Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет-технологій

ЗАТВЕРДЖУЮ

завідувач кафедри
мережевих та інтернет технологій

_____ ЮрійКРАВЧЕНКО

«_____» _____2022 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Галузі знань 17 «Електроніка та телекомунікації»
За спеціальністю 172 «Телекомунікації та радіотехніка»
Освітньо-професійна програма «Мережеві та інтернет-технологій»

на тему:

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ ВІДОБРАЖЕННЯ РОЗКЛАДУ ЗАНЯТЬ

Виконав: студент групи МІТ -41

НИЧИПОРУК ДМИТРО _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: доцент кафедри мережевих та інтернет-технологій
(посада)

к. т. н ОЛЕКСАНДР МАХОВИЧ _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ - 2022

Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра мережевих та інтернет-технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

Мережевих та інтернет-технологій

_____ Юрій КРАВЧЕНКО

«_____»_____2021 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Ничипорук Дмитро

(прізвище, ім'я, по батькові)

1. Тема роботи:

Розробка Web-додатку для відображення розкладу занять

Затверджена на засіданні кафедри МІТ «24» грудня 2021 р. протокол №8

2. Термін здачі закінченої роботи «30» травня 2022 р.

3. Вихідні дані до проєкту (роботи) Сучасні технології створення Web-додатків

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)

Вступ

1. Дослідження технологій та фреймворків для створення Web-додатків. Постановка задачі

1. 1 Огляд поширених фреймворків для створення Web-додатків. Аналіз предметної області

1.2 Постановка задачі

2. Вибір архітектури, платформи розробки, технологій та мови програмування.

3. Розробка Web-додатку.

3. 1. Проєктування моделі даних.

3. 2. Реалізація серверної частини.

3. 3. Реалізація клієнтської частини.

Висновки

5. Перелік графічного матеріалу 8-10 слайдів

Дата видачі завдання

Керівник роботи Олександр МАХОВИЧ

(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Дмитро НИЧИПОРУК

(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	02. 05. 2022	
2	Розділ 1	10. 05. 2022	
3	Розділ 2	15. 05. 2022	
4	Розділ 3	20. 05. 2022	
5	Доповідь та слайди	25. 05. 2022	
6	Пояснювальна записка	30. 05. 2022	

Здобувач вищої освіти _____ Дмитро НИЧИПОРУК

(підпис)

Керівник _____ Олександр МАХОВИЧ

(підпис)

РЕФЕРАТ

Бакалаврська робота на тему “Розробка Web-додатку для відображення розкладу занять”.

Склад пояснювальної записки: 65 с, 24 рис, 6 додатків, 6 джерел

Об’єкт дослідження: імплементація розкладу занять в навчальний процес.

Мета роботи: створення веб-додатку для відображення розкладу занять.

Предмет дослідження: Система виведення розкладу.

Методи дослідження: Програмування серверної частини, порівняння фреймворків.

В роботі проведено аналіз процесу розробки веб-додатку. Та порівняння різних інструментів реалізації веб-додатку.

Запропоновано вирішення проблеми з доведенням до відому студетнів та вчителів роскладу занятть.

Побудовано повноцінний веб-додаток для відображення розкладу занять

Результати здійснених у дипломному проєкті досліджень можуть бути використані як в цільовому навчальному закладі, так і в будь-якому іншому закладі освіти.

Ключові слова :CSS, HTML, RAZOR, C#, MVC, web-додаток, База даних, Веб-сайт.

ABSTRACT

Bachelor's thesis on "Development of a Web-application to display the schedule of classes. "

The composition of the explanatory note: 65 p, 24 figures, 6 appendices, 6 sources

Object of research: implementation of the schedule in the educational process.

Purpose: Create a web application to display the class schedule.

Subject of research: Schedule derivation system.

Research methods: Programming of the server part, comparison of frameworks.

The analysis of web application development is carried out in the work. Comparison of different tools of web application implementation is carried out.

A solution to the problem of informing students and teachers about the schedule of classes is proposed.

A full-fledged web application has been built to display the class schedule

The results of the research carried out in the diploma project can be used both in the target educational institution and in any other educational institution.

Keywords: CSS, HTML, RAZOR, C #, MVC, web-application, Database, Website.

Зміст

ВСТУП

Як відомо, в загальному розумінні розклад це документ що регламентує трудовий ритм, розглядається як фактор оптимізації навчального процесу. Проте в сучасному світі використання паперових документів для швидкого та точного представлення інформації групі людей є малоефективним та недоречним. Саме тому потрібне рішення яке дозволить отримати доступ до інформації про розклад занять будь-де та будь-коли й дозволить змінювати при потребі цю інформацію в будь-який момент. З цією метою і була створена ця робота.

В сучасному еру інформаційних технологій у будь-якого університету є власний сайт з інформацією про нього. Розклад це те на чому базується навчання і саме з цієї причини потрібно забезпечити своєчасний доступ до нього. Також потрібно забезпечити. Саме цю інформацію можна отримати в даному Web-додатку.

Ціллю даної роботи є проєктування та розробка Web-додатку для відображення готового розкладу занять

Додаток буде надавати користувачам інформацію про місце, а саме номер аудиторії та час проведення заняття. Користувачами будуть студенти та викладачі університету. Перевіряти та редагувати інформацію будуть адміністратори.

В першому розділі роботи представлений аналіз технологій що можуть бути використані для розробки веб додатків

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА ФРЕЙМВОРКІВ ДЛЯ СТВОРЕННЯ WEB-ДОДАТКІВ. ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд поширених фреймворків для створення Web-додатків. Аналіз предметної області

В цьому розділі будуть досліджені найпоширеніші фреймворки та технології для створення веб-додатків. А також представлені їх основні принципи роботи та архітектура.

HTML(англ. HyperText Markup Language) — це стандартизована мова розмітки гіпертексту документів для перегляду веб сторінок у браузері. Веб-браузери отримують HTML документ від сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора. Практично є основним елементов в створенні будь-яких сайтів або веб-додатків.

CSS (англ. Cascading Style Sheets) - це спеціальна мова стилю сторінок, ця мова використовується для опису їх зовнішнього вигляду. Найчастіше CSS використовується саме для HTML та XML сторінок, проте може бути застосована і в інших рішеннях. CSS використовується для того, щоб визначити кольори, шрифти, верстку та інші аспекти зовнішнього вигляду сторінки. Також в CSS є можливість розділити зміст сторінки це може покращити сприйняття та доступність контенту додати гнучкості та контролю за відображенням.

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS.

JavaScript (JS) - динамічна, об'єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися

даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки.

Node. js – це популярна платформа з відкритим кодом застосовується для виконання мережевих застосунків написаних на JavaScript. Якщо раніше JavaScript застосовувався для обробки даних в браузері користувача, то node. js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання що дозволило повноцінне використання JS як засіб для Back-end розробки. Архітектура Node. js зображена на рис 1.1

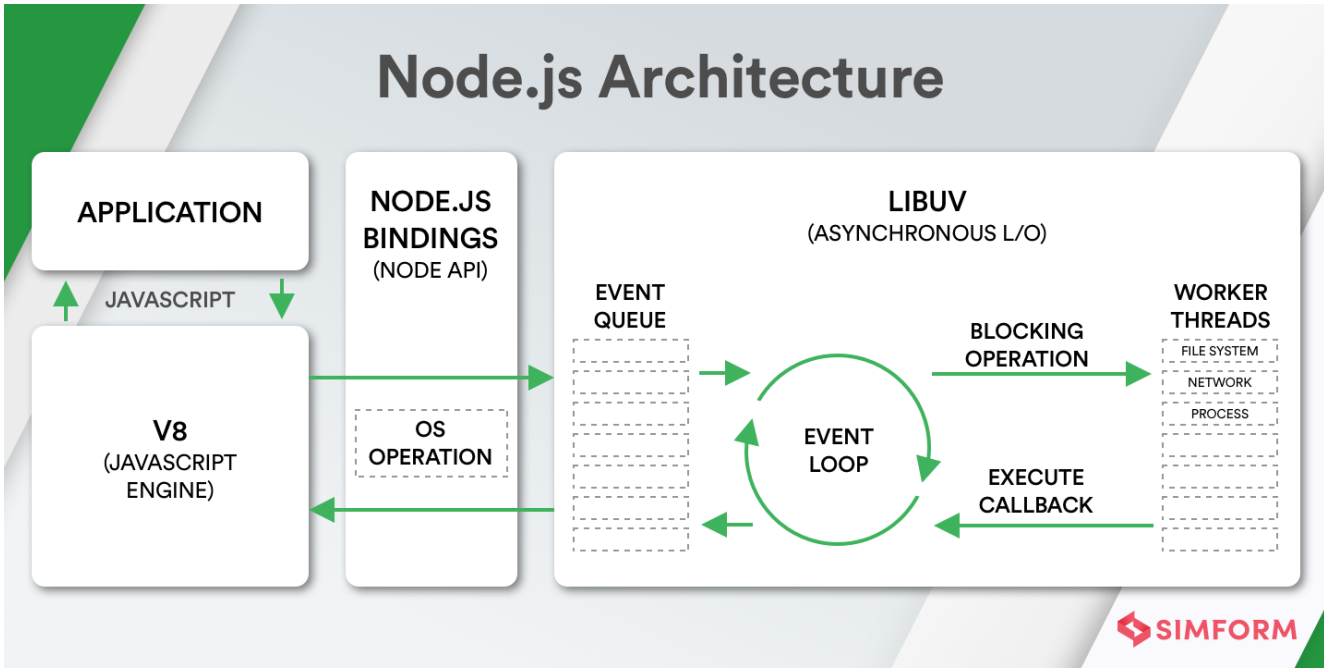


Рисунок 1.1-Архітектура роботи Node. js

Django - безкоштовний і відкритий фреймворк для створення веб додатків, написаний мовою програмування Python. Веб фреймворк — набір компонентів, які допомагають розробляти веб-сайти швидше і простіше. Також використовує паттерн MVC. Архітектура Django зображена на рис. 1.2

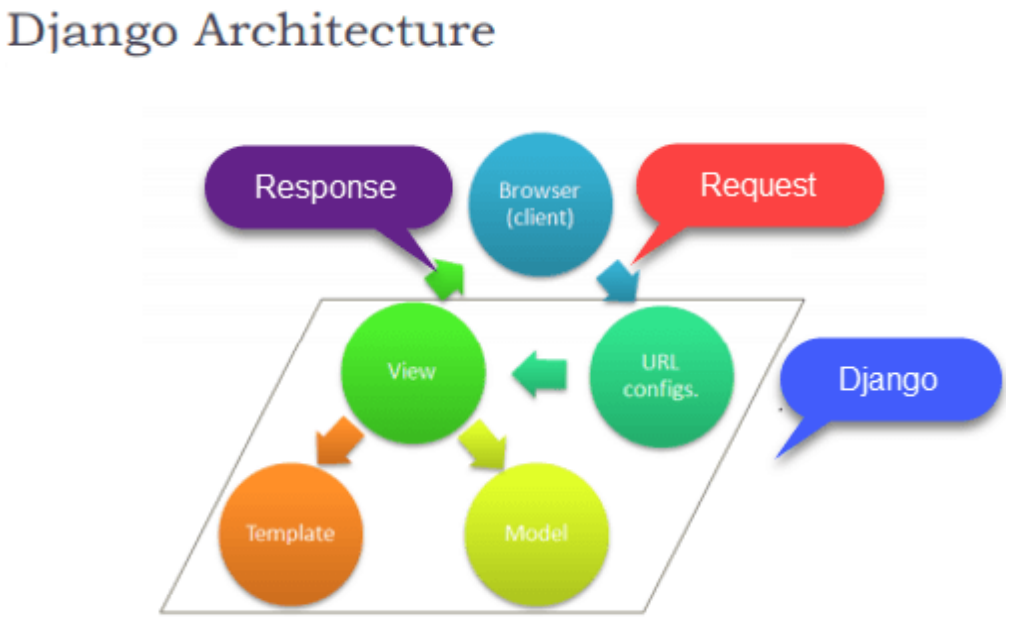


Рисунок 1.2- Архітектура Django

Bootstrap- це набір інструментів у вільному доступі для створення веб-сторінок та веб-додатків. Працює з HTML і CSS-шаблонами та з оформленням блоків навігації, кнопок, веб-форм та з іншими елементами веб інтерфейсу.

