопку “Додати” користувач попаде на сторінку ‘Журнал заходів”, де вже буде новий запис, продемонстровано це на рисунку 3.32.



Дата	Початок	Кінець	Вид	Зал	Організація /заклади	Клієнт /відпов. особа	Телефон	К-сть осіб	Сума	Відповідальна особа	Примітка	
Пн-19.05.25	12:00	15:30	Тесла шоу	Тесла	Ліцей 15	Світлана Богданівна	0997568907	28	2270			
Ср-21.05.25	11:30	12:30	Екскурсія	Презентаційний	Бурштинська школа №2	Олена Вікторівна	0550679665	30	1550	Назар Дуда		
Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Ліцей 10	Ілона	0997898909	21		Назар Дуда		

Рисунок 3.32 – Таблиця після додавання нового запису

Однак це ще не всі можливості, які є на сторінці із створенням нового запису, є приклади коли один клієнт може замовити декілька заходів і щоб не переписувати все по новій, біля надпису “Інформація про захід” є іконка як на

рисунку 3.33. При натисканні створюється ще одна форма з інформацією про новий захід, продемонстровано це на рисунку 3.34.

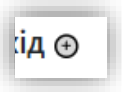


Рисунок 3.33 – Іконна створення додаткового заходу

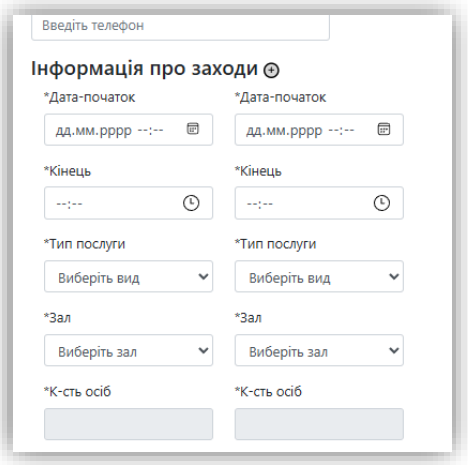


Рисунок 3.34 – Створення додаткового заходу для одно клієнта

Якщо іконку помилково використана то знизу конкретного розділу “Інформація про захід” є кнопка “Видалити”, яка прибирає даний запис (рис. 3.35).

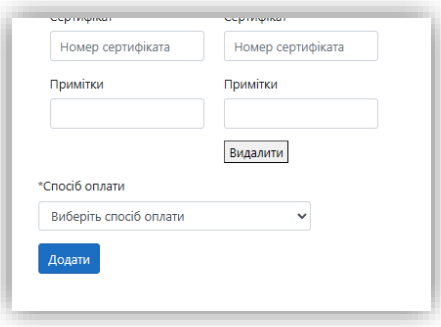


Рисунок 3.35 – Кнопка для видалення певного запису

Таких записів можна створювати безліч, але видаляти їх потрібно поступово для коректної роботи коду, кнопка “Видалити” для попереднього запису буде неактивна поки не видалиться даний (рис. 3.36).

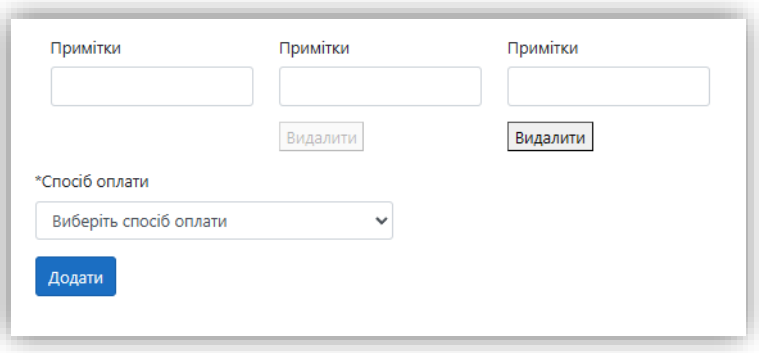
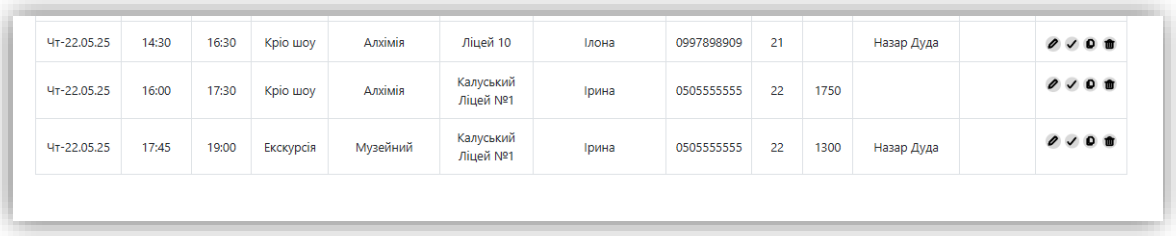


Рисунок 3.36 – Неактивна кнопка “Видалити для попереднього запису”

Реалізовано це з використанням jquery та js, фрагмент з фалу Create.js подано нижче:

```
function addColumnWithElements(indexOfColumn, btn){
    let el = $("#MyColumn").clone();
    el.attr("id", "MyColumn" + indexOfColumn);
    $("#MyConte").append(el);
    document.getElementById("MyColumn" +
indexOfColumn).style.marginLeft = "25px";
    renameAttrForAllFields(indexOfColumn);
    placeholdersForCol(indexOfColumn);
    let input = document.getElementById("inputStartDateTime" +
indexOfColumn);
    input.setAttribute("required", "");
    $("#" + ("MyColumn" + indexOfColumn)).append(btn);
    document.getElementById("btnDelCol").id = "btnDelCol" +
indexOfColumn;
    if(indexOfColumn >= 2){
       if(document.getElementById("btnDelCol" + (indexOfColumn -
1)) === null){
           window.location.reload();
       }
       document.getElementById("btnDelCol" + (indexOfColumn -
1)).disabled = true;
    }
}
```

Тут показано тільки один метод проте їх є більше і весь код файлу буде у додатку А. Отож спробуємо створити декілька записів на одного клієнта одночасно. Результат можна побачити на рисунку 3.37, це є достатньо корисною функцією, яка пришвидшує роботу адміністратора.



Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Ліцей 10	Ілона	0997898909	21		Назар Дуда		
Чт-22.05.25	16:00	17:30	Кріо шоу	Алхімія	Калуський Ліцей №1	Ірина	0505555555	22	1750			
Чт-22.05.25	17:45	19:00	Екскурсія	Музейний	Калуський Ліцей №1	Ірина	0505555555	22	1300	Назар Дуда		

Рисунок 3.37 – Додавання декількох записів одночасно

Тепер розглянемо роботу інструментів для кожного запису. Найпростіший інструментами є підтвердження та видалення запису. Працюють вони за певним схожим принципом. Натискаємо на іконку підтвердження і нам з’являється модальне вікно (рис. 3.38), де у користувача перепитають чи він дійсно хоче зробити цю дію. Він буде мати змогу підтвердити або відмінити дію. Після підтвердження у базі даних де значення IsConfirmed поміняється на true і це вже буде рахуватися як послуга що надалася повністю.

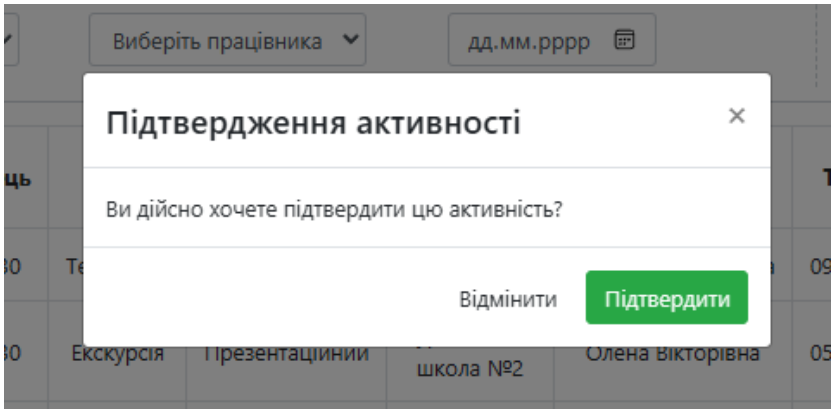


Рисунок 3.38 – Модальне вікно для підтвердження запису

Після підтвердження користувача буде переправлено на сторінку “Журнал заходів”, де він побачить що запис зник, оскільки відображаються тільки не підтверджені записи, щоб побачити підтверджені йому потрібно у параметрах

фільтрації вибрати це. Результат на рисунку 3.39.

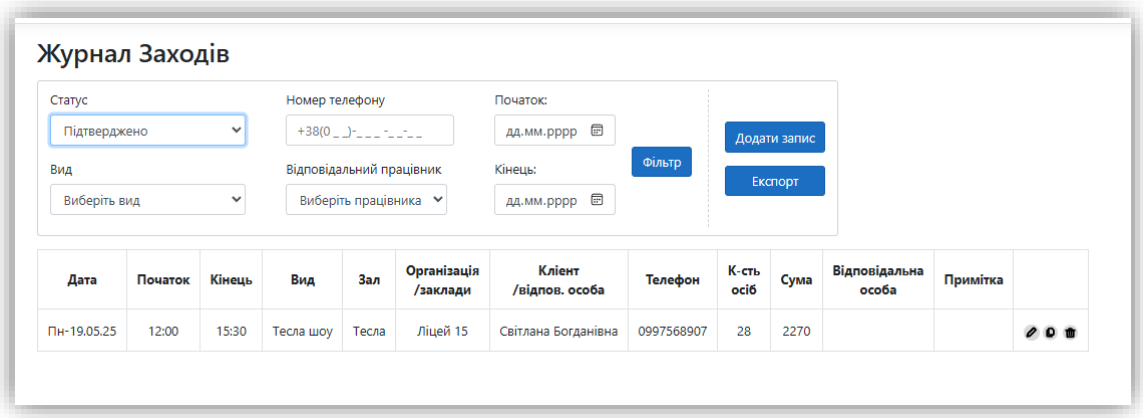


Рисунок 3.39 – Результат підтвердження і фільтрації запису

Наступний інструмент “Корзина” також видає модальне вікно як на рисунку 3.40.

