МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МЕЛЬНИК ВЛАДИСЛАВ АНДРІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА ВЕБДОДАТКУ ДЛЯ УПРАВЛІННЯ РОЗКЛАДОМ ЗАНЯТЬ ІЗ МОЖЛИВІСТЮ ПЕРСОНАЛІЗОВАНИХ НОТАТОК

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Наталія ВЕСЕЛОВСЬКА, професор кафедри
інформаційних технологій,
д. т. н., професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Бакалаврська робота присвячена проєктуванню та розробці вебдодатку для ефективного управління розкладом занять з інтегрованою системою персоналізованих нотаток. Проведено аналіз існуючих рішень, визначено функціональні та нефункціональні вимоги, обґрунтовано вибір архітектури MVC та технологічного стеку ASP.NET Core.

Розроблено структуру бази даних, реалізовано модулі автентифікації користувачів, управління розкладом та нотатками, функції пошуку та фільтрації. Створено адаптивний інтерфейс користувача з використанням Bootstrap.

Проведено функціональне тестування та економічний аналіз, який підтвердив високу ефективність проєкту. Розроблено рекомендації з охорони праці. Веб-додаток готовий до впровадження в навчальних закладах.

Ключові слова: веб-додаток, управління розкладом, персоналізовані нотатки, ASP.NET Core MVC, Entity Framework Core, SQLite, автентифікація користувачів, адаптивний дизайн.

ABSTRACT

The thesis is dedicated to the design and development of a web application for effective schedule management with an integrated system of personalized notes. An analysis of existing solutions was conducted, functional and non-functional requirements were defined, and the choice of MVC architecture and ASP.NET Core technology stack was justified.

A database structure was developed, and modules for user authentication, schedule and notes management, search and filtering functions were implemented. An adaptive user interface was created using Bootstrap.

Functional testing and economic analysis were conducted, confirming the high efficiency of the project. Occupational safety recommendations were developed. The web application is ready for implementation in educational institutions.

Keywords: web application, schedule management, personalized notes, ASP.NET Core MVC, Entity Framework Core, SQLite, user authentication, responsive design.

ЗМІСТ

ВСТУП

У сучасному світі, де темп життя постійно прискорюється, а обсяг інформації невпинно зростає, ефективне управління часом стає критично важливим фактором успіху як у навчальній, так і в професійній діяльності. Особливо гостро ця проблема стоїть перед студентами та викладачами навчальних закладів, які щоденно стикаються з необхідністю координації численних занять, завдань, дедлайнів та додаткової інформації. Традиційні методи ведення розкладу, такі як паперові щоденники або прості електронні таблиці, вже не відповідають вимогам сучасного динамічного освітнього середовища.

Актуальність теми дипломної роботи обумовлена кількома ключовими факторами. По-перше, цифрова трансформація освітньої галузі вимагає впровадження сучасних технологічних рішень для оптимізації навчального процесу. По-друге, існуючі програмні рішення часто є або надто складними та дорогими для індивідуального використання, або занадто простими та не забезпечують необхідної функціональності для ведення персоналізованих нотаток, інтегрованих з розкладом. По-третє, пандемія COVID-19 та перехід до змішаних форм навчання підкреслили важливість наявності гнучких інструментів для організації навчального процесу, доступних з будь-якого пристрою та місця.

Метою дипломної роботи є проєктування та розробка веб-додатку для ефективного управління розкладом занять з інтегрованою системою персоналізованих нотаток, що забезпечить підвищення продуктивності навчального процесу для студентів та викладачів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести аналіз предметної області та дослідити існуючі підходи до управління розкладом занять і ведення навчальних нотаток.
2. Визначити функціональні та нефункціональні вимоги до веб-додатку на основі потреб цільової аудиторії.
3. Обґрунтувати вибір архітектурного рішення та технологічного стеку для реалізації проєкту.
4. Спроєктувати структуру бази даних та основні модулі системи з урахуванням принципів масштабованості та безпеки.
5. Розробити інтуїтивно зрозумілий інтерфейс користувача з адаптивним дизайном для різних пристроїв.
6. Реалізувати ключові функціональні модулі веб-додатку, включаючи систему автентифікації, управління розкладом, персоналізовані нотатки та функції пошуку/фільтрації.
7. Провести функціональне тестування розробленого додатку та усунути виявлені недоліки.
8. Виконати економічний аналіз проєкту та обґрунтувати його доцільність.
9. Розробити рекомендації з охорони праці для користувачів та розробників подібних систем.

Об'єктом дослідження є процеси організації та управління навчальним розкладом у сучасних освітніх установах.

Предметом дослідження є методи та технології розробки веб-додатків для автоматизації управління розкладом занять з інтегрованою системою персоналізованих нотаток.

Наукова новизна роботи полягає у:

- розробці комплексного підходу до інтеграції функцій управління розкладом та ведення персоналізованих нотаток в єдиному веб-середовищі;
- створенні оптимізованої моделі даних, що забезпечує ефективний зв'язок між заняттями та пов'язаною з ними інформацією;

- впровадженні адаптивних алгоритмів фільтрації та пошуку, що враховують специфіку навчального процесу.

Практична значущість роботи визначається можливістю безпосереднього впровадження розробленого веб-додатку в навчальних закладах різного рівня для підвищення ефективності організації навчального процесу. Додаток може використовуватися як студентами для планування навчальної діяльності та ведення конспектів, так і викладачами для організації викладацької роботи.

Розроблене рішення є економічно доступним завдяки використанню безкоштовних технологій та може бути легко адаптоване під специфічні потреби конкретного навчального закладу або індивідуального користувача.

Результати роботи можуть бути використані як основа для подальшого розвитку та вдосконалення системи, додавання нових функціональних можливостей, таких як інтеграція з календарними сервісами, автоматичні нагадування, колективна робота над розкладом тощо.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ

1.1. Призначення і загальні параметри проєктованого вебдодатку

У сучасному динамічному світі ефективне управління часом є ключовим фактором успіху як для студентів, так і для викладачів та будь-яких інших осіб, чия діяльність пов'язана з розкладом та плануванням. Традиційні методи ведення розкладу, такі як паперові щоденники або прості таблиці, часто не забезпечують необхідної гнучкості, доступності та можливостей для персоналізації. Саме тому розробка спеціалізованих програмних рішень для управління розкладом є актуальним завданням.

Основне призначення проєктованого веб-додатку надання користувачам зручного та інтуїтивно зрозумілого інструменту для створення, перегляду та управління розкладом занять (або будь-яких інших періодичних подій) з можливістю додавання персоналізованих нотаток до кожної події. Додаток покликаний допомогти користувачам систематизувати свій час, не пропускати важливі події та мати швидкий доступ до необхідної інформації, пов'язаної із заняттями.

Головні цілі розробки веб-додатку:

1. Централізація інформації про розклад. Забезпечити єдине місце для зберігання та доступу до актуального розкладу занять.
2. Підвищення ефективності планування. Дозволити користувачам легко додавати, редагувати та видаляти заняття, а також візуалізувати свій графік у зручному форматі.
3. Персоналізація досвіду. Надати можливість кожному користувачеві вести власний розклад та додавати до занять приватні нотатки, що містять важливу інформацію, завдання, нагадування тощо.

4. Забезпечення доступності. Як веб-додаток, система має бути доступною з будь-якого пристрою, що має доступ до Інтернету та веб-браузер.
5. Інтуїтивно зрозумілий інтерфейс. Створити простий та зручний інтерфейс користувача, який не потребуватиме тривалого навчання для ефективного використання.