AngularJS(Angular) — JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки роботи з односторінковими додатками. Односторінковий додаток це додаток що складається з однієї HTML сторінки з CSS і JavaScript і динамічно завантажується у відповідь на дії користувача. Мета Angular — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки. Фреймворк працює зі сторінкою HTML, що містить додаткові атрибути та пов'язує області вводу або виводу сторінки з моделлю, яка є звичайними змінними JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних. На рисунку 1.3 зображена архітектура Angular.

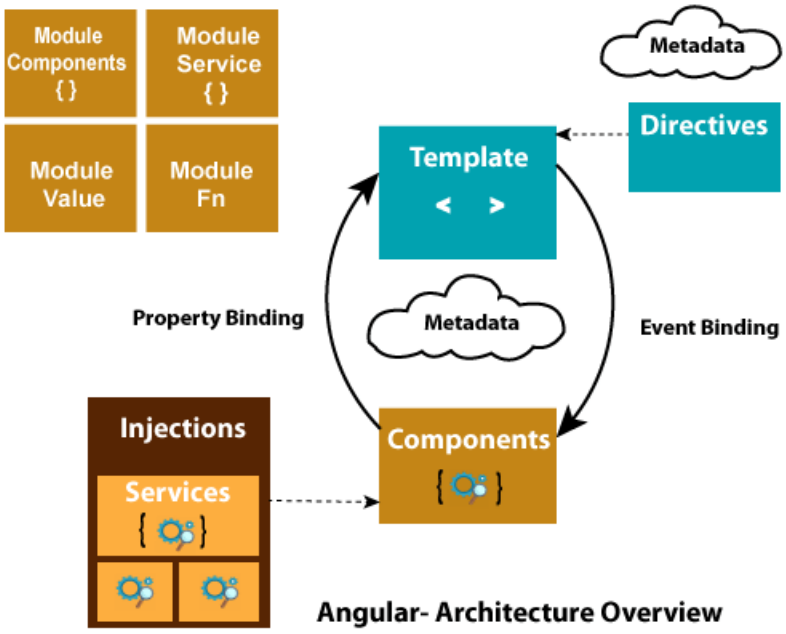


Рисунок 1.3 — Архітектура Angular

ASP .NET ENTITY FRAMEWORK(EF)- це рішення для роботи з базами даних, яке використовується у програмуванні мовами сімейства. NET. Воно дозволяє взаємодіяти з СУБД з допомогою сутностей (entity), а не таблиць. Також код із використанням EF пишеться набагато швидше. Наприклад, працюючи з базами даних безпосередньо, розробник повинен турбуватися про підключення, підготовку SQL і параметрів, надсилання запитів та транзакцій. На Entity Framework все це робиться автоматично — програміст працює безпосередньо з сутностями та тільки говорить EF, що потрібно зберегти зміни, як саме це працює представлено на рисунку 1.4.

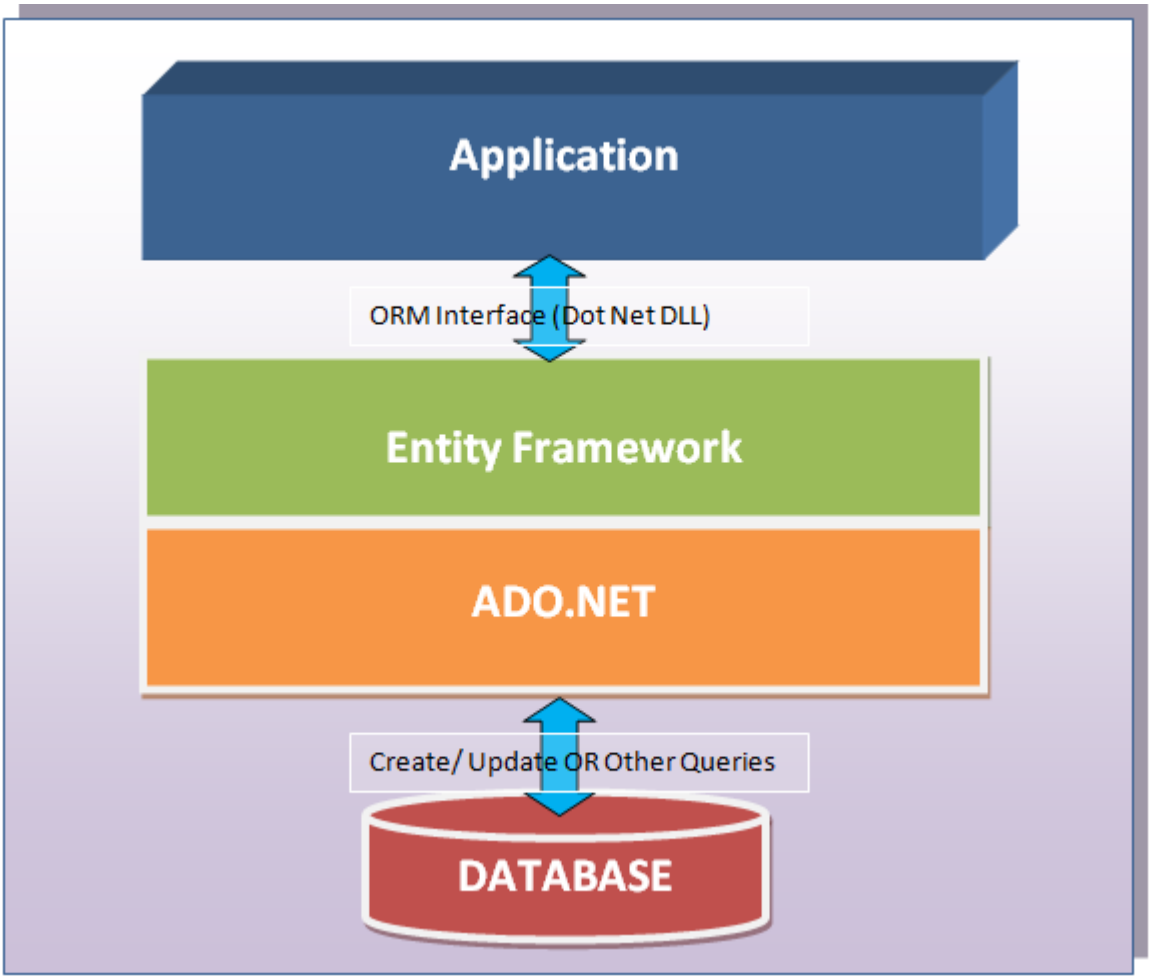


Рисунок 1.4 — Схема роботи Entity framework

1.2 Постановка задачі

Метою даної роботи є створення прототипу веб-додатку, що дозволяє створювати та демонструвати розклад занять в навчальному закладі

Для цього було сформовано наступний перелік завдань:

- Оглянути технології що можуть реалізувати поставлену задачу
- Розробити архітектуру веб-додатку
- Реалізувати модель даних для додатка
- Реалізувати багатокористувацький інтерфейс
- Провести тестування

2 ВИБІР АРХІТЕКТУРИ, ПЛАТФОРМИ РОЗРОБКИ, ТЕХНОЛОГІЙ ТА МОВИ ПРОГРАМУВАННЯ.

1 Вибір архітектури

Архітектура веб-програми в основному представляє відносини та взаємодії між такими компонентами, як інтерфейси, монітори обробки транзакцій, бази даних та інші. Основна мета – переконатися, що всі елементи правильно працюють разом. Логіка досить проста: коли користувач вводить URL-адресу у браузері та натискає "введення", браузер робить запит до сервера. Сервер відповідає, а потім показує потрібну веб-сторінку. Всі ці компоненти створюють архітектуру веб-програми.

Веб-додаток складається з трьох частин: користувальницької, серверної частин та бази даних

Клієнтська частина відображається у браузері: веб-сторінки показують вміст програми, яка зазвичай закодована за допомогою HTML, JavaScript та CSS. Основний обов'язок браузера – бути посередником між користувачем та сервером.

На сервері веб-додатків зберігаються та обробляються дані. Бекенд відповідає за інтеграцію із зовнішніми системами. Це центральна частина програми, яка контролює всі операції.

База даних зберігає всі необхідні дані та надсилає їх на сервер веб-додатків на певний запит.

Клієнтську частину зазвичай розробляють з допомогою HTML, CSS та JavaScript. Веб-браузери запускають код і перетворюють його на інтерфейс, тому немає необхідності в налаштуванні операційної системи. Щодо серверного компонента, він побудований на Java, . Net, Node. JS, Python та інших мовах програмування. Сервер складається з двох частин — логіки програми та бази даних. Логіка програми — це центр керування веб-програмою. База даних відповідає за зберігання інформації (наприклад облікових даних).

Під час створення веб-сайтів використовуються так звані архітектурні

шаблони. Архітектурний шаблон -це загальне і повторюване рішення часто виникаючої проблеми архітектури додатків у межах заданого контексту. Розглянемо найвикористовуваніші

1. Багаторівневий шаблон

Цей шаблон використовується для структурування програм, які можна розкласти на групи деяких подзадач, що є певних рівнях абстракції. Кожен шар надає послуги для наступного, вищого шару.

Загалом у веб-додатках зустрічаються наступні рівні:

- Рівень представлення(Presentation layer)
- Рівень бізнес-логіки (BLL)
- Рівень обслуговування даних (DSL)
- Рівень доступу до даних (Data Access Layer)

Рівень представлення

Рівень представлення(Presentation layer) відображає інтерфейс користувача і спрощує взаємодію з користувачем. Рівень представлення має компоненти інтерфейсу користувача, які візуалізують і показують дані для користувачів. Також існують компоненти процесу користувача, які задають взаємодію з користувачем. PL надає всю необхідну інформацію клієнтській стороні. Основна мета рівня представлення – отримати вхідні дані, обробити запити користувачів, надіслати їх у службу даних та показати результати.

Рівень бізнес-логіки (BLL)

BLL відповідає за належний обмін даними. Цей рівень визначає логіку бізнес-операцій та правил. Прикладом може слугувати вхід користувача на сайт.

Рівень обслуговування даних

DSL передає дані, опрацьовані рівнем бізнес-логіки, на рівень представлення. Цей рівень гарантує безпеку даних, ізолюючи бізнес-логіку з боку клієнта.

Рівень доступу до даних

Data Access Layer пропонує спрощений доступ до даних, що зберігаються у постійних сховищах, таких як бінарні файли та файли XML. Рівень доступу до

даних також управляє операціями CRUD – створення, читання, оновлення, видалення.