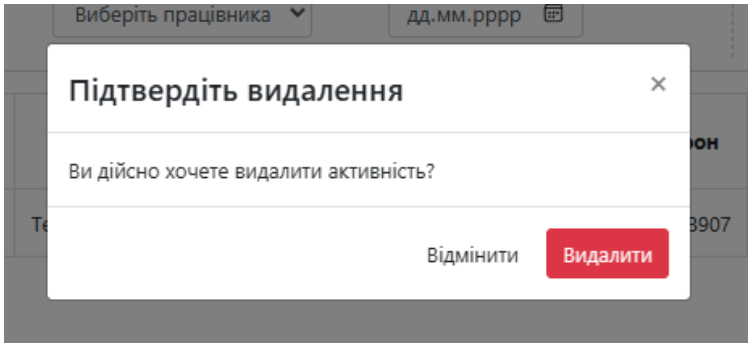
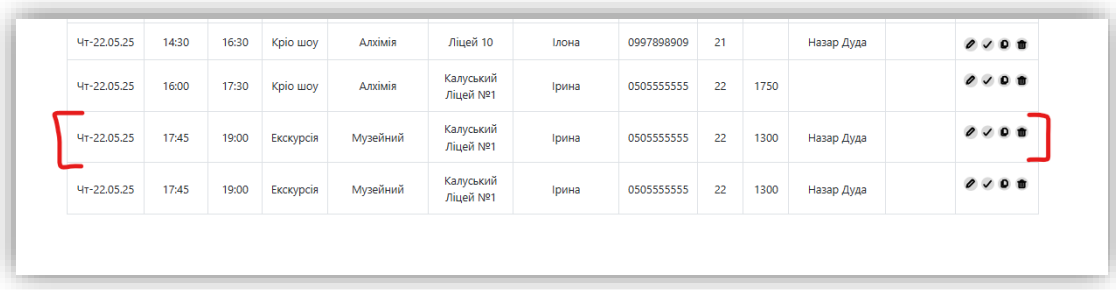


Рисунок 3.40 – Модальне вікно для видалення запису

Коли користувач натисне “Видалити” то запис видалиться із бази даних і відповідно не буде показуватися у таблиці.

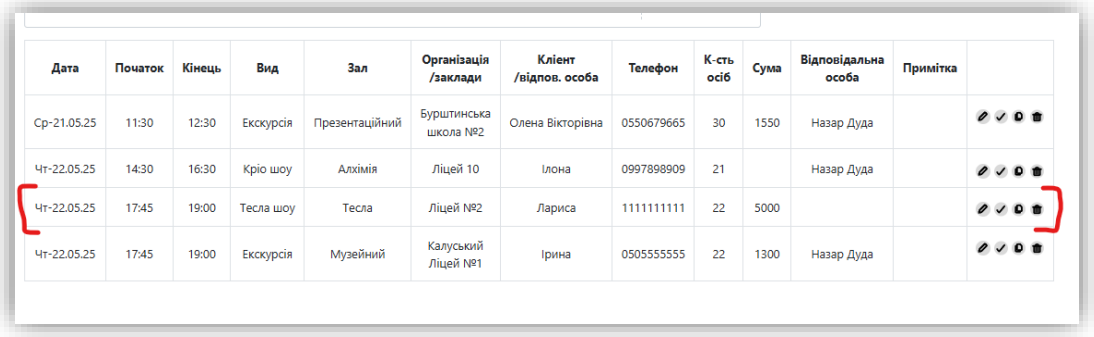
Функція копіювання повністю створює новий запис із усіма даними запису-оригіналу. Результат копіювання показано на рисунку 3.41. Як показано на рисунку виділений запис повністю був скопійованим у запис нижче.



Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Ліцей 10	Ілона	0997898909	21		Назар Дуда	
Чт-22.05.25	16:00	17:30	Кріо шоу	Алхімія	Калуський Ліцей №1	Ірина	0505555555	22	1750		
Чт-22.05.25	17:45	19:00	Екскурсія	Музейний	Калуський Ліцей №1	Ірина	0505555555	22	1300	Назар Дуда	
Чт-22.05.25	17:45	19:00	Екскурсія	Музейний	Калуський Ліцей №1	Ірина	0505555555	22	1300	Назар Дуда	

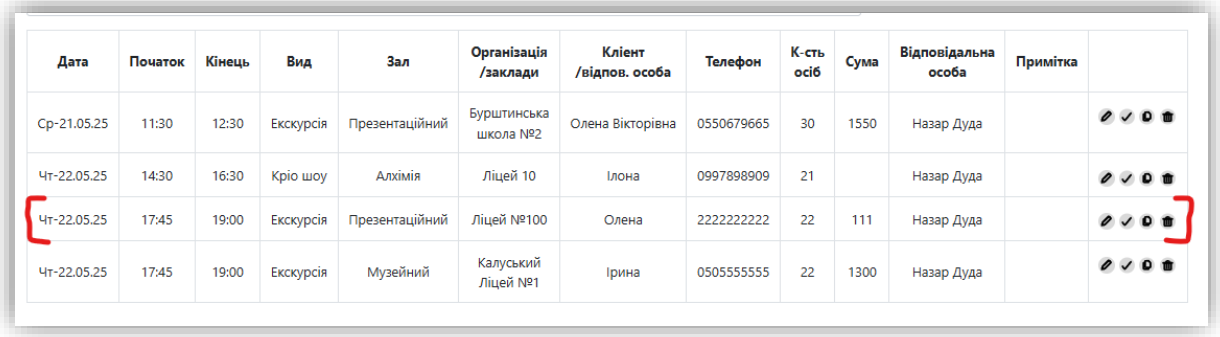
Рисунок 3.41 – Результат копіювання

Функція редагування працює наступним чином, користувач натискає на іконку і його перекидає на сторінку редагування, яка є схожою до сторінки створення запису, тільки там уже поля заповнені відповідними даними. Нижче показано два рисунки 3.42 та 3.43, які показую записи до редагування і після редагування.