Загальні параметри та характеристики проєктованого веб-додатку^

- Тип додатку. Веб-додаток, реалізований за архітектурою MVC (Model-View-Controller).
- Технологічна платформа. ASP.NET Core, мова програмування C#.
- База даних: SQLite для зберігання даних користувачів, розкладів та нотаток (як було реалізовано в процесі розробки).
- Автентифікація та Авторизація. Система індивідуальних облікових записів користувачів на базі ASP.NET Core Identity, що забезпечує персоналізований доступ до даних. Кожен користувач має доступ лише до власного розкладу та нотаток.
- Основний функціонал:
  - Реєстрація та вхід користувачів.
  - Створення, перегляд, редагування та видалення занять у розкладі.
  - Можливість вказувати для заняття: назву предмету, тип заняття (лекція, практика тощо), викладача, аудиторію, час початку та закінчення, колір для візуального розрізнення.
  - Створення, перегляд, редагування та видалення персоналізованих нотаток, прив'язаних до конкретних занять.
  - Відображення розкладу у вигляді списку/таблиці.
  - Можливість пошуку та фільтрації занять за різними критеріями (назва, тип, дата).
- Інтерфейс користувача. Адаптивний дизайн, реалізований з використанням HTML, CSS та бібліотеки Bootstrap для забезпечення коректного відображення на різних пристроях (десктопи, планшети, мобільні телефони).

- Масштабованість та розширюваність. Архітектура додатку передбачає можливість подальшого розширення функціоналу (наприклад, додавання сповіщень, інтеграції з календарями, розширених можливостей звітності тощо).

Проєктований веб-додаток спрямований на вирішення реальних потреб користувачів в організації свого навчального або робочого процесу, роблячи його більш структурованим та ефективним завдяки поєднанню функцій управління розкладом та персоналізованих нотаток.

1.2. Огляд та аналіз існуючих підходів і програмних рішень для управління розкладом занять і нотатками

Для ефективного управління розкладом занять та пов'язаними з ними нотатками існує низка підходів та програмних рішень, які можна умовно розділити на кілька категорій [3].

1. Традиційні (паперові) методи:
   - Паперові щоденники, планери, блокноти. Цей підхід є найдавнішим і все ще використовується деякими людьми завдяки своїй простоті та тактильності. Однак він має суттєві недоліки: обмежений простір, відсутність автоматичних нагадувань, складність редагування та внесення змін, ризик втрати даних, відсутність можливості пошуку та синхронізації між пристроями. Персоналізовані нотатки ведуться вручну, що також обмежує їхню функціональність.
2. Електронні таблиці та текстові редактори:
   - Microsoft Excel, Google Sheets, Microsoft Word, Google Docs. Багато користувачів створюють власні шаблони розкладів у табличних процесорах або ведуть нотатки у текстових документах. Це більш гнучкий підхід порівняно з паперовим, що дозволяє легко редагувати

дані та зберігати їх в електронному вигляді. Деякі табличні процесори дозволяють базову фільтрацію.

- Недоліки. Відсутність спеціалізованих функцій для управління розкладом (нагадування, візуалізація календаря, обробка повторюваних подій), обмежені можливості для інтеграції нотаток безпосередньо з подіями розкладу, а також необхідність ручного структурування даних.

3. Загальновживані календарні додатки:

- Google Calendar, Outlook Calendar, Apple Calendar. Ці додатки є потужними інструментами для управління подіями, включаючи можливість створення повторюваних зустрічей, встановлення нагадувань, спільного доступу до календарів та синхронізації між різними пристроями. Багато з них дозволяють додавати короткі описи або нотатки до подій.
- Недоліки. Хоча вони добре підходять для загального планування, їхній функціонал може бути надлишковим або недостатньо спеціалізованим саме для навчального розкладу (наприклад, відсутність полів для типу заняття, викладача, аудиторії як окремих сутностей). Можливості для ведення розгорнутих, структурованих нотаток, тісно інтегрованих із заняттями, часто обмежені.

4. Спеціалізовані програмні рішення для управління розкладом (часто для освітніх закладів):

- Системи управління навчанням (LMS) як Moodle, Google Classroom, Canvas LMS [4]. Ці платформи часто включають модулі для розкладу, але вони орієнтовані на потреби всього навчального закладу, а не на індивідуальне персоналізоване планування студента чи викладача з детальними нотатками.
- Комерційні та безкоштовні додатки-планувальники для студентів. Існує низка мобільних та десктопних додатків, розроблених спеціально для студентів (наприклад, My Study Life, iStudiez Pro, Egenda). Вони часто

пропонують зручний інтерфейс для введення розкладу, домашніх завдань, іспитів.

- Недоліки. Деякі з них можуть бути платними, мати обмежений функціонал у безкоштовних версіях, бути прив'язаними до конкретної платформи (мобільний додаток без повноцінної веб-версії) або не надавати достатньої гнучкості для ведення розгорнутих персоналізованих нотаток, інтегрованих з кожним заняттям.

5. Додатки для ведення нотаток:

- Evernote, Notion, Microsoft OneNote, Google Keep. Ці сервіси є потужними інструментами для створення та організації нотаток, підтримують форматування, вкладення файлів, теги тощо.
- Недоліки. Вони не є спеціалізованими для управління розкладом. Хоча можна створити нотатки для кожного заняття, відсутня пряма інтеграція з календарною сіткою розкладу, автоматичні нагадування про заняття та спеціалізовані поля для атрибутів заняття. Користувачеві доводиться підтримувати дві окремі системи або вручну пов'язувати інформацію.

Аналіз існуючих рішень показує наступне:

- Повністю паперові методи є застарілими для ефективного управління.
- Загальні офісні інструменти не мають необхідної спеціалізації.
- Популярні календарні додатки добре справляються з подіями, але можуть бути недостатньо гнучкими для специфіки навчального розкладу та глибокої інтеграції з персональними нотатками.
- Спеціалізовані додатки для студентів часто є хорошим рішенням, але можуть мати обмеження (платформа, вартість, функціонал нотаток).
- Потужні системи для нотаток не інтегровані з управлінням розкладом.

Існує ніша для розробки веб-додатку, який би поєднував у собі зручне управління розкладом занять зі специфічними для навчального процесу атрибутами та гнучкі можливості для ведення персоналізованих нотаток, прив'язаних до цих занять. Проєктований додаток "SheduleAppMVC" має на

меті заповнити цю нішу, пропонуючи простий, доступний (веб-доступ) та функціональний інструмент для індивідуального користувача.

1.3. Визначення цільової аудиторії та її функціональних потреб

Для успішної розробки та впровадження веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток критично важливим є чітке визначення його цільової аудиторії та розуміння її специфічних функціональних потреб. Хоча додаток може бути корисним для широкого кола користувачів, що потребують організації свого часу, основний фокус робиться на наступних групах [4]:

1. Студенти вищих навчальних закладів та коледжів:
   - Характеристики. Активні молоді люди, які мають насичений графік, що включає лекції, семінари, практичні та лабораторні роботи, консультації, іспити та заліки. Часто мають декілька предметів одночасно, різні аудиторії та викладачів. Потребують ефективного планування навчального процесу та позанавчальної діяльності. Активно користуються цифровими технологіями.
   - Функціональні потреби:
     - Можливість легко вносити та візуалізувати свій розклад занять на тиждень/місяць.
     - Швидке редагування розкладу у випадку змін.
     - Деталізація кожного заняття: назва предмету, тип, викладач, аудиторія, точний час.
     - Можливість додавати персональні нотатки до кожного заняття (наприклад, домашні завдання, ключові тези лекції, питання до викладача, терміни здачі робіт).
     - Пошук та фільтрація занять за різними параметрами.

- Доступ до розкладу з різних пристроїв (комп'ютер, планшет, смартфон).
- Можливість візуально розрізняти предмети (наприклад, за допомогою кольорів).

2. Викладачі та аспіранти:
   - Характеристики. Мають власний розклад викладання, консультацій, наукової роботи, зустрічей. Потребують інструменту для організації робочого часу та підготовки до занять. Можуть мати декілька груп студентів та різні дисципліни.
   - Функціональні потреби:
     - Ведення власного розкладу викладацької діяльності.
     - Можливість додавати нотатки до занять (наприклад, план лекції, список необхідних матеріалів, зауваження щодо студентів або тем).
     - Планування консультацій та інших робочих зустрічей.
     - Зручний перегляд свого завантаження на тиждень/місяць.
3. Учні старших класів та слухачі курсів:
   - Характеристики. Схожі потреби зі студентами, але, можливо, з меншою кількістю предметів та більшою структурованістю розкладу, заданою навчальним закладом. Також можуть потребувати ведення нотаток для підготовки до уроків та іспитів.
   - Функціональні потреби:
     - Простий інструмент для фіксації розкладу уроків та додаткових занять.
     - Можливість додавати нотатки та завдання до уроків.
4. Фрілансери та особи з гнучким графіком, що потребують структурування періодичних завдань:
   - Характеристики. Хоча основний фокус на навчальному процесі, додаток може бути корисним для будь-кого, хто має регулярні, повторювані події та потребує ведення нотаток до них.