2. Клієнт-серверний шаблон

Цей шаблон складається з двох частин: сервера та безлічі клієнтів. Серверний компонент надає послуги клієнтським компонентам. Клієнти запитують послуги у сервера, а він, своєю чергою, надає ці послуги клієнтам. Більше того, сервер продовжує прослухувати клієнтські запити.

3. Дошка

Такий шаблон підходить для проблем, для яких немає чітких детермінованих рішень. Шаблон «Дошка» складається з трьох основних компонентів:

- Дошка-це структурована глобальна пам'ять, що містить об'єкти із простору можливих рішень;
- Джерело — спеціалізовані модулі зі своїм власним уявленням;
- Компоненти керування – вибирає, налаштовує та виконує модулі.

4. Модель-Представлення-Контроллер(MVC)

Цей шаблон розділяє програму на 3 частини :

Модель (Model) надає дані та реагує на команди контролера, змінюючи свій стан, існує два типи моделей:

1. Моделі відображень, які виражають самі дані, що передаються з контролерів у відображення та навпаки.
2. Моделі предметної області, що містять дані предметної області поряд з операціями, перетвореннями та правилами для створення, зберігання та маніпулювання даними.

Модель (Model) визначає всі аспекти предметної області. Модель також відповідає за збереження загального стану та цілісності даних.

Модель повинна:

- містити дані предметної області

- містити логіку для створення, керування та модифікації даних предметної області
- надавати чистий API-інтерфейс, що відкриває доступ до даних моделі та операцій над ними.

Контролер (Controller) є з’єднувальним матеріалом патерну MVC, виконуючи роль каналів між моделлю даних та їх відображенням. Контролери визначають дії, що представляють бізнес-логіку, що оперує на моделі даних та забезпечує відображення даними, які мають бути візуалізованими користувачам.

Контролер повинен:

- містити дії, що потрібні для оновлення моделі на підставі взаємодії з користувачем.

Відображення (View) містять логіку, що необхідна для відображення даних користувачу або для збору даних від користувача так, що вони можуть бути оброблені деяким методом контролера.

Для проєктування веб додатка було вибрано архітектуру MVC оскільки вона повністю відповідає цілям розробки.

2 Вибір платформи розробки, технологій та мови програмування

Основною платформою розробки для додатка було вибрано ASP .NET .

Технологія web-програмування, запропонована корпорацією Microsoft, може вважатися порівняно молодою — якщо говорити саме про ASP. NET. Тим не менш, кількість сайтів, написаних на ASP. NET, зростає буквально на очах – відповідно, збільшується і попит на фахівців, які мають її. За своєю функціональністю ця платформа перевершує багатьох своїх конкурентів, проте багато розробників віддають перевагу іншим засобам розробки через свою нелюбов до корпорації Microsoft і комерційного програмного забезпечення, в цілому. Хоча, звичайно, це все вдруге, порівняно з прив'язкою ASP. NET до операційної системи Windows.

Для створення додатку буде використано шаблон MVC функціонал якого було описано вище. Також буде використовуватися EF(entity framework) технологія що дозволяє взаємодіяти з допомогою БД в ASP. NET MVC. Наприклад, працюючи з базами даних безпосередньо, розробник повинен турбуватися про підключення, підготовку SQL і параметрів, надсилання запитів та транзакцій. На Entity Framework все це робиться автоматично — програміст працює безпосередньо з сутностями та тільки говорить EF, що потрібно зберегти зміни. Основними технологіями Frontend розробки будуть HTML CSS та JavaScript. Для написання HTML розмітки буде використовуватися синтаксис Razor.

Для авторизації буде використовуватися технологія ASP. NET Identity, це API-інтерфейс, який використовується у веб-додатках для керування даними користувачів, виконання автентичності та авторизації.

Як основна база даних буде використовуватися вбудована в Visual Studio БДО SQL Server.

Як основна мова програмування звичайно ж буде використовуватися C#

C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи. NET.

Взявши всі ці технології можна почати розробку WEB-додатку.

3 РОЗРОБКА WEB-ДОДАТКУ

3.1 проєктування моделі даних

Розробка будь-якого додатка який працює з даними починається з розробки та проєктування моделі даних. WEB-додаток не виключення, розробка моделі є основним завданням і починати розробку без моделі неможливо.

Проєктування бази даних складається з наступних етапів:

1. збір інформації;
2. визначення сутностей;
3. визначення атрибутів кожної сутності;
4. визначення зв'язків між сутностями;
5. нормалізація;
6. перетворення до фізичної моделі;
7. створення бази даних

Перші 5 етапів це логічне проєктування всі інші фізичне

Ціль логічного моделювання це створення на основі концептуальної моделі.

Результат цього моделювання залежить від вибору моделі даних(реляційна, мережева, ієрархічна) яка визначається типом використовуваної СУБД. Далі розглянемо етапи логічного моделювання.

1)Збір інформації

На цьому етапі вам необхідно точно визначити, як використовуватиметься база даних і яка інформація в ній зберігатиметься.

2)Визначення сутностей

На даному етапі визначаються сутності з яких буде складатися база даних

В представленому web-додатку всього 9 сутностей, а саме :

Користувач(User), Студент(Student), Вчитель(Teacher), група(Group), Підгрупа(Subgroup), Предмет(Subject Мається на увазі назва заняття), Семестр(Semester), Час заняття (PairTime), Заняття(Pare)

3)Визначення атрибутів кожної сутності

Атрибут є властивістю, що описує сутність. Атрибути бувають різного типу і визначаються як число, дата чи текст в залежності від потреби. Всі дані, що зберігаються в атрибуті, повинні мати однаковий тип і мати однакові властивості. У фізичній моделі атрибути називають стовпцями.

Користувач(User) містить в собі атрибути email(використовується для авторизації користувача), password (Пароль для кожного користувача свій пароль допускається створення однакових паролів, проте не бажано), role (Роль користувача в кожного користувача є роль всього є 3 ролі : Адміністратор, Вчитель та студент).

Студент(Student) містить в собі атрибути first name(ім’я студента), last name(прізвище студента).

Вчитель(Teacher) містить в собі атрибути first name(ім’я вчителя), last name(прізвище вчителя).

група(Group) містить в собі атрибути course (номер курсу в навчальному закладі), group name(прізвище вчителя).

Підгрупа(Subgroup) містить в собі атрибути subgr_name (назва підгрупи), subgr_name(Назва підгрупи).

Предмет(Subject) містить в собі один атрибут, а саме subject name(назва предмету)

Семестр(Semester) містить в собі атрибути semester name(Назва семестру),

begin date (Дата початку семестру) та end date (Дата кінця семестру)

Час заняття (PairTime) містить в собі атрибути begin_time (час початку заняття)та end_time(час закінчення заняття)

Заняття(Pare) практично основна сутність навколо якої побудований майже все функціонал додатку.

Сутність містить в собі наступний атрибут weekday(день тижня) всі інші атрибути це зовнішні ключі та вони будуть представлені в наступоному етапі.

4)Визначення зв'язків між сутностями

Реляційні бази даних дозволяють поєднувати інформацію, що належить

різним сутностям. Відношення — це ситуація, коли одна сутність посилається на первинний ключ іншої сутності.

Тип відношення визначає кількість записів сутності, пов'язаних із записом іншої сутності. Відносини поділяються на три основні типи, про які буде розказано далі

Один до одного(one-to-one)

Кожному запису першої сутності відповідає лише один запис із другої сутності. А кожному запису другої сутності відповідає лише один запис із першої сутності

Один до багатьох(one-to-many)

Кожному запису першої сутності може відповідати кілька записів з другої сутності. Однак кожному запису другої сутності відповідає лише один запис із першої сутності.

Багато до багатьох(many-to-many)

Кожному запису першої сутності може відповідати кілька записів з другої сутності. Однак і для кожного запису другої сутності може бути відповідність для багатьох записів з першої сутності

Перед початком створення зв'язків між сутностями потрібно додати поле id(Первинний ключ) до кожної існуючої в моделі сутності. Відразу після цього та розгляду всіх доступних відношень можна почати встановлювати відношення в представленій моделі даних .

Користувач(User) буде мати зв’зок з двома сутностями, а саме Teacher(Вчитель)та Student(Студент). зв'язок буде один до одного, адже для оного користувача може існувати лише один вчитель/студент. Для реалізації зв'язку додамо в сутності Teacher(Вчитель)та Student(Студент) зовнішній ключ user_id.

Студент(Student) крім описаного в сутності User зв’зку буде також мати зв’зок з сутністю Subgroup(Підгрупа) зв'язок буде мати вид one-to-many. Тобто в одній підгрупі може бути багато студентів та в одного студента може бути лише одна підгрупа. Для цього додамо в таблицю Student зовнішній ключ subgroup_id.

Вчитель(Teacher) крім описаного в сутності User зв’зку має зв’язок з

таблицею Pares(Заняття)це зв'язок типу one-to-many тобто для одного вчителя може буди багато пар, але для одної пари може бути одна пара. Більш детально він буде описаний під час розбору сутності Pares.

Група(Group) має зв’язок з сутністю Subgroup(Підгрупа) це зв’язок типу one-to-many тобто для однієї групи(Group) може існувати багато підгруп (Subgroup). Для цього додамо в таблицю Subgroup зовнішній ключ group_id.

Підгрупа(Subgroup) має зв’язок з сутностями Group та Pares. Зв’язок між Group та Subgroup описаний вище, а зв’язок між сутностями Subgroup та Pare

буде many-to-many більш детально розглянемо його в сутності Pare.

Предмет(Subject) має зв’язок з сутністю Pare. Цей зв’язок представлений у вигляді one-to-many тобто може бути багато пар з одним предметом, проте не може бути багато предметів в однієї пари. Саме додавання поля розглянемо в сутності Pare.

Семестр(Semester) має зв’язок з сутністю Pare. Цей зв’язок представлений у вигляді one-to-many тобто може бути багато пар в одному семестрів, проте не може бути багато семестрі в однієї пари. Саме додавання поля розглянемо в сутності Pare.

Час заняття (PairTime) має зв’язок з сутністю Pare. Цей зв’язок представлений у вигляді one-to-many тобто в одного поля таблиці PairTime може бути зв’язок з багатьма полями таблиці Pare, Проте не може бути багато полів таблиці PairTime в однієї пари. Саме додавання поля розглянемо в сутності Pare.

Заняття(Pare) має зв’язки з сутностями PairTime, Semester, Subgroup, Teacher, Subject

Для реалізації зв’язків між таблицями Pare та PairTime, Semester, Teacher, Subject потрібно додати в таблицю Pare поля pair_time_id, semester_id, teacher_id, subject_id відповідно всі ці зв’язки мають тип many-to-many та були описані вище.

Для реалізації зв’язку між сутністю Pare та сутністю Subgroup потрібно буде створити окрему таблицю яка буде Pare_subgroup містити зовнішні ключі pare_id та subgroup_id. Це потрібно для того, щоб реалізувати зв’язок many-to-many між цими двома сутностями, саме цей зв’язок був обраний, тому що Для багатьох

підгруп може бути багато пар і те ж саме навпаки.

5)Нормалізація

Нормалізацією називається процес видалення надлишкових даних із бази даних. Кожен елемент даних повинен зберігатися в БД тільки в одному примірнику. Існує п'ять найпоширеніших форм нормалізації. Проте як правило, база даних приводять лише до третьої нормальної форми.