Дата	Початок	Кінець	Вид	Зал	Організація /заклади	Клієнт /відпов. особа	Телефон	К-сть осіб	Сума	Відповідальна особа	Примітка	
Ср-21.05.25	11:30	12:30	Екскурсія	Презентаційний	Бурштинська школа №2	Олена Вікторівна	0550679665	30	1550	Назар Дуда		
Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Ліцей 10	Ілона	0997898909	21		Назар Дуда		
Чт-22.05.25	17:45	19:00	Тесла шоу	Тесла	Ліцей №2	Лариса	1111111111	22	5000			
Чт-22.05.25	17:45	19:00	Екскурсія	Музейний	Калуський Ліцей №1	Ірина	0505555555	22	1300	Назар Дуда		

Рисунок 3.42 – Таблиця записів до редагування



Дата	Початок	Кінець	Вид	Зал	Організація /заклади	Клієнт /відпов. особа	Телефон	К-сть осіб	Сума	Відповідальна особа	Примітка	
Ср-21.05.25	11:30	12:30	Екскурсія	Презентаційний	Бурштинська школа №2	Олена Вікторівна	0550679665	30	1550	Назар Дуда		
Чт-22.05.25	14:30	16:30	Кріо шоу	Алхімія	Ліцей 10	Ілона	0997898909	21		Назар Дуда		
Чт-22.05.25	17:45	19:00	Екскурсія	Презентаційний	Ліцей №100	Олена	2222222222	22	111	Назар Дуда		
Чт-22.05.25	17:45	19:00	Екскурсія	Музейний	Калуський Ліцей №1	Ірина	0505555555	22	1300	Назар Дуда		

Рисунок 3.42 – Таблиця записів після редагування

Також адміністратор у меню зверху має надпис “Додати”, коли він

наводиться то у нього з’являється випадаючий список де пише зали, способи оплати, послуги, після натискання на будь який з надписів користувач потрапляє на сторінку із відповідним надписом. На рисунку 3.43 зображено сторінку із залами.

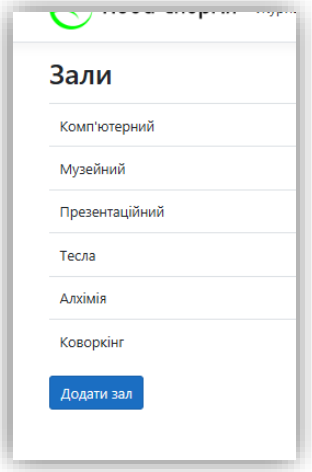


Рисунок 3.43 – Сторінка із залами

Тут є тільки одна кнопка “Додати зал”, оскільки видалення не передбачено технічним завданням. При натисканні на цю кнопку з’являється модальне вікно де користувач вписує назву залу і потім може його додати (рис.3.44).

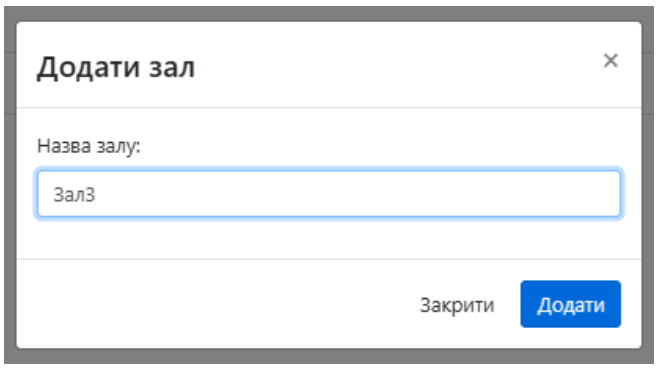


Рисунок 3.44 – Модальне вікно для додавання залу

Далі розглянемо процес експору даних з таблиці у excel. Для того щоб це реалізувати потрібно встановити додаткову бібліотеку для роботи з excel таблицями ClosedXML. Створив сервіс ExcelActivitiesService і там прописав методи що створюють таблицю, називають стовпці, отримують модель і дані

записують і у правильні стовпці, код даного класу показано у додатку А. Отже натиснувши на кнопку еспорт користувач побачить, користувач побачить що у нього пішов процес скачування таблиці (рис. 3.45).

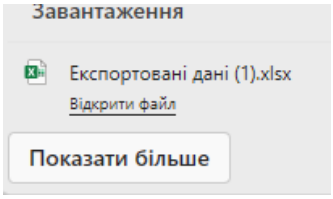


Рисунок 3.45 – Завантаження таблиці

Коли користувач відкриє таблицю то у нього будуть відформатовані дані як на рисунку 3.46.

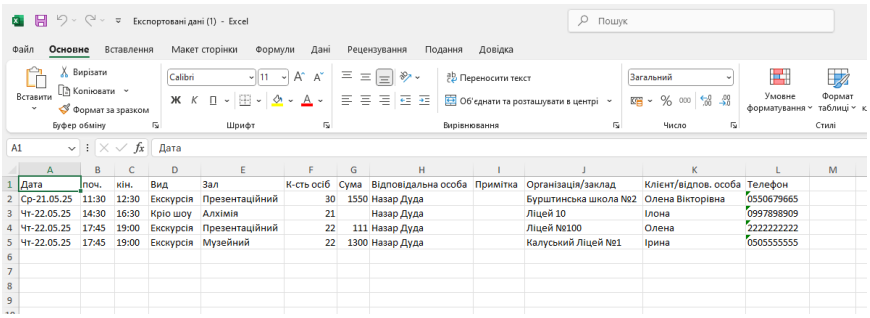


Рисунок 3.46 – Дані у Excel таблиці

Також користувач може експортувати відфільтровані дані, дані спочатку можуть відфільтруватися а потім попасти у парсер таблиці. Для цього потрібно вибрати дані фільтрації, але не натискати кнопку “Фільтр”, а зразу на “Експорт”, перевірим це завантаживши у excel таблиці тільки записи які були 21.05, для цього вибираєм ці дати у часових обмеження і експортуєм. Відкриваєм нову таблицю і дійсно бачимо там тільки 1 запис (рис. 3.47).