- Функціональні потреби:
  - Можливість створювати власні типи подій.
  - Ведення нотаток до робочих завдань або зустрічей.

Загальні функціональні потреби, що випливають з аналізу цільової аудиторії:

- Інтуїтивність та простота використання. Інтерфейс має бути зрозумілим без тривалого вивчення інструкцій.
- Надійність. Дані користувача (розклад, нотатки) мають безпечно зберігатися.
- Персоналізація. Кожен користувач працює зі своїм власним, ізольованим набором даних.
- Доступність. Можливість доступу до додатку через веб-браузер з різних пристроїв.
- Основні CRUD-операції. Легке створення, читання, оновлення та видалення як занять, так і нотаток.
- Візуалізація. Чітке та наочне представлення розкладу.
- Пошук та фільтрація. Можливість швидко знаходити потрібну інформацію.

Розуміння цих потреб дозволить сформулювати більш точні функціональні та нефункціональні вимоги до веб-додатку на наступних етапах аналізу.

1.4. Техніко-економічне обґрунтування вибору створення веб-додатку з персоналізованими нотатками

Рішення про розробку веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток ґрунтується на аналізі потреб цільової аудиторії, огляді існуючих рішень та оцінці потенційних переваг власної розробки.

Технічні аспекти обґрунтування:

1. Як показав аналіз у підрозділі 1.2, існуючі універсальні рішення (календарі, загальні планувальники) часто не повністю задовольняють специфічні потреби студентів та викладачів щодо деталізації розкладу (тип заняття, викладач, аудиторія) та глибокої інтеграції з персональними нотатками до кожного заняття. Спеціалізовані додатки можуть бути платформо-залежними або мати обмеження. Розробка власного веб-додатку дозволяє точно реалізувати необхідний функціонал.
2. Використання ASP.NET Core MVC на платформі .NET забезпечує високу продуктивність, надійність, безпеку та широкі можливості для розробки. Мова C# [5] є потужною та добре підтримуваною. Даний стек технологій дозволяє створювати масштабовані та легко підтримувані веб-додатки.
3. Веб-додаток за своєю природою є кросплатформеним і доступним з будь-якого пристрою, що має веб-браузер та доступ до Інтернету. Це усуває обмеження, пов'язані з мобільними або десктопними додатками, що вимагають встановлення.
4. Власна розробка надає повний контроль над архітектурою системи, структурою бази даних, логікою роботи та подальшим розвитком функціоналу. Це дозволяє оперативно реагувати на потреби користувачів та вносити необхідні зміни.
5. Поєднання функціоналу управління розкладом та ведення нотаток в одному інтегрованому середовищі є ключовою перевагою, оскільки усуває необхідність використання кількох окремих інструментів та ручного перенесення інформації.

Економічні аспекти обґрунтування:

1. Використання Visual Studio Community Edition, .NET Core SDK, SQLite [6] є безкоштовним для індивідуальної розробки та освітніх цілей, що суттєво знижує початкові витрати на програмне забезпечення.

2. Існують варіанти безкоштовного або недорогого хостингу для ASP.NET Core додатків (наприклад, Azure App Service Free Tier, або інші провайдери з підтримкою .NET), що робить розгортання та підтримку додатку економічно вигідним, особливо на початкових етапах або для некомерційного використання.
3. На відміну від деяких комерційних планувальників або систем управління навчанням, власний додаток не вимагатиме регулярних ліцензійних платежів від користувачів (якщо не планується його комерціалізація).
4. Хоча це непрямий економічний ефект для розробника, ефективний інструмент планування може суттєво підвищити продуктивність студентів та викладачів, допомагаючи їм краще організовувати свій час, вчасно виконувати завдання та зменшувати стрес, пов'язаний з дедлайнами та насиченим графіком. Це, в свою чергу, може позитивно вплинути на їхні академічні та професійні результати.
5. Для студента-розробника створення такого веб-додатку є цінним практичним досвідом, який підвищує його кваліфікацію та конкурентоспроможність на ринку праці. Цей досвід є важливим нематеріальним активом.
6. У разі успішної реалізації та позитивного відгуку від користувачів, додаток може бути розширений додатковим преміум-функціоналом та потенційно монетизований, хоча це не є першочерговою метою даного дипломного проєкту.

Обмеження та ризики:

- Створення власного додатку з нуля вимагає значних часових затрат на проєктування, розробку, тестування та налагодження.
- Після розгортання додаток потребуватиме періодичної підтримки, виправлення можливих помилок та, можливо, оновлення для сумісності з новими версіями браузерів або операційних систем.

Незважаючи на зазначені обмеження, технічні переваги (задоволення специфічних потреб, контроль, доступність) та економічні аспекти (низькі початкові витрати, потенціал для підвищення продуктивності користувачів, набуття досвіду) роблять розробку веб-додатку доцільною та обґрунтованою. Інтеграція функціоналу управління розкладом з можливістю ведення персоналізованих нотаток в єдиному веб-інтерфейсі є ключовою цінністю проєкту.

1.5. Формулювання функціональних та нефункціональних вимог до веб-додатку

На основі аналізу предметної області, потреб цільової аудиторії та загального призначення проєкту, формулюються наступні функціональні та нефункціональні вимоги до веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток.

Функціональні вимоги (ФВ):

- ФВ-1: Управління обліковими записами користувачів
  - ФВ-1.1. Система повинна надавати можливість реєстрації нових користувачів (наприклад, за допомогою електронної пошти та пароля).
  - ФВ-1.2. Система повинна забезпечувати автентифікацію зареєстрованих користувачів (вхід до системи).
  - ФВ-1.3. Система повинна надавати можливість виходу користувача із системи.
  - ФВ-1.4. Кожен користувач повинен мати доступ лише до власного розкладу та нотаток.
  - ФВ-1.5 (опціонально): Можливість відновлення паролю.
- ФВ-2: Управління розкладом занять

- ФВ-2.1. Користувач повинен мати можливість створювати нове заняття, вказуючи:
  - Назву предмету (обов'язково).
  - Тип заняття (обов'язково, вибір зі списку: лекція, практика, лабораторна тощо).
  - Час початку (обов'язково, дата та час).
  - Час закінчення (обов'язково, дата та час; має бути пізніше часу початку).
  - Викладача (необов'язково).
  - Аудиторію (необов'язково).
  - Колір для візуального маркування події (необов'язково).
- ФВ-2.2. Користувач повинен мати можливість переглядати свій розклад у вигляді списку/таблиці.
- ФВ-2.3. Користувач повинен мати можливість переглядати деталі конкретного заняття.
- ФВ-2.4. Користувач повинен мати можливість редагувати існуючі заняття у своєму розкладі.
- ФВ-2.5. Користувач повинен мати можливість видаляти заняття зі свого розкладу.

- ФВ-3: Управління персоналізованими нотатками
  - ФВ-3.1. Користувач повинен мати можливість створювати нову текстову нотатку, прив'язану до конкретного заняття.
  - ФВ-3.2. Користувач повинен мати можливість переглядати список нотаток, прив'язаних до конкретного заняття.
  - ФВ-3.3. Користувач повинен мати можливість редагувати вміст існуючої нотатки.
  - ФВ-3.4. Користувач повинен мати можливість видаляти нотатки.
  - ФВ-3.5. Нотатки мають відображати дату створення та останнього оновлення.
- ФВ-4: Пошук та фільтрація занять

- ФВ-4.1. Користувач повинен мати можливість здійснювати пошук занять за текстовим фрагментом (назва предмету, викладач, аудиторія).
- ФВ-4.2. Користувач повинен мати можливість фільтрувати заняття за типом.
- ФВ-4.3. Користувач повинен мати можливість фільтрувати заняття за діапазоном дат (дата початку та/або дата закінчення).
- ФВ-4.4. Система повинна дозволяти комбінувати критерії пошуку та фільтрації.
- ФВ-4.5. Користувач повинен мати можливість скинути застосовані фільтри.