6)Перетворення до фізичної моделі;

Наступним кроком, після створення логічної моделі, є побудова фізичної моделі. Фізична модель – це практична реалізація бази даних. Фізична модель визначає всі об'єкти, які мають бути реалізовані. При переході від логічної моделі до фізичної сутності перетворюються на таблиці, а атрибути на стовпці(поля).

Практична реалізація моделі даних буде зображена на рисунку 3.1

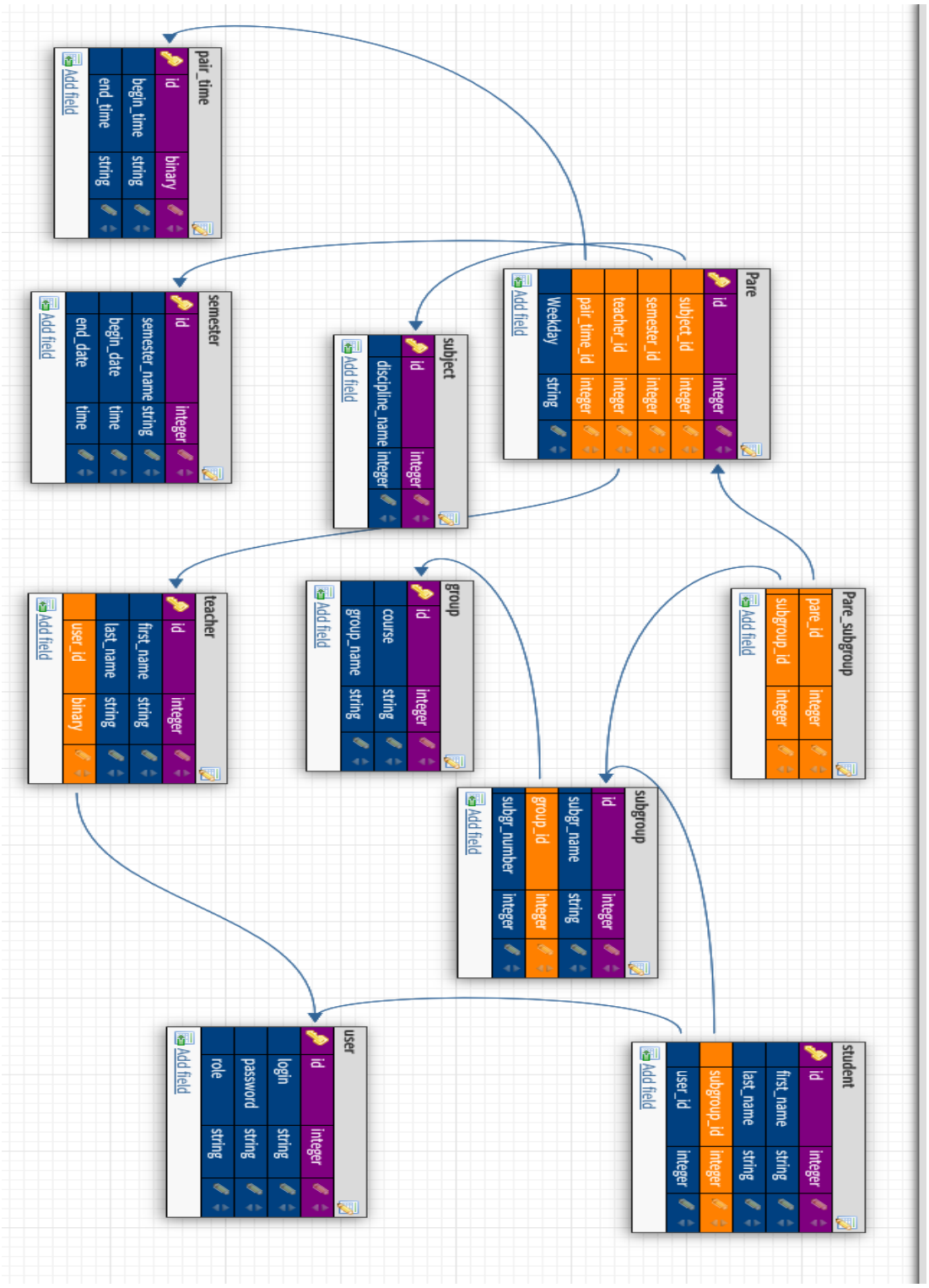


Рисунок 3.1 - Модель даних для додатку

7)Створення бази даних

Детальна реалізація БД буде розглянута в наступному розділі. Також для створення БД буде використаний інструмент SQL SERVER

3.2 Реалізація серверної частини

Бекенд – це «начинки» сайту, логіка його роботи, взаємодія веб-сервера та контенту. Ця частина є найскладнішою в реалізації та вимагає багато часу якщо порівняти з Frontend розробкою. Схема роботи Backend та загальна логіка сайту взагалі зводяться до трьох пунктів:

1. Отримання та обробка даних від користувача (наприклад, пошук запиту).
2. Обробка даних на сервері (пошук даних у базі).
3. Отримання відповіді від сервера та переведення інформації у View (виведення результату)

Загалом при розробці серверної частини перед розробником стоять наступні завдання:

- проєктування та імплементація бази даних
- Реалізація авторизації та автентифікації для користувача
- Налаштувати користувальницьких ролей.
- CRUD — Create, Read, Update, Delete

3.2.1 проєктування та імплементація бази даних

Оскільки безпосередньо проєктування БД уже було розгляну вище буде розглянута саме її імплементація з допомогою Entity Framework.

В EF користувач не повинен ніяк взаємодіяти з базою даних вся взаємодія відбувається саме з боку фреймворку. Для того, щоб створити нову таблицю

потрібно створити так званий об’єкт Entity приклад якого зображено нижче.

Спершу потрібно створити Entity та оскільки C# є об’єктно-орієнтованою мовою програмування, то Entity має представляти клас з його властивостями які вносяться до таблиці як поля бази даних. На рисунку 3.2 продемонстровано приклад Enity.

```
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
namespace ScheduleProg.Models
{

    public class PairTime
    {

        public int Id { get; set; }


        public string Begin_Time { get; set; }


        public string End_Time { get; set; }


        public string Full_Time { get{ return (Begin_Time + '-' + End_Time); } }
        [ValidateNever]

        public List<Pare> Pares { get; set; }

    }
}
```

Рисунок 3.2 -Приклад Entity

Далі можна додати клас, який використовуватиметься для підключення до бази даних, — він називається контекстом. Весь код контексту приведений в додатку Г, а на рисунку 3.3 буде подано лише фрагмент:

```
namespace ScheduleProg.Data
{

    47 references
    public class ApplicationDbContext : IdentityDbContext<User>
    {
        15 references
        public DbSet<Group> Groups { get; set; }
        14 references
        public DbSet<PairTime> PairTimes { get; set; }
        30 references
        public DbSet<Pare> Schedules { get; set; }
        16 references
        public DbSet<Semester> Semesters { get; set; }

        14 references
        public DbSet<Student> Students { get; set; }

        14 references
        public DbSet<Teacher> Teachers { get; set; }

        21 references
        public DbSet<Subgroup> Subgroups { get; set; }

        11 references
        public DbSet<PareSubgroup> PareSubgroups { get; set; }
```

Рисунок 3.3 -Фрагмент контексту

Контекст містить в собі Dbset саме таким чином формується вміст для бази даних. Також всі об’єкти типу Dbset мають бути у множині. Також в контексті реалізуються зв’язки між сутностями на поданих зображеннях представлено створення зв’язків двох видів при цьому для many-to-many використовується таблиця посередник зі зв’язками one-to-many до обох таблиць на рисунку 3.4 зображено зв'язок від subject до pare, а на рисунку 3.5 від pare до subject

```
  /*b*/
  builder.Entity<Subject>()
.HasMany<Pare>(s => s.Pares)
.WithOne(p => p.Subject)
.HasForeignKey(p => p.Subject_Id);
  /*e*/
```

Рисунок 3.4 -Приклад зв'язку many-to-one

```
builder.Entity<Teacher>()
.HasOne(a => a.User)
.WithOne(b => b.Teacher)
.HasForeignKey<Teacher>(b => b.User_Id);
```

Рисунок 3.5 - Приклад зв'звку one-to-one

Після всіх цих маніпуляцій EF створює повноцінну реляційну БД на основі створених Entity та контексту ApllicationDBcontext(Залежить від назви вашого контексту) вигляд БД зображено на рисунку 3.6.

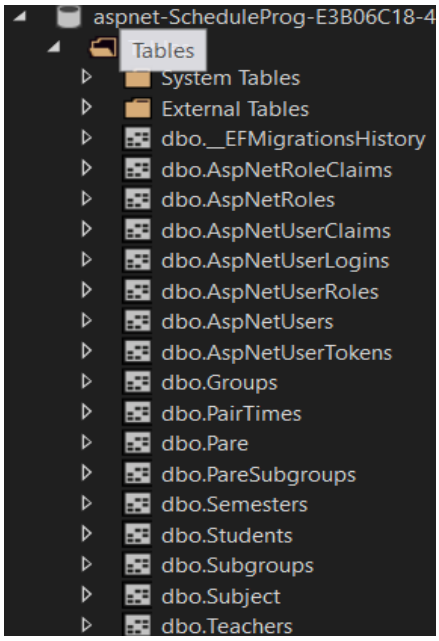


Рисунок 3.6 - Створена БД

3.2.2 Реалізація авторизації та автентифікації для користувача

Для реалізації авторизації та автентифікації для користувача використовується сервіс ASP. NET Identity буде використана так звана Cookie-Based Authentication

Автентифікація – це процес обміну обліковими даними для ідентифікації користувача. При аутентифікації на основі cookies унікальний ідентифікатор (файл cookie) створюється на стороні сервера та відправляється до браузера.

Коли користувач входить в веб-програму, браузер отримує файл cookie з сервера, зберігає його і відправляє з кожним наступним запитом, щоб сервер міг переконатися, що запити надходять від одного користувача. На рисунку 3.7 показано як додавався сервіс що відповідає за Cookie

```
builder.Services.ConfigureApplicationCookie(options =>
{
    // Cookie settings
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    I
    options.LoginPath = "/Identity/Account/Login";
    options.AccessDeniedPath = "/Identity/Account/AccessDenied";
    options.SlidingExpiration = true;
});
```

Рисунок 3.7- Імплементація Cookie в додаток

Також користувачу задаються певні параметри такі як мінімальна довжина пароля чи потрібні різні символи в паролі та ін. Конфігурація користувача представлена на рисунку 3.8

```
builder.Services.Configure<IdentityOptions>(options =>
{
    // Password settings.
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireUppercase = false;
    options.Password.RequiredLength = 6;
    options.User.RequireUniqueEmail = false;
});
```

Рисунок 3.8 - Конфігурація користувача

Далі розглянемо Авторизацію.

Авторизація — керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні (доступ до кімнати готелю за карткою), так і в галузі цифрових технологій (наприклад, автоматизована система контролю доступу) та ресурсів системи залежно від ідентифікатора і пароля

користувача або надання певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних.

В даному додатку авторизація виконана на основі ролей. Тобто для доступу до певної дії в контролері користувач має мати певну роль. Для цього також використовується сервіс Identity і на рисунку 3.9 представлені повне налаштування користувача. Переглянути весь код класу запуску можна в Додатку В.