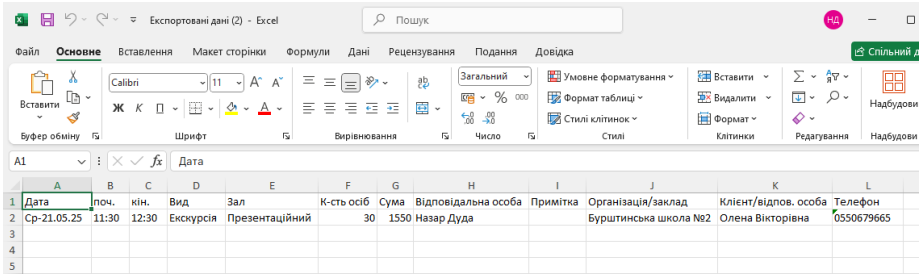


Рисунок 3.47 – Відфільтровані дані у Excel таблиці

3.2 Перевірка алгоритму хешування паролів та його коректності в інформаційній системі наукового містечка “Нова Енергія”

У сучасних інформаційних системах захист персональних даних користувачів є одним із ключових пріоритетів. Одним з найбільш вразливих елементів будь-якої системи автентифікації є зберігання паролів. Якщо паролі зберігаються у відкритому вигляді, будь-яке несанкціоноване отримання доступу до бази даних може призвести до серйозних наслідків — викрадення облікових записів, витоку конфіденційної інформації та порушення безпеки всієї системи.

Саме тому використання хешування паролів є необхідним та обов'язковим елементом захисту. Хешування дозволяє зберігати в базі даних не сам пароль, а лише його криптографічний відбиток (хеш), який неможливо або надзвичайно складно відновити у зворотному напрямку. Додатково, застосування унікального "сіль" (salt) для кожного пароля підвищує стійкість до атак типу "словниковий підбір" (dictionary attack) та "попередньо обчислені таблиці" (rainbow table attack), оскільки навіть однакові паролі матимуть різні хеш-значення.

Таким чином, реалізація механізму хешування паролів з використанням сучасних криптографічних підходів забезпечує необхідний рівень захисту інформаційної системи та дозволяє мінімізувати ризики, пов’язані з несанкціонованим доступом до облікових записів. Для впевненості у надійності такого механізму важливо провести комплексне юніт-тестування, що дозволяє переконатися в коректності роботи алгоритму в різних ситуаціях.

Отже приступив до написання xUnit тестів, нижче показано 2 тести із 6, які перевіряють правильність роботи алгоритмів хешування:

```
[Fact]
public void Hash_Should_Be_Different_For_SamePassword()
{
     string password = "TestPassword";
     string hash1 = PasswordHash.CalculateHash(password);
```

```
        string hash2 = PasswordHash.CalculateHash(password);
        Assert.NotEqual(hash1, hash2); // Різні, бо сіль різна
    }
    [Fact]
    public void Hash_Should_Match_When_CorrectPassword()
    {
        string password = "MySecretPassword";
        string hash = PasswordHash.CalculateHash(password);
        bool isValid = PasswordHash.IsPasswordsEqual(hash, password);
        Assert.True(isValid);
    }
```

Нижче описано роботу всіх 6 тестів:

– Hash_Should_Be_Different_For_SamePassword – цей тест перевіряє, що при хешуванні одного й того ж пароля кілька разів результат буде різним. Це демонструє, що алгоритм коректно використовує унікальний сіль (salt) для кожного нового хешування, що є основним принципом безпечного зберігання паролів;

– Hash_Should_Match_When_CorrectPassword - метою цього тесту є перевірка, що метод IsPasswordsEqual правильно ідентифікує вірний пароль. Тобто, якщо користувач вводить правильний пароль, система успішно розпізнає його як валідний, і метод повертає true;

– Hash_Should_Not_Match_When_WrongPassword - цей тест перевіряє ситуацію, коли користувач вводить неправильний пароль. У такому випадку метод IsPasswordsEqual має повернути false, що гарантує коректну роботу перевірки доступу;

– Hash_Should_ReturnFalse_When_EmptyPassword - даний тест перевіряє, що метод повертає false у випадку, якщо передати порожній пароль. Це дозволяє забезпечити базовий захист від некоректного або несанкціонованого вводу та знижує ризик помилок або вразливостей;

– Hash_Should_ReturnFalse_When_InvalidHashFormat - метою цього тесту є перевірка, як система реагує на хеш, який не відповідає очікуваному формату

(наприклад, якщо в рядку немає роздільника ":" або структура хеша пошкоджена). У таких випадках метод має повернути false та не допускати помилки виконання;

– Hash_Should_ReturnFalse_When_InvalidBase64 - цей тест перевіряє, чи система стійка до введення некоректного Base64 рядка (наприклад, якщо частина, яка мала би бути сіллю, не може бути декодована). В такому випадку метод також має безпечно повернути false, не допускаючи винятків або збою програми.