Нефункціональні вимоги (НФВ):

- НФВ-1: Продуктивність
  - НФВ-1.1. Час завантаження основних сторінок (список розкладу, форма додавання/редагування) не повинен перевищувати 3 секунди за умов середньої швидкості інтернет-з'єднання.
  - НФВ-1.2. Система повинна швидко реагувати на дії користувача (наприклад, збереження заняття, застосування фільтрів) – не більше 2 секунд.
- НФВ-2: Надійність
  - НФВ-2.1. Система повинна забезпечувати цілісність та збереження даних користувача (розклад, нотатки).
  - НФВ-2.2. Система повинна бути доступною для користувачів 99% часу (за винятком планових технічних робіт).
  - НФВ-2.3. Система повинна коректно обробляти помилки введення даних користувачем та надавати інформативні повідомлення.
- НФВ-3: Безпека
  - НФВ-3.1. Дані користувачів (особливо облікові дані) повинні зберігатися у зашифрованому вигляді (забезпечується ASP.NET Core Identity).

- НФВ-3.2. Доступ до розкладу та нотаток має бути обмежений лише авторизованим власником цих даних.
- НФВ-3.3. Система повинна бути захищена від поширених веб-вразливостей (наприклад, XSS, CSRF – забезпечується вбудованими механізмами ASP.NET Core).

- НФВ-4: Зручність використання (Usability)
  - НФВ-4.1. Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким для освоєння новими користувачами.
  - НФВ-4.2. Навігація по сайту має бути простою та логічною.
  - НФВ-4.3. Додаток повинен мати адаптивний дизайн для коректного відображення на різних розмірах екранів (десктопи, планшети, мобільні телефони).
- НФВ-5: Підтримуваність (Maintainability)
  - НФВ-5.1. Код додатку повинен бути добре структурованим, читабельним та коментованим для полегшення подальшої підтримки та модифікації.
  - НФВ-5.2. Архітектура додатку (MVC) повинна сприяти легкості внесення змін та додавання нового функціоналу.
- НФВ-6: Масштабованість (Scalability)
  - НФВ-6.1. Архітектура додатку повинна дозволяти потенційне збільшення кількості користувачів та обсягу даних без суттєвого погіршення продуктивності (хоча для SQLite існують обмеження порівняно з серверними СУБД).
- НФВ-7: Сумісність
  - НФВ-7.1. Веб-додаток повинен коректно працювати в останніх версіях поширених веб-браузерів (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).

Формулювання цих вимог є основою для подальшого проєктування, розробки та тестування веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Обґрунтування вибору архітектури веб-додатку

Вибір архітектури є фундаментальним етапом у процесі розробки будь-якого програмного забезпечення, оскільки він визначає структуру додатку, взаємодію його компонентів, а також впливає на його масштабованість, підтримуваність та гнучкість. Для розробки веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток було обрано архітектурний шаблон Model-View-Controller (MVC) [7]. Цей вибір обґрунтовується низкою переваг, які даний шаблон надає для створення структурованих, керованих та легко тестованих веб-додатків.

Архітектура MVC передбачає розділення додатку на три взаємопов'язані компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Кожен з цих компонентів має чітко визначену відповідальність, що сприяє кращій організації коду та полегшує його подальшу модифікацію та розвиток.

Модель (Model) представляє дані додатку та бізнес-логіку, пов'язану з цими даними. Вона відповідає за доступ до даних, їх обробку, валідацію та збереження. У контексті розроблюваного веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток, Модель включатиме сутності, такі як "Заняття" (ScheduledEvent), "Нотатка" (Note), "Користувач" (ApplicationUser), а також логіку для їх створення, оновлення, видалення та отримання з бази даних. Модель не залежить від інтерфейсу користувача і не містить логіки відображення.

Представлення (View) відповідає за візуалізацію даних, отриманих від Моделі, та за взаємодію з користувачем. Це та частина додатку, яку бачить користувач у своєму веб-браузері. Представлення генерують HTML-розмітку [9], відображають форми для введення даних, таблиці, списки та інші елементи інтерфейсу. У нашому випадку, це будуть сторінки для відображення

розкладу, форми для створення та редагування занять і нотаток, сторінки деталей тощо. Представлення не містять бізнес-логіки, а лише відображають дані, підготовлені Контролером.

Контролер (Controller) виступає як посередник між Моделлю та Представленням. Він обробляє HTTP-запити від користувача (наприклад, перехід за посиланням або відправка форми), взаємодіє з Моделлю для виконання необхідних операцій з даними (отримання, оновлення, збереження), а потім обирає відповідне Представлення для формування відповіді користувачеві, передаючи йому необхідні дані. Контролер інкапсулює логіку управління потоком запитів та взаємодії компонентів.

Схематично взаємодію компонентів в архітектурі MVC для проєктованого веб-додатку можна представити наступним чином (Рис. 2.1.). Користувач взаємодіє з Представленням, яке надсилає запит до Контролера. Контролер обробляє запит, за потреби звертається до Моделі для маніпуляцій з даними. Модель повертає дані Контролеру, який потім обирає відповідне Представлення та передає йому ці дані для відображення користувачеві.

Рисунок 2.1 – Архітектура Model-View-Controller (MVC) веб-додатку

Переваги вибору архітектури MVC для даного проєкту:

1. Кожен компонент (Модель, Представлення, Контролер) має свою чітко визначену роль, що полегшує розуміння, розробку та тестування окремих частин додатку.

2. Завдяки слабкій зв'язаності компонентів, зміни в одному з них (наприклад, в інтерфейсі користувача) мінімально впливають на інші (наприклад, на бізнес-логіку). Це спрощує внесення змін та додавання нового функціоналу.
3. Різні команди або розробники можуть працювати над різними компонентами (наприклад, один над UI, інший над бізнес-логікою) одночасно.
4. Кожен компонент можна тестувати незалежно. Наприклад, бізнес-логіку в Моделі можна тестувати без залучення UI, а логіку Контролерів – за допомогою юніт-тестів.
5. Логіка, реалізована в Моделі або Контролерах, може бути повторно використана різними Представленнями.
6. MVC є стандартом де-факто для багатьох веб-фреймворків, включаючи ASP.NET Core, і добре зарекомендував себе для створення масштабованих та керованих веб-рішень.

Враховуючи ці переваги, архітектура MVC [10] є оптимальним вибором для розробки веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток, оскільки вона забезпечує необхідну структуру, гнучкість та підтримуваність для реалізації всього запланованого функціоналу та потенційного подальшого розвитку проєкту.

2.2. Вибір і обґрунтування технологічного стеку

Вибір відповідного технологічного стеку є критично важливим для успішної реалізації будь-якого програмного проєкту, оскільки він визначає інструменти, мови програмування, фреймворки та бібліотеки, які будуть використовуватися на всіх етапах розробки. Для веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток було обрано технологічний стек, що базується на платформі .NET та сучасних веб-

технологіях, з метою забезпечення продуктивності, надійності, безпеки та зручності розробки.

Серверна частина додатку (backend) реалізована з використанням ASP.NET Core MVC [11]. Цей фреймворк від Microsoft є потужним, гнучким та високопродуктивним рішенням для створення сучасних веб-додатків та API. Він побудований на базі .NET (у даному проєкті використовується .NET 8), що забезпечує кросплатформеність, високу швидкість виконання та доступ до широкої екосистеми бібліотек. Архітектура MVC, як було обґрунтовано раніше, сприяє чіткому розділенню логіки, що полегшує розробку та підтримку. Основною мовою програмування для серверної частини обрано C#, яка є сучасною, об'єктно-орієнтованою мовою з сильною типізацією, що сприяє написанню надійного та легко читабельного коду. Вбудовані механізми ASP.NET Core, такі як dependency injection, middleware конвеєр та система маршрутизації, значно спрощують процес розробки.