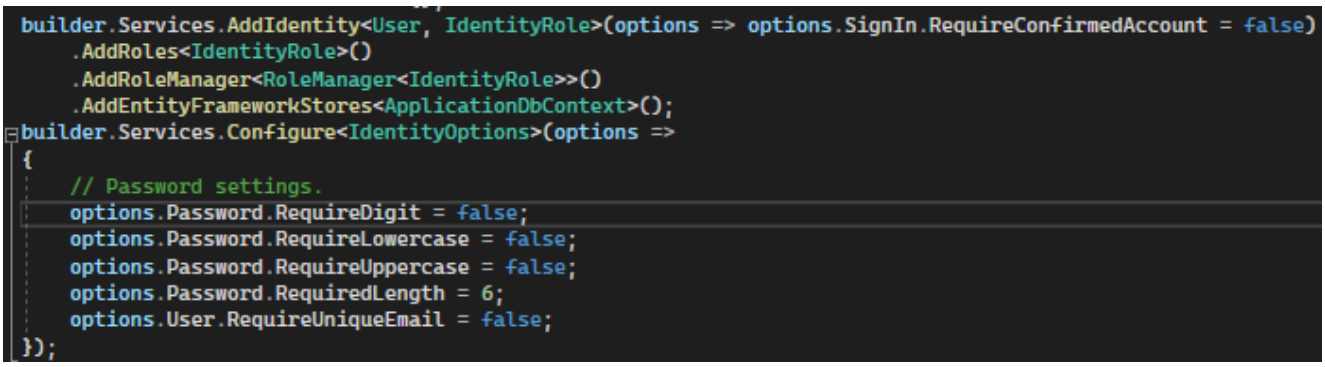
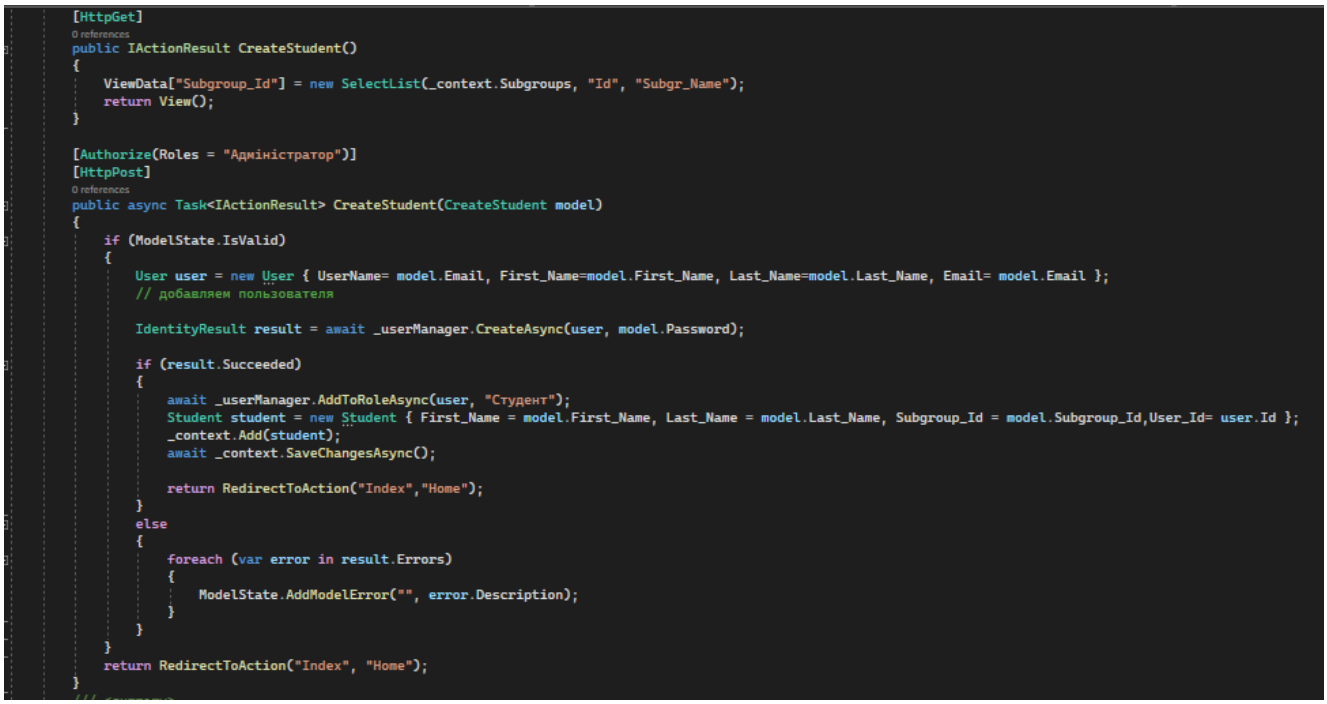


Рисунок 3.9 -Повне налаштування користувача

3.2.3 Налаштувати користувальницьких ролей.

У кожного користувача повинна бути роль для функціонування додаткана рисунку 3.10 представлено додавання для кожного користувача ролі. В Asp. net для того, щоб обмежити по ролях доступ для користувачів використовується атрибут [Authorize(Roles=”Rolename”)]. Цей атрибут дозволяє проводити дії над сторінкою або певною дією лише користувачам з роллю „Rolename” повний код можна переглянути в додатку А:



```
[HttpGet]
0 references
public IActionResult CreateStudent()
{
    ViewData["Subgroup_Id"] = new SelectList(_context.Subgroups, "Id", "Subgr_Name");
    return View();
}

[Authorize(Roles = "Адміністратор")]
[HttpPost]
0 references
public async Task<IActionResult> CreateStudent(CreateStudent model)
{
    if (ModelState.IsValid)
    {
        User user = new User { UserName= model.Email, First_Name=model.First_Name, Last_Name=model.Last_Name, Email= model.Email };
        // добавляем пользователя

        IdentityResult result = await _userManager.CreateAsync(user, model.Password);

        if (result.Succeeded)
        {
            await _userManager.AddToRoleAsync(user, "Студент");
            Student student = new Student { First_Name = model.First_Name, Last_Name = model.Last_Name, Subgroup_Id = model.Subgroup_Id,User_Id= user.Id };
            _context.Add(student);
            await _context.SaveChangesAsync();

            return RedirectToAction("Index","Home");
        }
        else
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }
        }
    }
    return RedirectToAction("Index", "Home");
}
```

Рисунок 3.10 - Обмеження доступу користувачу

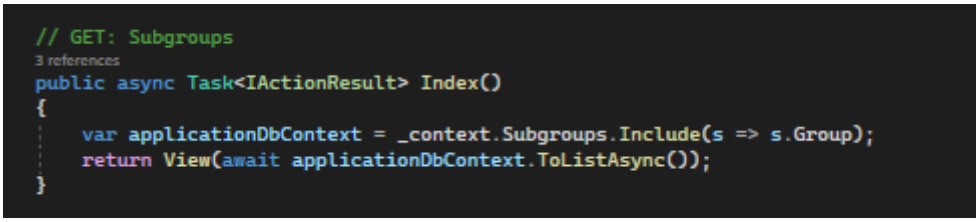
3.2.4 CRUD — Create, Read, Update, Delete

Для нормальної роботи додатка потрібно забезпечити основні дії над данними такі як видалення, перегляд, зміна та видалення даних.

Для кожної моделі повинні бути тою чи іншою мірою реалізовані всі або майже всі дії над її даними прикладом буде слугувати модель Subgroups.

Для всіх цих дій створений окремий контролер який має назву SubgroupsController (повний код представлений в додатку Е) він дозволяє проводити 4 основні операції над даними:

Read - Операція зображена на рисунку 3.11



```
// GET: Subgroups
3 references
public async Task<IActionResult> Index()
{
    var applicationDbContext = _context.Subgroups.Include(s => s.Group);
    return View(await applicationDbContext.ToListAsync());
}
```

Рисунок 3.11- дія Read представлена як Index

Create - Операція зображена на рисунку 3.12

```
public IActionResult Create()
{
    ViewData["Group_Id"] = new SelectList(_context.Groups, "Id", "Name_Of_Group");
    return View();
}

// POST: Subgroups/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Subgr_Name,Group_Id")] Subgroup subgroup)
{
    if (ModelState.IsValid)
    {
        _context.Add(subgroup);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["Group_Id"] = new SelectList(_context.Groups, "Id", "Id", subgroup.Group_Id);
    return View(subgroup);
}
```

Рисунок 3.12 - Дія create

Delete - Операція зображена на рисунку 3.13

```
// GET: Subgroups/Delete/5
0 references
public async Task<IActionResult> Delete(int? id)
{
    if (id == null || _context.Subgroups == null)
    {
        return NotFound();
    }

    var subgroup = await _context.Subgroups
        .Include(s => s.Group)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (subgroup == null)
    {
        return NotFound();
    }

    return View(subgroup);
}

// POST: Subgroups/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.Subgroups == null)
    {
        return Problem("Entity set 'ApplicationDbContext.Subgroups'  is null.");
    }
    var subgroup = await _context.Subgroups.FindAsync(id);
    if (subgroup != null)
    {
        _context.Subgroups.Remove(subgroup);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Рисунок 3.13- Дія Delete

Update(Edit) – операція зображена на рисунку 3.14

```
0 references
public async Task<IActionResult> Edit(int? id)
{
    if (id == null || _context.Subgroups == null)
    {
        return NotFound();
    }

    var subgroup = await _context.Subgroups.FindAsync(id);
    if (subgroup == null)
    {
        return NotFound();
    }
    ViewData["Group_Id"] = new SelectList(_context.Groups, "Id", "Id", subgroup.Group_Id);
    return View(subgroup);
}

// POST: Subgroups/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, [Bind("Id,Subgr_Name,Group_Id")] Subgroup subgroup)
{
    if (id != subgroup.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(subgroup);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!SubgroupExists(subgroup.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["Group_Id"] = new SelectList(_context.Groups, "Id", "Id", subgroup.Group_Id);
    return View(subgroup);
}
```

Рисунок 3.14 - Дія Update(Edit)

Всі ці дії тою чи іншою мірою реалізовано для всіх моделей. І оскільки серверна частина завершена можна переходити до клієнтської

3.3 Реалізація клієнтської частини.

Frontend (фронтенд або клієнтська частина програми) виконується у браузері користувача. Програма може складатися тільки з клієнтської частини, якщо не потрібно зберігати дані користувача довше однієї сесії. Це можуть бути, наприклад, фоторедактори чи прості іграшки.

Для реалізації клієнтської частини було використано влаштовані в розмітку Razor(детально описано в розділі 1) вона дозволяє імплементувати мову програмування C# напряму в HTML. Також певною мірою використовується JavaScript, проте це тільки влаштовані скрипти які потрібні для роботи Bootstrap та згенеровані при створенні проєкту за допомогою Visual Studio. Приклад розмітки Razor продемонстрований на рисунку 3.15.

```
@model IEnumerable<ScheduleProg.Models.Subject>

@{
    ViewData["Title"] = "Index";
}

<h1>Предмети</h1>


<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Discipline_Name)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Discipline_Name)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Змінити</a> |
                <a asp-action="Delete" asp-route-id="@item.Id">Видалити</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Рисунок 3.15 - вигляд сторінки Razor

Кожен веб-додаток має головну сторінку і даний не є виключенням, в додатку реалізовано дві головні сторінки одна для користувачів з роллю Адміністратор і інша сторінка для всіх інших користувачів. Приклад зовнішнього вигляду буде представлено на рисунку 3.16.