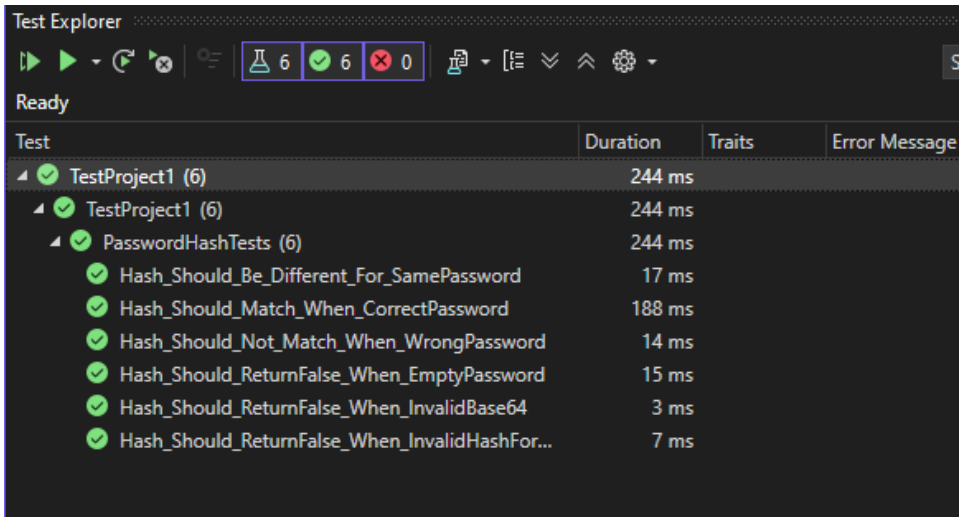


Рисунок 3.48 – Результат тестування

Після запуску тестів я отримав такий результат (рис. 3.48), це свідчить, що даний клас з його методами хешування працює правильно, а отже користувач може бути впевненим що його дані є захищеними.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було розроблено веб-сайт для адміністрування та обліку діяльності наукового містечка "Нова Енергія". Для реалізації проєкту використано мову програмування C#, фреймворк ASP.NET Core MVC та Entity Framework Core, бібліотеку jQuery.

У першому розділі проведено аналіз інформаційного забезпечення веб-сайту, зосереджено увагу на вивченні основних потреб наукового містечка "Нова Енергія". На основі цього сформульовано вимоги до проєкту: розробити зручний веб-сайт із функціями обліку, адміністрування даних та управління інформацією про користувачів і послуги.

У другому розділі здійснено побудову структури веб-сайту, визначено функціонал для кожного типу користувачів, спроєктовано базу даних із чітко описаними таблицями та зв’язками між ними. Окрім цього, створено діаграми послідовностей, що відображають основні сценарії взаємодії користувача із системою, а також розроблено діаграми USE-CASE, які демонструють доступні функції для звичайного користувача та адміністратора.

У третьому розділі виконано розробку та впровадження веб-сайту для наукового містечка "Нова Енергія", розгорнуто базу даних, налаштовано архітектуру застосунку та реалізовано основний функціонал. У підсумку створено робочий веб-сайт, який забезпечує адміністрування користувачів, облік послуг, перегляд та редагування інформації. Проведено тестування працездатності веб-сайту та перевірку ключових функцій. Також виконано модульне тестування алгоритму хешування паролів за допомогою інструменту xUnit, що підтвердило правильність його роботи та безпеку обробки облікових даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Фрімен А. Pro ASP.NET Core MVC 2: підручник. – Apress, 2017. – 1017 с.
2. Шарп Дж. Visual C# 2022 і платформа .NET 6.0: підручник. – Microsoft Press, 2022. – 960 с.
3. Метц В. Г., Робертс С. Дж., Ларсен Б. Microsoft SQL Server 2019 для професіоналів: підручник. – К.: Діалектика, 2021. – 912 с.
4. Ліберті Дж. Programming C# 8.0: підручник. – O'Reilly Media, 2020. – 704 с.
5. Еспозіто Д. Modern Web Development with ASP.NET Core 3: підручник. – Microsoft Press, 2020. – 432 с.
6. Страал Р. West Wind WebSurge and Web Performance with ASP.NET Core: підручник. – West Wind Technologies, 2021. – 284 с.
7. Нагель К. Professional C# and .NET – 2021 Edition: підручник. – Wrox, 2021. – 1504 с.
8. Уотсон Б. Writing High-Performance .NET Code: підручник. – Kindle Edition, 2020. – 425 с.
9. Ділейні К. SQL Server Internals: підручник. – Microsoft Press, 2022. – 960 с.
10. Прейс Дж., Фрімен А. Microsoft ASP.NET Core in Action: підручник. – Manning Publications, 2021. – 624 с.

ДОДАТКИ

Вміст файлу «Starup. cs»

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authentication.Cookies;
using MyApp.DB;
using MyApp.DB.Entities;
using MyApp.Services.Interfaces;
using MyApp.Services.Services;

namespace MyApp.Web
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method
to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultCon
nection")));


services.AddAuthentication(CookieAuthenticationDefaults.Authentica
tionScheme)
                    .AddCookie(options =>
                    {
                        options.LoginPath = new
Microsoft.AspNetCore.Http.PathString("/Account/Login");
                    });

            services.AddAutoMapper(GetType().Assembly);

            services.AddScoped<IIdentityService,
IdentityService>();
            services.AddScoped<IClientService, ClientService>();