Для взаємодії з базою даних використовується Entity Framework Core (EF Core). Це сучасний, об'єктно-реляційний мапер (ORM) для .NET, який дозволяє розробникам працювати з базою даних, використовуючи об'єкти C#, замість написання SQL-запитів вручну [12]. EF Core спрощує операції CRUD (Create, Read, Update, Delete), підтримує міграції для управління схемою бази даних та інтегрується з різними системами управління базами даних. У даному проєкті як система управління базами даних (СУБД) обрано SQLite. SQLite є легкою, файловою, вбудованою СУБД, яка не потребує окремого серверного процесу. Це робить її ідеальним вибором для локальної розробки, невеликих та середніх додатків, а також для дипломних проєктів, де простота налаштування та розгортання є важливими факторами. Вона забезпечує достатню продуктивність для потреб даного веб-додатку та легко інтегрується з EF Core [13].

Автентифікація та авторизація користувачів реалізована за допомогою ASP.NET Core Identity. Ця система надає готовий набір функцій для управління користувачами, включаючи реєстрацію, вхід, управління

паролями, ролями та клеймами. Вона добре інтегрується з Entity Framework Core для зберігання даних користувачів у базі даних та забезпечує надійний механізм захисту доступу до персоналізованих даних розкладу та нотаток.

Клієнтська частина додатку (frontend) побудована з використанням стандартних веб-технологій: HTML (HyperText Markup Language) для структурування вмісту сторінок, CSS (Cascading Style Sheets) для їх візуального оформлення та JavaScript для реалізації динамічної поведінки на стороні клієнта, якщо це необхідно [14]. Для прискорення розробки інтерфейсу користувача та забезпечення його адаптивності використовується CSS-фреймворк Bootstrap. Bootstrap надає великий набір готових компонентів (кнопки, форми, навігаційні панелі, сітки тощо) та стилів, що дозволяє швидко створювати сучасні та адаптивні веб-інтерфейси, які добре виглядають на різних пристроях. Синтаксис Razor використовується у файлах представлень (.cshtml) для вбудовування C# коду в HTML-розмітку, що дозволяє динамічно генерувати вміст сторінок на сервері.

Середовищем розробки обрано Microsoft Visual Studio [15], яка є потужною інтегрованою системою розробки (IDE) з відмінною підтримкою .NET та C#, інструментами для відладки, управлінням NuGet-пакетами та інтеграцією з системами контролю версій. Для контролю версій коду використовується система Git.

Обраний технологічний стек забезпечує збалансоване поєднання продуктивності, надійності, швидкості розробки та доступності інструментів, що робить його оптимальним для реалізації веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток.

2.3. Проєктування основних модулів системи

Проєктування веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток передбачає виділення кількох ключових функціональних модулів, кожен з яких відповідає за певну частину

загальної функціональності системи. Такий модульний підхід дозволяє структурувати розробку, спростити тестування та забезпечити кращу підтримуваність коду. Основними модулями системи є: модуль управління користувачами, модуль управління розкладом занять, модуль управління персоналізованими нотатками та функціонал пошуку і фільтрації.

Модуль управління користувачами є фундаментальним, оскільки він забезпечує автентифікацію та авторизацію, що є необхідним для персоналізованого доступу до даних. Цей модуль відповідає за процеси реєстрації нових користувачів, входу існуючих користувачів до системи, виходу із системи, а також за управління профілями користувачів (наприклад, зміна пароля). В основі цього модуля лежить система ASP.NET Core Identity [16], яка надає надійні та перевірені механізми для роботи з обліковими записами. Взаємодія користувача з цим модулем відбувається через спеціалізовані сторінки інтерфейсу (реєстрація, вхід), які обробляються відповідними компонентами ASP.NET Core Identity (Рис. 2.2.). Дані користувачів зберігаються у базі даних в таблицях, що створюються та управляються системою Identity.

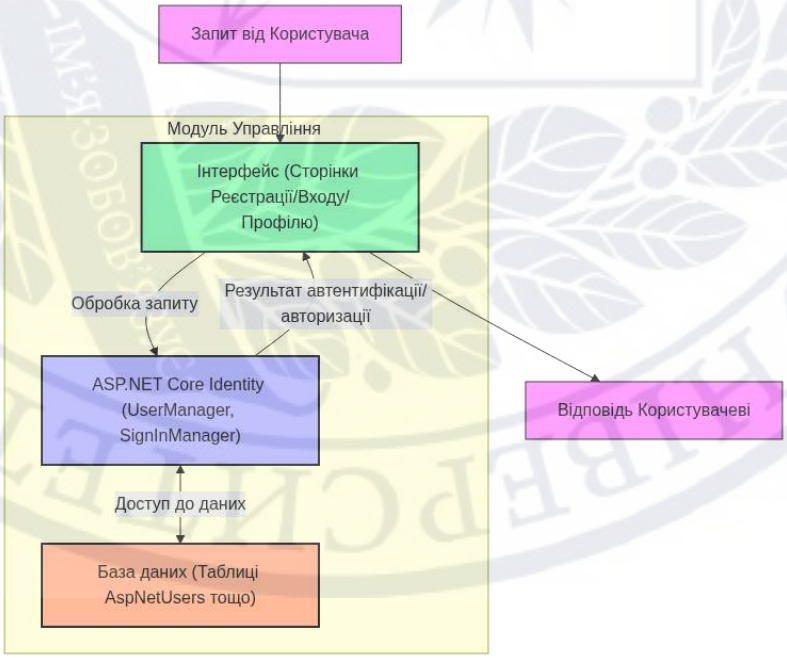

Рисунок 2.2 – Компонентна схема модуля управління користувачами

Модуль управління розкладом занять є центральним у функціональності веб-додатку. Він дозволяє авторизованим користувачам створювати, переглядати, редагувати та видаляти записи про заняття у своєму персональному розкладі. Кожне заняття характеризується набором атрибутів, таких як назва предмету, тип заняття, час початку та закінчення, викладач, аудиторія та колір для візуалізації. Цей модуль реалізований за допомогою ScheduleController, який обробляє HTTP-запити [17] від користувача, сервісу ScheduleService, що інкапсулює бізнес-логіку та взаємодію з базою даних, та моделей даних (ScheduledEvent, EventType). Взаємодія з базою даних відбувається через ApplicationDbContext.

Компонентна структура цього модуля представлена на Рис. 2.3.