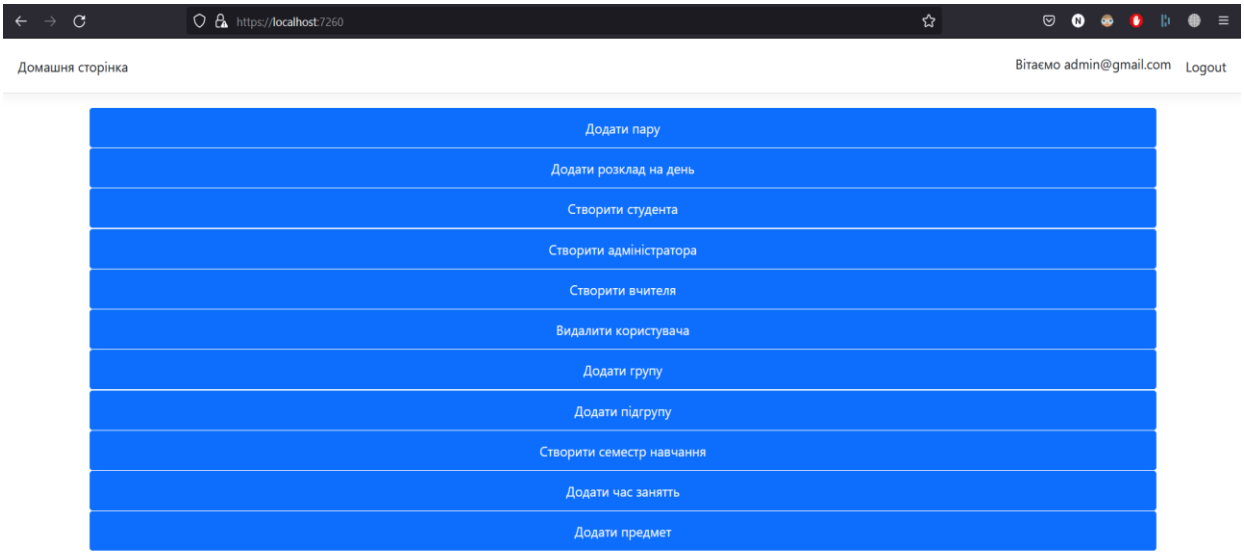


Рисунок 3.16 - зовнішній вигляд головної сторінки для адміністратора

Для того, щоб потрапити на сайти кожному користувачу потрібно авторизуватися. Вигляд екрана входу представлений на рисунку 3.17.

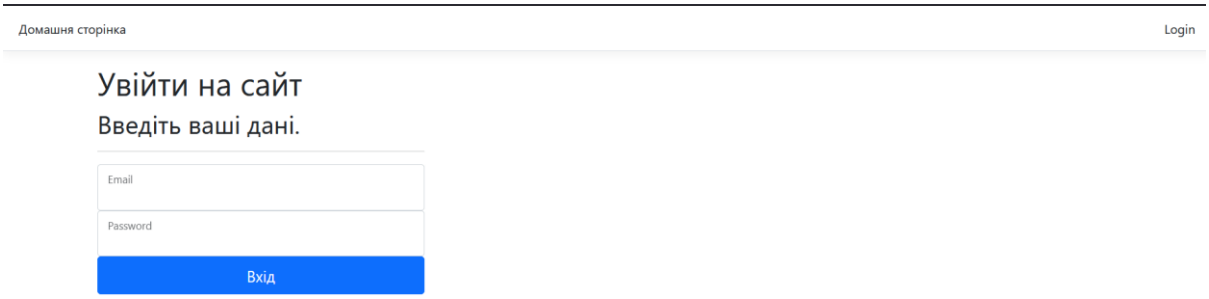


Рисунок 3.17 - сторінка входу

Адмін може заповнювати даними таблиці з допомогою певних дій. Наприклад на рисунку 3.18 зображено створення нового елемента для Pare

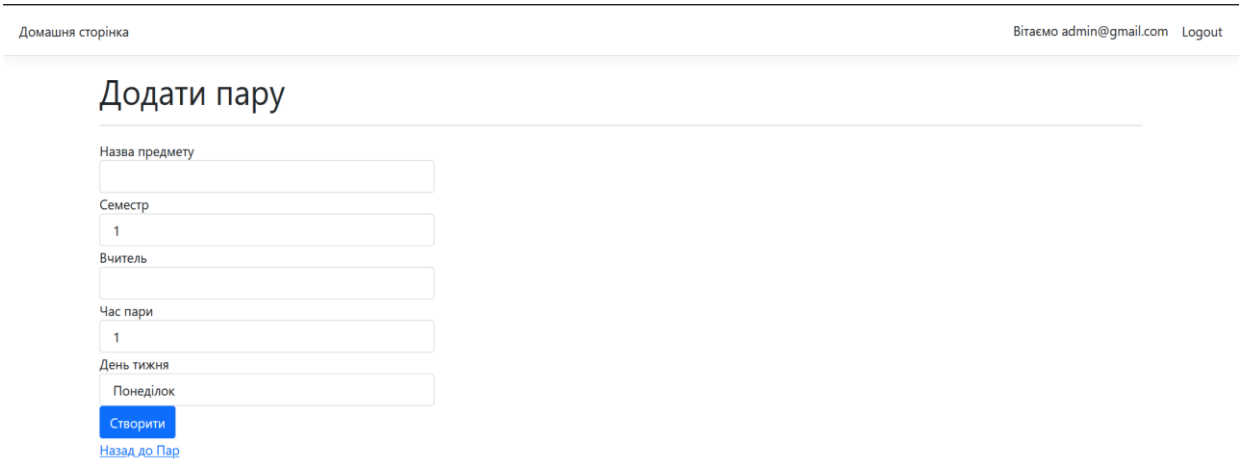


Рисунок 3.18 - Приклад створення нового екземпляру Pare

На рисунку 3.19 зображений перегляд адміном списку всіх пар

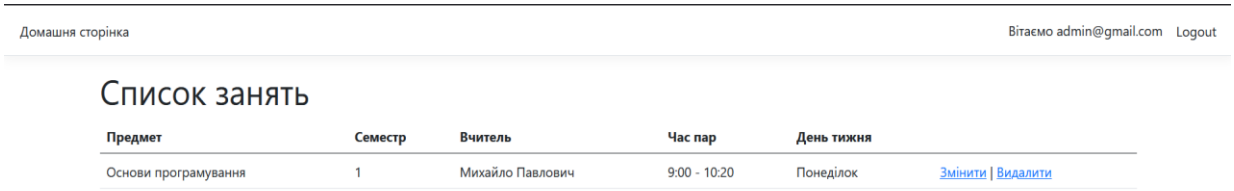


Рисунок 3.19 - Взаємодія з Pare перегляд даних

Інтерфейс для видалення інформації з таблиці Pare зображений на рис. 3.20

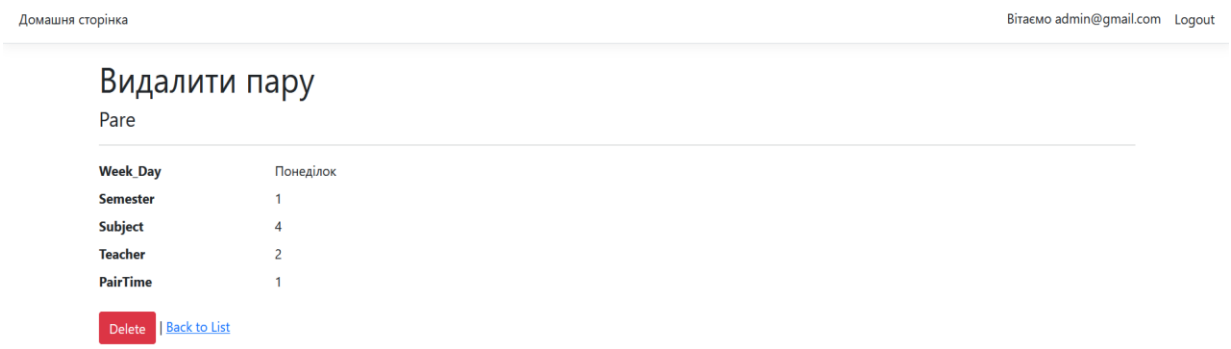


Рисунок 3. 20 - Інтерфейс видалення

Було розглянуто всі основні дії. Клієнтська частина працює справно.

ВИСНОВКИ

В ході виконання дипломної роботи було розроблено WEB-додаток для відображення розкладу занять, який дозволяє користувачу отримати інформацію про розклад занять для його групи, або розклад для вчителя.

Були проаналізовані технології для створення WEB-додатку і зважаючи тематику та технічне завдання були обрані відповідні засоби розробки для роботи додатка.

В ході виконання дипломної роботи були досягнуті наступні цілі:

- проведений аналіз предметної галузі
- реалізовано додаток на базі платформи ASP. NET
- програма наповнена даними з бази даних університету

Результатом роботи є повноцінний web-додаток для перегляду розкладу навчального закладу

Додаток відповідає технічному завданню і готовий до використання.

Задачі випускної кваліфікаційної роботи були успішно вирішені та було досягнуто її цілі.

ПЕРЕЛІК ПОСИЛАНЬ

1)Матеріал з Вікіпедії — вільної енциклопедії ASP. NET

URL:https://en. wikipedia. org/wiki/ASP. NET

2)ASP. NET документація

URL:https://docs. microsoft. com/en-us/aspnet/core/?view=aspnetcore-6. 0

3) Bootstrap 5 документація

URL:https://getbootstrap. com/docs/5. 0/getting-started/introduction/

4)ASP. NET MVC Succinctly Nick Harrison

URL:https://www. syncfusion. com/succinctly-free-ebooks/aspnet-mvc-succinctly/conceptual-overview

5)Introduction to Identity

URL:https://jakeydocs. readthedocs. io/en/latest/security/authentication/identity. html

6)Guide to ASP. NET Cookies

URL:https://www. codeproject. com/Articles/31914/Beginner-s-Guide-To-ASP-NET-Cookies