ManagerService>();
            services.AddScoped<IActivityService,
```

```
ActivityService>();
            services.AddScoped<IChangeListsService,
ChangeListsService>();
            services.AddScoped<IActivityRecordService,
ActivityRecordService>();
            services.AddScoped<IExcelActivitiesService,
ExcelActivitiesService>();
            //services.AddScoped<IExcelTimeRecordsService,
ExcelTimeRecordsService>();
            services.AddControllersWithViews();
        }

        public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern:
"{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

Вміст файлу «Program. cs»

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using Microsoft.Extensions.DependencyInjection;
using MyApp.DB;
```

```
namespace MyApp.Web
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();
            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;
                try
                {
                    var context =
services.GetRequiredService<AppDbContext>();
                    SampleData.Initial(context);
                }
                catch (Exception ex)
                {
                    var logger =
services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding
the DB.");
                }
            }
            host.Run();
        }

        public static IHostBuilder CreateHostBuilder(string[]
args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Вміст файлу «CreateActivityRecord. js»

```
$(document).ready(function () {

    $(function () {

        let i = 1;

        let btn = `<div class="form-group">

                    <button type="button" id="btnDelCol"
onclick="delColumn(id)" >Видалити</button>
```

```
        </div>`;
        placeholdersForCol();
        $("#btnAddCol").on("click", function () {
            $("#spanInfo").text("Інформація про заходи");
            addColumnWithElements(i, btn);
            i++;
        });
    });
});
function addColumnWithElements(indexOfColumn, btn){
    let el = $("#MyColumn").clone();
    el.attr("id", "MyColumn" + indexOfColumn);
    $("#MyConte").append(el);
    document.getElementById("MyColumn" +
indexOfColumn).style.marginLeft = "25px";
    renameAttrForAllFields(indexOfColumn);
    placeholdersForCol(indexOfColumn);
    let input = document.getElementById("inputStartDateTime" +
indexOfColumn);
    input.setAttribute("required", "");
    $("#" + ("MyColumn" + indexOfColumn)).append(btn);
    document.getElementById("btnDelCol").id = "btnDelCol" +
indexOfColumn;
    if(indexOfColumn >= 2){
       if(document.getElementById("btnDelCol" + (indexOfColumn -
1)) === null){
           window.location.reload();
       }
       document.getElementById("btnDelCol" + (indexOfColumn -
```

```
 1)).disabled = true;

    }

}

function delColumn(idOfElement) {

    let digits = idOfElement.match(/\d+/g);

    let idOfColumn = 'MyColumn' + digits[0];

    $("#" + idOfColumn).remove();

    if(parseInt(digits[0]) === 1){

        window.location.reload();

    }

    document.getElementById("btnDelCol" + (digits[0] -
1)).disabled = false;

}


$('#BtnSubmitAll').click(function () {

    let countOfChildren =
parseInt($('#inputCountOfChildren').val());

    let countOfAdults = parseInt($('#inputCountOfAdults').val());

    let countOfMaleGender =
parseInt($('#inputCountOfMaleGender').val());

    let countOfFemaleGender =
parseInt($('#inputCountOfFemaleGender').val());


    if ((countOfChildren + countOfAdults) !== (countOfMaleGender +
countOfFemaleGender)) {

        alert('Сума дітей та дорослих не дорівнює сумі чоловіків
та жінок');

        return false;

    }
```

```
});


function placeholdersForCol(i = "FirstCol") {
    let idOfSelectTypeOfService;
    let idOfInputCountOfPeople;
    let idOfInputCountOfChildren;
    let idOfInputCountOfAdults;
    let idOfInputSum;


    if(i === "FirstCol"){
        idOfSelectTypeOfService = "selectTypeOfService";
        idOfInputCountOfPeople = "inputCountOfPeople";
        idOfInputCountOfChildren = "inputCountOfChildren";
        idOfInputCountOfAdults = "inputCountOfAdults";
        idOfInputSum = "inputSum";
    }
    else {
        idOfSelectTypeOfService = "selectTypeOfService" + i;
        idOfInputCountOfPeople = "inputCountOfPeople" + i;
        idOfInputCountOfChildren = "inputCountOfChildren" + i;
        idOfInputCountOfAdults = "inputCountOfAdults" + i;
        idOfInputSum = "inputSum" + i;
    }
    let selectTypeOfService =
document.getElementById(idOfSelectTypeOfService);
    let inputCountOfPeople =
document.getElementById(idOfInputCountOfPeople);
    let inputCountOfChildren =
```

```
document.getElementById(idOfInputCountOfChildren);
    let inputCountOfAdults =
document.getElementById(idOfInputCountOfAdults);
    let inputSum = document.getElementById(idOfInputSum);

    [inputCountOfAdults, inputCountOfChildren].forEach(input => {
        input.addEventListener("input", sumOfAdultsAndChildren);
        input.addEventListener("input", sumOfActivity);
    });
    selectTypeOfService.addEventListener("change", sumOfActivity);

    function sumOfAdultsAndChildren() {
        const valueAdults = parseInt(inputCountOfAdults.value) ||
0;
        const valueChildren = parseInt(inputCountOfChildren.value)
|| 0;
        inputCountOfPeople.value = valueAdults + valueChildren;
    }
    function sumOfActivity() {
        const valueAdults = parseInt(inputCountOfAdults.value) ||
0;
        const valueChildren = parseInt(inputCountOfChildren.value)
|| 0;

        let sum;
        switch (selectTypeOfService.value) {
            case "Персональне відвідування":
                sum = (valueAdults * 70) + (valueChildren * 50);
                break;
```

```
            case "Тесла шоу":
                sum = (valueAdults * 90) + (valueChildren * 80);
                break;
            case "Екскурсія":
                const valuePeople =
parseInt(inputCountOfPeople.value)
                if (valuePeople >= 1 && valuePeople <= 15) {
                    sum = 975;
                } else if (valuePeople >= 16 && valuePeople <= 20)
{
                    sum = 1100;
                } else if (valuePeople >= 21 && valuePeople <= 25)
{
                    sum = 1250;
                } else if (valuePeople >= 26 && valuePeople <= 30)
{
                    sum = 1350;
                } else{
                    sum = 0;
                    break;
                }
                break;
            default:
                sum = 0;
        }
        inputSum.placeholder = "~ " + sum + " грн";
    }
}
```

```
function renameAttrForInputField(i, nameOfField){
    let father = $("#" + ("MyColumn" + i));
    let input = father.find("#input" + nameOfField);
    let label = father.find("#label" + nameOfField);
    let span = father.find("#span" + nameOfField);
    label.attr("for", "Activities_" + i + "__" + nameOfField);
    input.attr("name", "Activities[" + i + "]." + nameOfField);
    span.attr("data-valmsg-for", "Activities[" + i + "]." +
nameOfField);

    label.attr("id", ("label" + nameOfField) + i);
    input.attr("id", ("input" + nameOfField) + i);
    span.attr("id", ("span" + nameOfField) + i);
}
function renameAttrForSelectField(i, nameOfField){
    let father = $("#" + ("MyColumn" + i));
    let label = father.find("#label" + nameOfField);
    let select = father.find("#select"  + nameOfField);
    let span = father.find("#span"  + nameOfField);
    label.attr("for", "Activities_" + i + "__" + nameOfField);
    select.attr("name", "Activities[" + i + "]." + nameOfField);
    span.attr("data-valmsg-for", "Activities[" + i + "]." +
nameOfField);

    label.attr("id", ("label" + nameOfField) + i);
    select.attr("id", ("select" + nameOfField) + i);
    span.attr("id", ("span" + nameOfField) + i);
}
```

```
function renameAttrForAllFields(i){
    renameAttrForInputField(i, "StartDateTime");
    renameAttrForInputField(i, "EndTime");
    renameAttrForSelectField(i, "TypeOfService");
    renameAttrForSelectField(i, "Hall");
    renameAttrForInputField(i, "CountOfPeople");
    renameAttrForInputField(i, "CountOfChildren");
    renameAttrForInputField(i, "CountOfAdults");
    renameAttrForInputField(i, "CountOfMaleGender");
    renameAttrForInputField(i, "CountOfFemaleGender");
    renameAttrForInputField(i, "AgeGroup");
    renameAttrForInputField(i, "Sum");
    renameAttrForSelectField(i, "ResponsiblePerson");
    renameAttrForInputField(i, "FindFrom");
    renameAttrForInputField(i, "NumberOfCertificate");
    renameAttrForInputField(i, "Note");
}
```