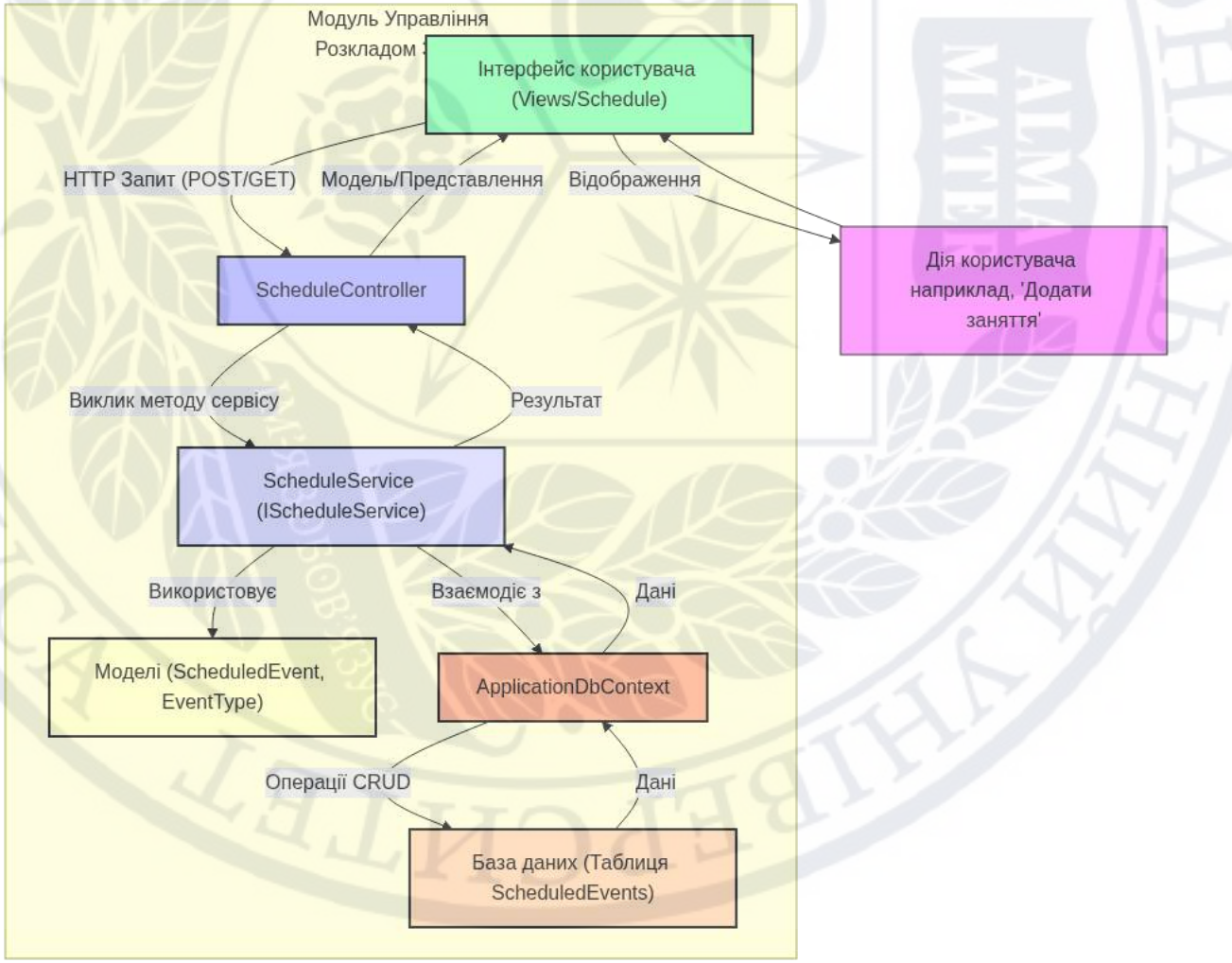

Рисунок 2.3 – Компонентна схема модуля управління розкладом занять

Модуль управління персоналізованими нотатками тісно інтегрований з модулем розкладу і дозволяє користувачам додавати, переглядати, редагувати та видаляти текстові нотатки, прив'язані до конкретних занять. Це дає можливість зберігати важливу інформацію, завдання або будь-які інші записи безпосередньо в контексті відповідного заняття. Цей модуль включає NotesController, сервіс NoteService та модель Note. Взаємодія з базою даних також здійснюється через ApplicationDbContext. Схема взаємодії компонентів цього модуля (Рис. 2.4.) є аналогічною до модуля управління розкладом, з акцентом на операції з нотатками та їх зв'язок із заняттями.

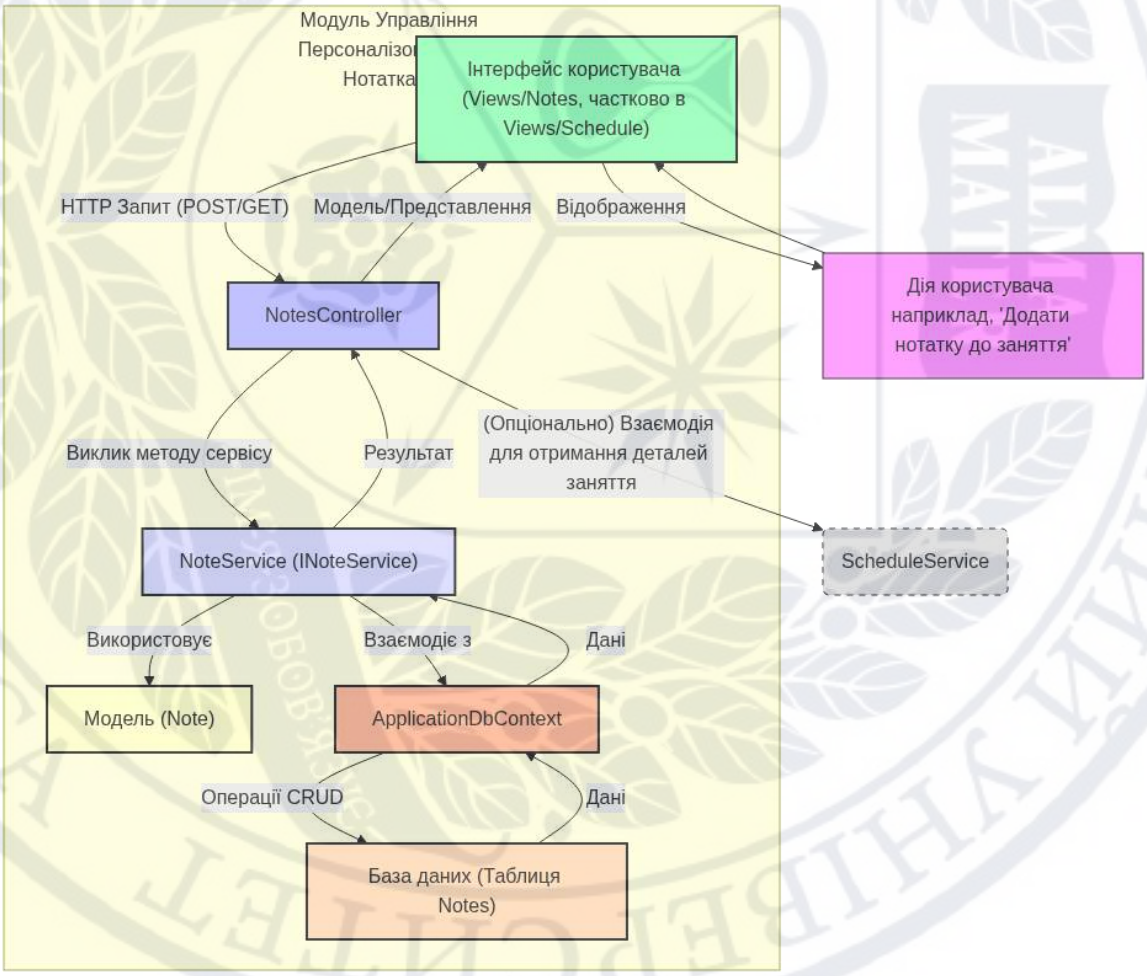

Рисунок 2.4 – Компонентна схема модуля управління персоналізованими нотатками

Функціонал пошуку та фільтрації занять реалізований як частина модуля управління розкладом. Він надає користувачеві можливість швидко знаходити потрібні заняття у своєму розкладі за різними критеріями: текстовий пошук за назвою предмету, ім'ям викладача або аудиторією, фільтрація за типом заняття та за діапазоном дат. Цей функціонал реалізований шляхом розширення дії Index у ScheduleController для приймання параметрів фільтрації та модифікації відповідного методу в ScheduleService для формування запиту до бази даних з урахуванням цих параметрів. Схема роботи цього функціоналу показана на Рис. 2.5.