ДОДАТКИ

ДОДАТОК А

```
@model IEnumerable<ScheduleProg. Models. Pare>
@{
    ViewData["Title"] = "Index";
}

<style>

. week-day{

  width:5%;
  margin:0;
  padding:0;
}
. week-day-text{
     text-align: center;
  writing-mode: vertical-lr;
  text-orientation: upright;
}
. pare-body{
flex:9 0 0%;
}
. row
{
    --bs-gutter-x: 0;
    --bs-gutter-:0;

}
. centred-text{
justify-content: center;
text-align: center;
}

. schedule-body{
    width:90%;
}


</style>
```

```
@{
if(ViewBag. monday_item!=null
|| ViewBag. tuesday_item!=null
|| ViewBag. wednesday_item!=null
|| ViewBag. thursday_item!=null
|| ViewBag. friday_item!=null

){
<div class="container">

   <h1 class="centred-text">Розклад</h1>
   <div class="row footer">
   <div class="row week-day border centred-text" >День</div>
   <div class="row  schedule-body">
   <div class="row">
   <div class="col border centred-text ">
       Час


   </div>
    <div class="col pare-body border centred-text ">
        Розклад занятть
   </div>
    <div class="col border centred-text">
        Час
   </div>
   </div>



</div >
 <div class="row week-day centred-text border" >День</div>
</div>



<div class="row">
   <div class="row week-day week-day-text centred-text border" >Понеділок</div>
   <div class="row  schedule-body">
@foreach (var item in ViewBag. monday_item) {
   <div class="row">
   <div class="col border ">
```

```
            @item. PairTime. Begin_Time - @item. PairTime. End_Time


    </div>
     <div class="col pare-body border">
         @item. Semester. Semester_Name
         @item. Subject. Discipline_Name
              @item. Teacher. First_Name @item. Teacher. Last_Name
    </div>
     <div class="col border ">


            @item. PairTime. Begin_Time - @item. PairTime. End_Time


    </div>
    </div>



}</div >
 <div class="row week-day week-day-text centred-text border" >Понеділок</div>
</div>
<div class="row">
    <div class="row week-day week-day-text centred-text border" >Вівторок</div>
    <div class="row  schedule-body">
          @foreach (var item in ViewBag. tuesday_item) {

                <div class="row">
                   <div class="col border ">

                      @item. PairTime. Begin_Time - @item. PairTime. End_Time

                   </div>
                   <div class="col border pare-body ">
                      @item. Semester. Semester_Name
                      @item. Subject. Discipline_Name
                      @item. Teacher. First_Name @item. Teacher. Last_Name
                   </div>
                   <div class="col border pare-time ">

                      @item. PairTime. Begin_Time - @item. PairTime. End_Time

                   </div>
                </div>
```

```
}</div >
 <div class="row week-day week-day-text centred-text border" >Вівторок</div>
</div>

<div class="row">
   <div class="row week-day week-day-text centred-text border" >Середа</div>
   <div class="row  schedule-body">
         @foreach (var item in ViewBag. wednesday_item) {

               <div class="row">
                  <div class="col border ">

                     @item. PairTime. Begin_Time - @item. PairTime. End_Time

                  </div>
                  <div class="col border pare-body ">
                     @item. Semester. Semester_Name
                     @item. Subject. Discipline_Name
                     @item. Teacher. First_Name @item. Teacher. Last_Name
                  </div>
                  <div class="col border pare-time ">

                     @item. PairTime. Begin_Time - @item. PairTime. End_Time

                  </div>
               </div>




}</div >
 <div class="row week-day week-day-text centred-text border" >Середа</div>
</div>


<div class="row">
   <div class="row week-day week-day-text centred-text border" >Четвер</div>
   <div class="row  schedule-body">
         @foreach (var item in ViewBag. thursday_item) {

               <div class="row">
                  <div class="col border ">
```

```
                @item. PairTime. Begin_Time - @item. PairTime. End_Time

            </div>
            <div class="col border pare-body ">
                @item. Semester. Semester_Name
                @item. Subject. Discipline_Name
                @item. Teacher. First_Name @item. Teacher. Last_Name
            </div>
            <div class="col border pare-time ">

                @item. PairTime. Begin_Time - @item. PairTime. End_Time

            </div>
        </div>




}</div >
 <div class="row week-day week-day-text centred-text border" >Четвер</div>
</div>


<div class="row">
   <div class="row week-day week-day-text centred-text border" >П'ятниця</div>
   <div class="row  schedule-body">
        @foreach (var item in ViewBag. friday_item) {

            <div class="row">
                <div class="col border ">

                    @item. PairTime. Begin_Time - @item. PairTime. End_Time

                </div>
                <div class="col border pare-body ">
                    @item. Semester. Semester_Name
                    @item. Subject. Discipline_Name
                    @item. Teacher. First_Name @item. Teacher. Last_Name
                </div>
                <div class="col border pare-time ">

                    @item. PairTime. Begin_Time - @item. PairTime. End_Time
```

```
                    </div>
                </div>



}</div >
 <div class="row week-day week-day-text centred-text border" >П'ятниця</div>
</div>
</div>
}

else{
<h1>Поки що розклад не готовий</h1>

}
}
```

ДОДАТОК Б

```
using System. Threading. Tasks;
using Microsoft. AspNetCore. Mvc;
using ScheduleProg. Models;
using Microsoft. AspNetCore. Identity;
using Microsoft. AspNetCore. Mvc. Rendering;
using ScheduleProg. Data;
using Microsoft. AspNetCore. Authentication;
using Microsoft. AspNetCore. Authorization;
using Microsoft. EntityFrameworkCore;

namespace ScheduleProg. Controllers
{
   [Authorize]
   public class AccountController : Controller
   {
      private readonly UserManager<User> _userManager;
      private readonly SignInManager<User> _signInManager;
      private readonly ApplicationDbContext _context;

      public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, ApplicationDbContext
context)
      {
         _userManager = userManager;
         _signInManager = signInManager;
         _context=context;
      }

      public IActionResult Index()
      {


         ViewData["Subgroup_Id"] = new SelectList(_context. Subgroups, "Id", "Subgr_Name");
         return View();
      }

      [Authorize(Roles = "Адміністратор")]

      [HttpGet]
      public IActionResult CreateStudent()
      {
         ViewData["Subgroup_Id"] = new SelectList(_context. Subgroups, "Id", "Subgr_Name");
```

```
        return View();
    }

    [Authorize(Roles = "Адміністратор")]
    [HttpPost]
    public async Task<IActionResult> CreateStudent(CreateStudent model)
    {
        if (ModelState. IsValid)
        {
            User user = new User { UserName= model. Email, First_Name=model. First_Name, Last_Name=model. Last_Name,
Email= model. Email };
            // добавляем пользователя

            IdentityResult result = await _userManager. CreateAsync(user, model. Password);

            if (result. Succeeded)
            {
                await _userManager. AddToRoleAsync(user, "Студент");
                Student student = new Student { First_Name = model. First_Name, Last_Name = model. Last_Name, Subgroup_Id =
model. Subgroup_Id, User_Id= user. Id };
                _context. Add(student);
                await _context. SaveChangesAsync();

                return RedirectToAction("Index", "Home");
            }
            else
            {
                foreach (var error in result. Errors)
                {
                    ModelState. AddModelError("", error. Description);
                }
            }
        }
        return RedirectToAction("Index", "Home");
    }
    /// <summary>

    // GET: PairTimes/Delete/5

    [Authorize(Roles = "Адміністратор")]
    public IActionResult DeleteUser()
    {
```

```
    ViewData["User_id"] = new SelectList(_context. Users, "Id", "Email");


    return View();
}

// POST: PairTimes/Delete/5
[HttpPost, ActionName("DeleteUser")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteUser(string User_id)
{
    var user = await _userManager. FindByIdAsync(User_id);
    var Student = await _context. Students. FirstOrDefaultAsync(p=>p. User_Id==User_id);

    var Teacher= await _context. Teachers. FirstOrDefaultAsync(p => p. User_Id == User_id);
    if (Student != null)
    {
        _context. Students. Remove(Student);
        await _userManager. DeleteAsync(user);
        await _context. SaveChangesAsync();
    }
    if (Teacher != null)
    {
        _context. Teachers. Remove(Teacher);
        await _userManager. DeleteAsync(user);
        await _context. SaveChangesAsync();
    }
    else {
        await _userManager. DeleteAsync(user);
        await _context. SaveChangesAsync();
    }
    return View("Home/AdminPage");
}



[Authorize(Roles = "Адміністратор")]
[HttpGet]
public IActionResult CreateTeacher()
{
    return View();
}
[HttpPost]
public async Task<IActionResult> CreateTeacher(CreateTeacher model)
```

```
    {
        if (ModelState. IsValid)
        {
            User user = new User { UserName = model. Email, First_Name = model. First_Name, Last_Name = model.
Last_Name, Email = model. Email };
            // добавляем пользователя

            IdentityResult result = await _userManager. CreateAsync(user, model. Password);

            if (result. Succeeded)
            {
                await _userManager. AddToRoleAsync(user, "Вчитель");
                Teacher teacher= new Teacher{ First_Name = model. First_Name, Last_Name = model. Last_Name, User_Id = user.
Id };
                _context. Add(teacher);
                await _context. SaveChangesAsync();

                return RedirectToAction("Index", "Home");
            }
            else
            {
                foreach (var error in result. Errors)
                {
                    ModelState. AddModelError("", error. Description);
                }
            }
        }
        return RedirectToAction("Index", "Home");
    }


    [Authorize(Roles = "Адміністратор")]
    public IActionResult CreateAdministrator()
    {
        return View();
    }

    [Authorize(Roles = "Адміністратор")]
    [HttpPost]
    public async Task<IActionResult> CreateAdministrator(CreateAdministrator model)
    {
        if (ModelState. IsValid)
        {
```

```
            User user = new User { UserName = model. Email, First_Name = model. First_Name, Last_Name = model.
Last_Name, Email = model. Email };
            // добавляем пользователя

            IdentityResult result = await _userManager. CreateAsync(user, model. Password);

            if (result. Succeeded)
            {
               await _userManager. AddToRoleAsync(user, "Адміністратор");
               await _context. SaveChangesAsync();

               return RedirectToAction("Index", "Home");
            }
            else
            {
               foreach (var error in result. Errors)
               {
                  ModelState. AddModelError("", error. Description);
               }
            }
         }
         return RedirectToAction("Index", "Home");
      }



   }
}
```

ДОДАТОК В

```
using Microsoft. AspNetCore. Identity;
using Microsoft. EntityFrameworkCore;
using Microsoft. Extensions. DependencyInjection. Extensions;
using ScheduleProg;
using ScheduleProg. Data;
using ScheduleProg. Models;
```

```
var builder = WebApplication. CreateBuilder(args);

var connectionString = builder. Configuration. GetConnectionString("DefaultConnection");
builder. Services. AddDbContext<ApplicationDbContext>(options =>
    options. UseSqlServer(connectionString));;



builder. Services. AddDbContext<ApplicationDbContext>(options =>
    options. UseSqlServer(connectionString));;


// Add services to the container.
builder. Services. AddControllersWithViews();
builder. Services. AddAuthentication();
builder. Services. AddRazorPages();
builder. Services. AddDbContext<ApplicationDbContext>(options =>
    options. UseSqlServer(connectionString));
builder. Services. AddDatabaseDeveloperPageExceptionFilter();
builder. Services. AddHealthChecks();
builder. Services. AddIdentity<User, IdentityRole>(options => options. SignIn. RequireConfirmedAccount = false)
   . AddRoles<IdentityRole>()
   . AddRoleManager<RoleManager<IdentityRole>>()
   . AddEntityFrameworkStores<ApplicationDbContext>();
builder. Services. Configure<IdentityOptions>(options =>
{
    // Password settings.
    options. Password. RequireDigit = false;
    options. Password. RequireLowercase = false;
    options. Password. RequireUppercase = false;
    options. Password. RequiredLength = 6;
    options. User. RequireUniqueEmail = false;
});
builder. Services. TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();
builder. Services. AddHttpContextAccessor();
builder. Services. AddControllersWithViews();
builder. Services. ConfigureApplicationCookie(options =>
{
    // Cookie settings
    options. Cookie. HttpOnly = true;
    options. ExpireTimeSpan = TimeSpan. FromMinutes(60);

    options. LoginPath = "/Identity/Account/Login";
```

```
    options. AccessDeniedPath = "/Identity/Account/AccessDenied";
    options. SlidingExpiration = true;
});
var app = builder. Build();
app. UseAuthentication();


// Configure the HTTP request pipeline.
if (app. Environment. IsDevelopment())
{
    app. UseMigrationsEndPoint();
}
else
{
    app. UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka. ms/aspnetcore-
hsts.
    app. UseHsts();
}

using (var scope = app. Services. CreateScope())
{
    var services = scope. ServiceProvider;
    try
    {
        var userManager = services. GetRequiredService<UserManager<User>>();
        var rolesManager = services. GetRequiredService<RoleManager<IdentityRole>>();
        await RoleInitializer. InitializeAsync(userManager, rolesManager);
    }
    catch (Exception ex)
    {
        var logger = services. GetRequiredService<ILogger<Program>>();
        logger. LogError(ex, "An error occurred while seeding the database. ");
    }
}

app. UseHttpsRedirection();
app. UseStaticFiles();

app. UseRouting();


app. UseAuthorization();
app. MapControllerRoute(
```

```
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app. MapRazorPages();

app. Run();
```