Вміст файлу «CreateActivityRecord. cs»

```
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Reflection;
using System.Threading.Tasks;
using AutoMapper;
using ClosedXML.Excel;
using MyApp.Services.Interfaces;
```

```
using MyApp.Services.Models;


namespace MyApp.Services.Services
{
    public class ExcelActivitiesService : IExcelActivitiesService
    {
        private readonly IMapper _mapper;
        private IXLWorksheet _worksheet;


        public ExcelActivitiesService(IMapper mapper)
        {
            _mapper = mapper;
        }


        public async Task<byte[]>
GetArrayOfBytes(List<ActivityRecordModel> list)
        {
            using XLWorkbook workbook = new XLWorkbook();
            CreateAndFillWorkSheetForExcursions(workbook, list);
            using var stream = new MemoryStream();
            workbook.SaveAs(stream);
            await stream.FlushAsync();
            return stream.ToArray();
        }


        private void
CreateAndFillWorkSheetForExcursions(XLWorkbook workbook,
List<ActivityRecordModel> list1)
```

```
        {
            _worksheet = workbook.AddWorksheet("Лист1");
            FillNameOfColumnsForExcursions();
            var list =
_mapper.Map<List<ExcelActivityRecordModel>>(list1);
            FillDataFromListOfExcursions(list);
            MakeAdjustOfColumns(12);
        }
        private void FillNameOfColumnsForExcursions()
        {
            _worksheet.Cell("A1").SetValue("Дата");
            _worksheet.Cell("B1").SetValue("поч.");
            _worksheet.Cell("C1").SetValue("кін.");
            _worksheet.Cell("D1").SetValue("Вид");
            _worksheet.Cell("E1").SetValue("Зал");
            _worksheet.Cell("F1").SetValue("К-сть осіб");
            _worksheet.Cell("G1").SetValue("Сума");
            _worksheet.Cell("H1").SetValue("Відповідальна особа");
            _worksheet.Cell("I1").SetValue("Примітка");
            _worksheet.Cell("J1").SetValue("Організація/заклад");
            _worksheet.Cell("K1").SetValue("Клієнт/відпов.
особа");
            _worksheet.Cell("L1").SetValue("Телефон");
        }
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: **Сайт для адміністрування та обліку даними наукового містечка “Нова енергія” з використанням ASP .NET Core MVC, entity framework та jquery**

Обсяг пояснювальної записки 66 аркушів:

10 таблиць;

59 рисунків;

1 додаток.

Дата завершення роботи: *12 червня 2025р*.

Підпис студента- _____*Дуда Н.В*.