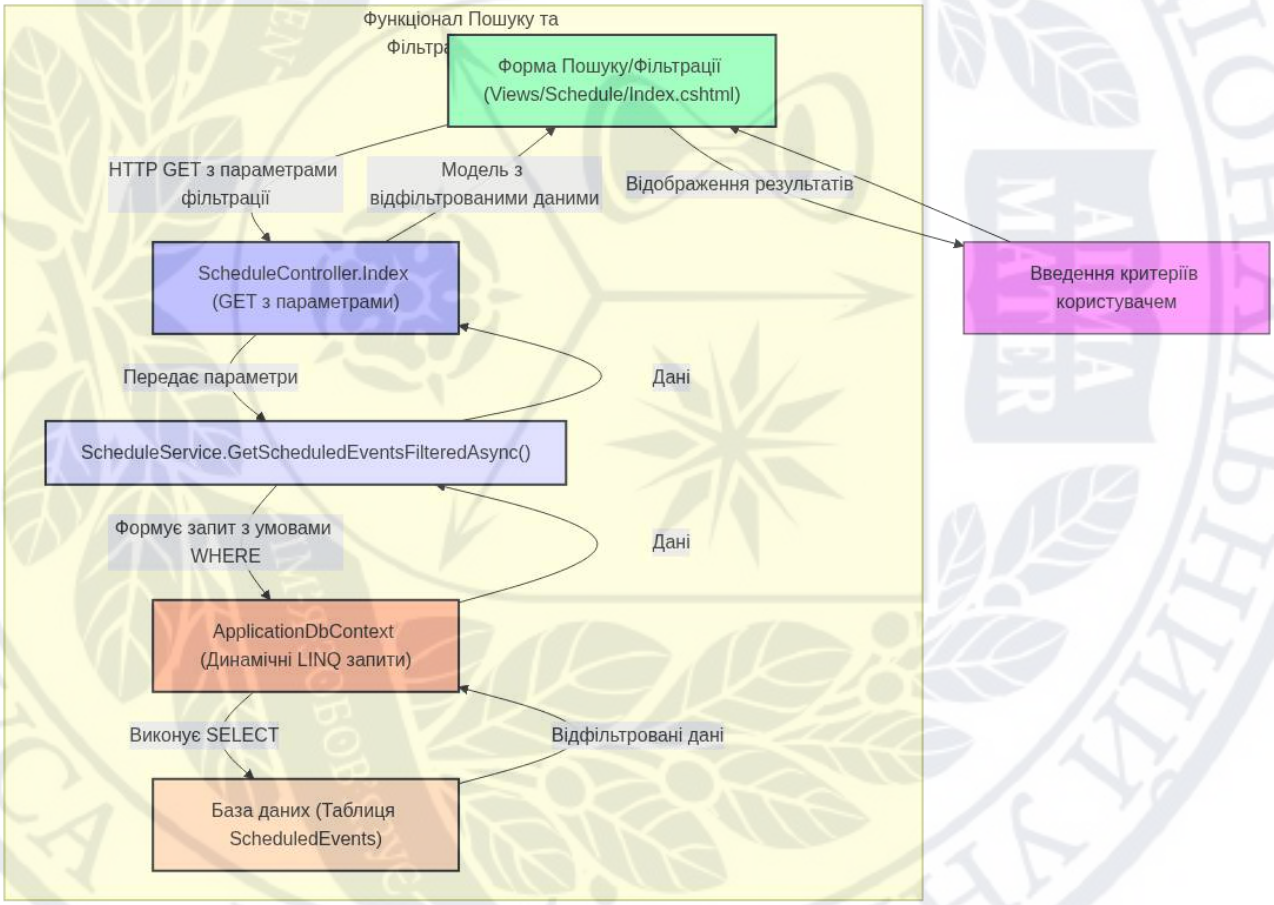

Рисунок 2.5 – Схема роботи функціоналу пошуку та фільтрації занять

Взаємодія цих модулів забезпечує комплексну функціональність веб-додатку, дозволяючи користувачам ефективно управляти своїм розкладом та

пов'язаною інформацією. Чітке розмежування відповідальностей між модулями та їх компонентами сприяє створенню надійної та гнучкої системи.

2.4. Проєктування структури бази даних

Проєктування структури бази даних є ключовим аспектом розробки веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток, оскільки від коректності та ефективності моделі даних залежить функціональність, продуктивність та масштабованість системи. Для зберігання даних використовується реляційна база даних SQLite [18], а взаємодія з нею здійснюється через Entity Framework Core за підходом "Code First". Це означає, що структура бази даних визначається на основі класів моделей C#.

Основними сутностями, що зберігаються в базі даних, є користувачі, заняття та нотатки. Додатково, система ASP.NET Core Identity генерує власні таблиці для управління користувачами, ролями та пов'язаними даними (клейми, логіни, токени).

Основні таблиці, що проєктуються для додатку:

1. AspNetUsers (таблиця Identity): Зберігає інформацію про зареєстрованих користувачів. Ключові поля включають Id (первинний ключ, рядок), UserName, NormalizedUserName, Email, NormalizedEmail, PasswordHash, SecurityStamp, ConcurrencyStamp та інші стандартні поля Identity. Ця таблиця розширюється кастомним класом ApplicationUser, який додає навігаційні властивості для зв'язку з заняттями та нотатками конкретного користувача.
2. ScheduledEvents (Заняття): Ця таблиця зберігає інформацію про кожне заплановане заняття. Її атрибути включають:
   - Id (ціле число, первинний ключ, автоінкремент): Унікальний ідентифікатор заняття.
   - SubjectName (рядок, обов'язкове): Назва предмету або заняття.

- Type (ціле число, обов'язкове): Тип заняття, що відповідає значенням перерахування EventType (наприклад, Лекція, Практика).
- Teacher (рядок, необов'язкове): Ім'я викладача.
- Auditorium (рядок, необов'язкове): Номер аудиторії або місце проведення.
- StartTime (дата/час, обов'язкове): Дата та час початку заняття.
- EndTime (дата/час, обов'язкове): Дата та час закінчення заняття.
- Color (рядок, необов'язкове): Колір для візуального маркування події у форматі HEX (наприклад, "#FF0000").
- UserId (рядок, обов'язкове, зовнішній ключ): Ідентифікатор користувача з таблиці AspNetUsers, якому належить це заняття.

3. Notes (Нотатки): Ця таблиця зберігає персоналізовані нотатки користувачів. Її атрибути включають:
   - Id (ціле число, первинний ключ, автоінкремент): Унікальний ідентифікатор нотатки.
   - Content (текст, обов'язкове): Вміст нотатки.
   - CreatedAt (дата/час): Дата та час створення нотатки.
   - LastUpdatedAt (дата/час): Дата та час останнього оновлення нотатки.
   - UserId (рядок, обов'язкове, зовнішній ключ): Ідентифікатор користувача з таблиці AspNetUsers, якому належить ця нотатка.
   - ScheduledEventId (ціле число, необов'язкове, зовнішній ключ): Ідентифікатор заняття з таблиці ScheduledEvents, до якого прив'язана ця нотатка. Якщо значення NULL, нотатка вважається загальною, не прив'язаною до конкретного заняття.

Зв'язки між таблицями:

- AspNetUsers та ScheduledEvents. Зв'язок "один-до-багатьох". Один користувач може мати багато запланованих занять, але кожне заняття належить лише одному користувачеві. Зв'язок реалізується через

зовнішній ключ UserId у таблиці ScheduledEvents, який посилається на Id у таблиці AspNetUsers. При видаленні користувача всі пов'язані з ним заняття також видаляються (каскадне видалення).

- AspNetUsers та Notes. Зв'язок "один-до-багатьох". Один користувач може мати багато нотаток, але кожна нотатка належить лише одному користувачеві. Зв'язок реалізується через зовнішній ключ UserId у таблиці Notes, який посилається на Id у таблиці AspNetUsers. Поведінка при видаленні користувача для нотаток, не прив'язаних до занять, налаштована на обмеження (Restrict), щоб уникнути випадкової втрати даних, якщо це не є бажаним.
- ScheduledEvents та Notes. Зв'язок "один-до-багатьох". Одне заняття може мати багато пов'язаних нотаток, але кожна нотатка (якщо вона прив'язана до заняття) пов'язана лише з одним заняттям. Зв'язок реалізується через зовнішній ключ ScheduledEventId у таблиці Notes, який посилається на Id у таблиці ScheduledEvents. Це поле може бути NULL. При видаленні заняття всі пов'язані з ним нотатки також видаляються (каскадне видалення).

ER-діаграма, що ілюструє структуру основних таблиць та зв'язків між ними, представлена на Рис. 2.6.