ДОДАТОК Г

```
using Microsoft. AspNetCore. Identity;
using Microsoft. AspNetCore. Identity. EntityFrameworkCore;
using Microsoft. EntityFrameworkCore;
using ScheduleProg. Models;

namespace ScheduleProg. Data
{

   public class ApplicationDbContext : IdentityDbContext<User>
   {
      public DbSet<Group> Groups { get; set; }
      public DbSet<PairTime> PairTimes { get; set; }
      public DbSet<Pare> Schedules { get; set; }
      public DbSet<Semester> Semesters { get; set; }

      public DbSet<Student> Students { get; set; }

      public DbSet<Teacher> Teachers { get; set; }

      public DbSet<Subgroup> Subgroups { get; set; }

      public DbSet<PareSubgroup> PareSubgroups { get; set; }


      public DbSet<Pare> Pares { get; set; }
      public DbSet<ScheduleProg. Models. Subject> Subject { get; set; }

      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
         : base(options)
      {
      }









      protected override void OnModelCreating(ModelBuilder builder)
      {
```

```
  /*b*/
  base. OnModelCreating(builder);
  builder. Entity<Semester>()
 . HasMany<Pare>(semester => semester. Pares)
 . WithOne(pare => pare. Semester)
 . HasForeignKey(pare => pare. Semester_Id);
  /*e*/
  /*  builder. Entity<Group>()
  . HasMany<Pare>(group => group. Pares)
  . WithOne(pare => pare. Group)
  . HasForeignKey(pare => pare. Group_Id);*/

  /*b*/
  builder. Entity<Subject>()
. HasMany<Pare>(s => s. Pares)
. WithOne(p => p. Subject)
. HasForeignKey(p => p. Subject_Id);
  /*e*/

  /*b*/
  builder. Entity<Teacher>()
. HasMany<Pare>(teacher => teacher. Pares)
. WithOne(pare => pare. Teacher)
. HasForeignKey(pare => pare. Teacher_Id);
  /*e*/

  /*b*/
  builder. Entity<PairTime>()
. HasMany<Pare>(pairtime => pairtime. Pares)
. WithOne(pare => pare. PairTime)
. HasForeignKey(pare => pare. Pair_Time_Id);
  /*e*/

  /*b*/
  builder. Entity<PareSubgroup>()
. HasOne<Pare>(ps => ps. Pare)
. WithMany(p=> p. PareSubgroups)
. HasForeignKey(ps => ps. Pare_Id);
  /*e*/

  /*Many-to-Many */
  builder. Entity<PareSubgroup>(). HasKey(ps => new { ps. Pare_Id, ps. Subgroup_Id });
 builder. Entity<PareSubgroup>()
. HasOne<Subgroup>(ps => ps. Subgroup)
```

```
        . WithMany(s => s. PareSubgroups)
        . HasForeignKey(ps => ps. Subgroup_Id);


          /*e*/

          /*b*/
          builder. Entity<Group>()
          . HasMany<Subgroup>(s=>s. Subgroups)
          . WithOne(g=>g. Group)
          . HasForeignKey(g=>g. Group_Id);
          /*e*/

          /*b*/
          builder. Entity<Subgroup>()
          . HasMany<Student>(s => s. Students)
          . WithOne(g => g. Subgroup)
          . HasForeignKey(g => g. Subgroup_Id);

          builder. Entity<Student>()
          . HasOne(a => a. User)
          . WithOne(b => b. Student)
          . HasForeignKey<Student>(b => b. User_Id);

          builder. Entity<Teacher>()
          . HasOne(a => a. User)
          . WithOne(b => b. Teacher)
          . HasForeignKey<Teacher>(b => b. User_Id);









      }


  }
}
```

ДОДАТОК Д

```
// Licensed to the. NET Foundation under one or more agreements.
// The. NET Foundation licenses this file to you under the MIT license.
#nullable disable

using System;
using System. Collections. Generic;
using System. ComponentModel. DataAnnotations;
using System. Linq;
using System. Threading. Tasks;
using Microsoft. AspNetCore. Authorization;
using Microsoft. AspNetCore. Authentication;
using Microsoft. AspNetCore. Identity;
using Microsoft. AspNetCore. Identity. UI. Services;
using Microsoft. AspNetCore. Mvc;
using Microsoft. AspNetCore. Mvc. RazorPages;
using Microsoft. Extensions. Logging;
using ScheduleProg. Models;

namespace ScheduleProg. Areas. Identity. Pages. Account
{
   public class LoginModel : PageModel
   {
      private readonly SignInManager<User> _signInManager;
      private readonly ILogger<LoginModel> _logger;

      public LoginModel(SignInManager<User> signInManager, ILogger<LoginModel> logger)
      {
         _signInManager = signInManager;
         _logger = logger;
      }

      /// <summary>
      ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
      ///     directly from your code. This API may change or be removed in future releases.
      /// </summary>
      [BindProperty]
      public InputModel Input { get; set; }

      /// <summary>
```

```
    ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    public IList<AuthenticationScheme> ExternalLogins { get; set; }

    /// <summary>
    ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    public string ReturnUrl { get; set; }

    /// <summary>
    ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    [TempData]
    public string ErrorMessage { get; set; }

    /// <summary>
    ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    public class InputModel
    {
        /// <summary>
        ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        /// <summary>
        ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
        [Required]
        [DataType(DataType. Password)]
        public string Password { get; set; }

        /// <summary>
        ///     This API supports the ASP. NET Core Identity default UI infrastructure and is not intended to be used
        ///     directly from your code. This API may change or be removed in future releases.
        /// </summary>
```

```
    }

    public async Task OnGetAsync(string returnUrl = null)
    {


        if (!string. IsNullOrEmpty(ErrorMessage))
        {
            ModelState. AddModelError(string. Empty, ErrorMessage);
        }

        returnUrl ??= Url. Content("~/");

        // Clear the existing external cookie to ensure a clean login process
        await HttpContext. SignOutAsync(IdentityConstants. ExternalScheme);

        ExternalLogins = (await _signInManager. GetExternalAuthenticationSchemesAsync()). ToList();

        ReturnUrl = returnUrl;
        if (User. Identity. IsAuthenticated)
        {



            Redirect("~/Pares/Index");
        }
    }

    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        if (User. Identity. IsAuthenticated)
        {

            return Redirect("~/Pares/Index");
        }
        returnUrl ??= Url. Content("~/");

        ExternalLogins = (await _signInManager. GetExternalAuthenticationSchemesAsync()). ToList();

        if (ModelState. IsValid)
        {
            // This doesn't count login failures towards account lockout
            // To enable password failures to trigger account lockout, set lockoutOnFailure: true
            var result = await _signInManager. PasswordSignInAsync(Input. Email, Input. Password, false, lockoutOnFailure:
false);
```

```
            if (result. Succeeded)
            {
                _logger. LogInformation("User logged in. ");
                return LocalRedirect(returnUrl);
            }
            if (result. RequiresTwoFactor)
            {
                return RedirectToPage(". /LoginWith2fa", new { ReturnUrl = returnUrl });
            }
            if (result. IsLockedOut)
            {
                _logger. LogWarning("User account locked out. ");
                return RedirectToPage(". /Lockout");
            }
            else
            {
                ModelState. AddModelError(string. Empty, "Invalid login attempt. ");
                return Page();
            }
        }

        // If we got this far, something failed, redisplay form
        return Page();
    }
  }
}
```

ДОДАТОК Е

```
using System;
using System. Collections. Generic;
using System. Linq;
using System. Threading. Tasks;
using Microsoft. AspNetCore. Authorization;
using Microsoft. AspNetCore. Mvc;
using Microsoft. AspNetCore. Mvc. Rendering;
using Microsoft. EntityFrameworkCore;
using ScheduleProg. Data;
using ScheduleProg. Models;

namespace ScheduleProg. Controllers
{
    [Authorize(Roles = "Адміністратор")]
    public class SubgroupsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public SubgroupsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Subgroups
        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context. Subgroups. Include(s => s. Group);
            return View(await applicationDbContext. ToListAsync());
        }

        // GET: Subgroups/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context. Subgroups == null)
            {
                return NotFound();
            }

            var subgroup = await _context. Subgroups
              . Include(s => s. Group)
```

```
        . FirstOrDefaultAsync(m => m. Id == id);
    if (subgroup == null)
    {
        return NotFound();
    }

    return View(subgroup);
}

// GET: Subgroups/Create
public IActionResult Create()
{
    ViewData["Group_Id"] = new SelectList(_context. Groups, "Id", "Name_Of_Group");
    return View();
}

// POST: Subgroups/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go. microsoft. com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id, Subgr_Name, Group_Id")] Subgroup subgroup)
{
    if (ModelState. IsValid)
    {
        _context. Add(subgroup);
        await _context. SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["Group_Id"] = new SelectList(_context. Groups, "Id", "Id", subgroup. Group_Id);
    return View(subgroup);
}

// GET: Subgroups/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null || _context. Subgroups == null)
    {
        return NotFound();
    }

    var subgroup = await _context. Subgroups. FindAsync(id);
    if (subgroup == null)
    {
```

```
        return NotFound();
    }
    ViewData["Group_Id"] = new SelectList(_context. Groups, "Id", "Id", subgroup. Group_Id);
    return View(subgroup);
}

// POST: Subgroups/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go. microsoft. com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id, Subgr_Name, Group_Id")] Subgroup subgroup)
{
    if (id != subgroup. Id)
    {
        return NotFound();
    }

    if (ModelState. IsValid)
    {
        try
        {
            _context. Update(subgroup);
            await _context. SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!SubgroupExists(subgroup. Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["Group_Id"] = new SelectList(_context. Groups, "Id", "Id", subgroup. Group_Id);
    return View(subgroup);
}

// GET: Subgroups/Delete/5
public async Task<IActionResult> Delete(int? id)
```

```
        {
            if (id == null || _context. Subgroups == null)
            {
                return NotFound();
            }

            var subgroup = await _context. Subgroups
                . Include(s => s. Group)
                . FirstOrDefaultAsync(m => m. Id == id);
            if (subgroup == null)
            {
                return NotFound();
            }

            return View(subgroup);
        }

        // POST: Subgroups/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context. Subgroups == null)
            {
                return Problem("Entity set 'ApplicationDbContext. Subgroups'  is null. ");
            }
            var subgroup = await _context. Subgroups. FindAsync(id);
            if (subgroup != null)
            {
                _context. Subgroups. Remove(subgroup);
            }

            await _context. SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool SubgroupExists(int id)
        {
          return (_context. Subgroups?. Any(e => e. Id == id)). GetValueOrDefault();
        }
    }
}
```