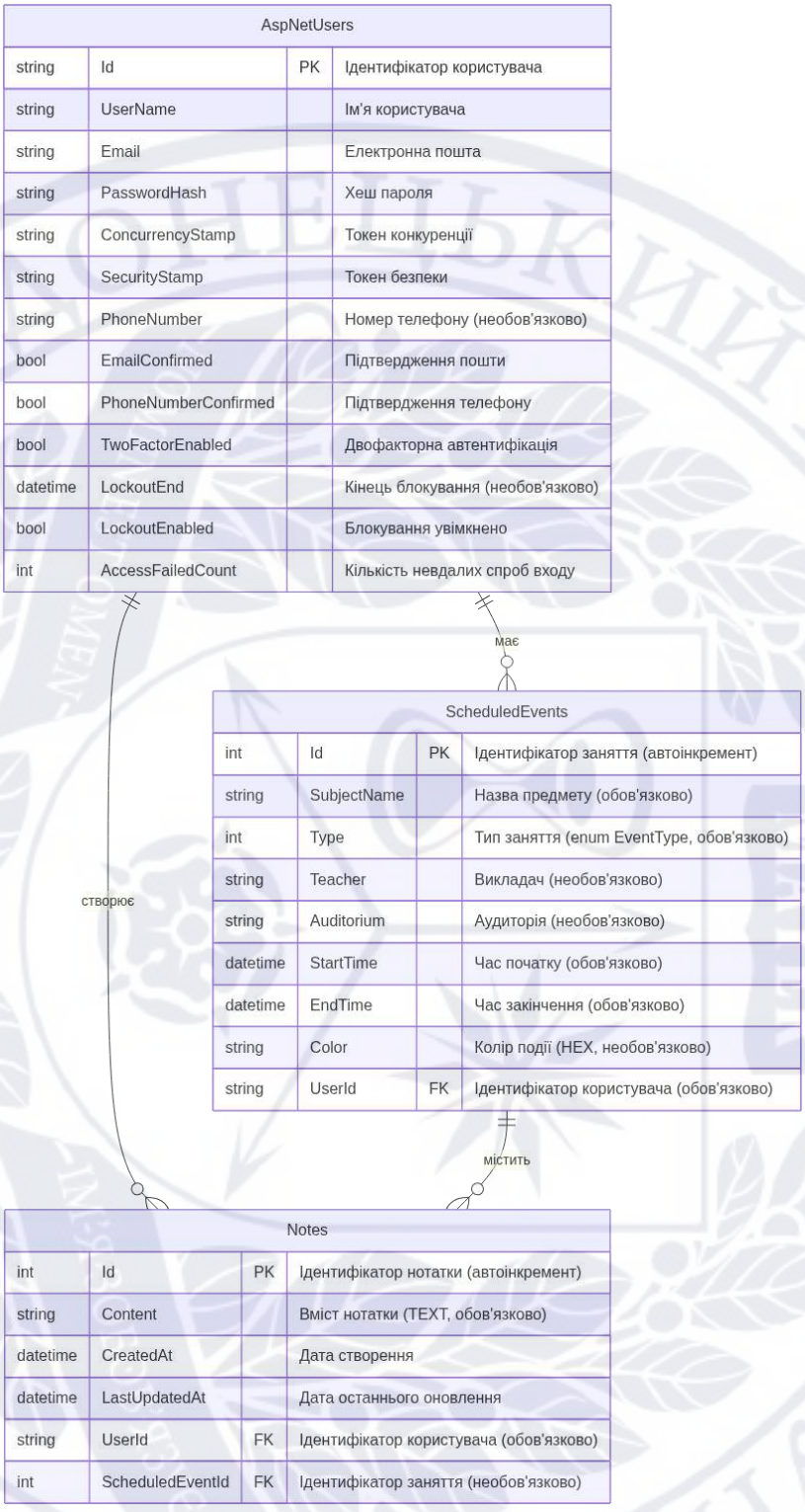

Рисунок 2.6 – ER-діаграма бази даних веб-додатку

На додаток до цих основних таблиць, ASP.NET Core Identity також створює таблиці для ролей (AspNetRoles), клеймів користувачів (AspNetUserClaims), логінів користувачів (AspNetUserLogins), токенів

користувачів (AspNetUserTokens) та зв'язків між користувачами та ролями (AspNetUserRoles). Ці таблиці забезпечують повноцінну систему управління ідентифікацією та доступом.

Обрана структура бази даних є достатньо гнучкою для зберігання всієї необхідної інформації, забезпечує цілісність даних через визначені зв'язки та зовнішні ключі, та дозволяє ефективно виконувати запити для реалізації функціоналу веб-додатку. Використання EF Core міграцій дозволяє легко еволюціонувати схему бази даних у майбутньому при додаванні нового функціоналу.

2.5. Проєктування інтерфейсу користувача (UI) та досвіду користувача (UX)

Проєктування інтерфейсу користувача (UI) та досвіду користувача (UX) для веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток спрямоване на створення інтуїтивно зрозумілого, зручного та ефективного середовища для взаємодії користувача з системою. Ключовими принципами при проєктуванні UI/UX [19] є простота, послідовність, доступність та візуальна привабливість.

Загальна структура інтерфейсу побудована на основі стандартного макета (_Layout.cshtml), який включає навігаційну панель (хедер), основну область для відображення контенту сторінки та футер. Навігаційна панель забезпечує швидкий доступ до основних розділів: головної сторінки, сторінки "Мій Розклад" (для автентифікованих користувачів), а також елементів управління обліковим записом (реєстрація, вхід, вихід, управління профілем), які надаються частковим представленням _LoginPartial.cshtml. Використання CSS-фреймворку Bootstrap [20] забезпечує адаптивність інтерфейсу, що дозволяє комфортно користуватися додатком на пристроях з різними розмірами екранів, від десктопних моніторів до мобільних телефонів.

Основною сторінкою взаємодії для автентифікованого користувача є сторінка "Мій Розклад" (Views/Schedule/Index.cshtml). Вона спроєктована таким чином, щоб надавати огляд запланованих занять у вигляді таблиці, де кожне заняття представлене окремим рядком. Для візуального розрізнення та швидкої ідентифікації, рядок таблиці, що відповідає заняттю, може мати фоновий колір, вказаний користувачем для цієї події, при цьому колір тексту автоматично підбирається для забезпечення контрастності та читабельності. Кожна колонка таблиці відображає ключову інформацію про заняття: назву предмету, тип (локалізований українською мовою), час початку та закінчення, ім'я викладача та аудиторію. Колонка "Дії" містить кнопки для редагування заняття, перегляду його деталей, управління нотатками (з відображенням їх кількості) та видалення заняття. Ці кнопки стилізовані за допомогою Bootstrap [21] та мають іконки для кращої інтуїтивності.

Для покращення UX на сторінці "Мій Розклад" реалізовано панель пошуку та фільтрації. Ця панель за замовчуванням може бути згорнутою і розкриватися при натисканні на відповідну кнопку ("Фільтри та Пошук"), розташовану поруч із кнопкою "Додати нове заняття". Такий підхід дозволяє не перевантажувати інтерфейс, якщо користувач не потребує фільтрації. Форма фільтрації дозволяє здійснювати пошук за текстовим рядком (по назві предмету, викладачу, аудиторії), фільтрувати за типом заняття (випадаючий список з локалізованими назвами) та за діапазоном дат. Після застосування фільтрів, форма зберігає введені значення, що дозволяє користувачеві бачити поточні критерії відбору. Також передбачена кнопка "Очистити фільтри" для швидкого повернення до повного списку занять.

Процес створення та редагування занять відбувається на окремих сторінках (Views/Schedule/Create.cshtml та Views/Schedule/Edit.cshtml), які використовують спільне часткове представлення _ScheduledEventFormPartial.cshtml для самої форми. Форма містить поля для всіх атрибутів заняття, використовуючи відповідні елементи вводу HTML5 (наприклад, datetime-local для вибору дати та часу, color для вибору кольору).

Валідація даних відбувається як на стороні клієнта (за допомогою jQuery Unobtrusive Validation), так і на стороні сервера, з відображенням інформативних повідомлень про помилки безпосередньо на формі. Це допомагає користувачеві швидко виправити помилки введення.

Аналогічний підхід застосовується і для управління нотатками. Сторінка Views/Notes/IndexForEvent.cshtml відображає список нотаток для конкретного заняття та дозволяє додати нову. Форми створення та редагування нотаток (Views/Notes/Create.cshtml, Views/Notes/Edit.cshtml) також використовують часткове представлення для поля вводу тексту нотатки.

Загальний потік взаємодії користувача з системою (user flow) для ключового сценарію, такого як додавання нового заняття, можна проілюструвати наступною схемою (Рис. 2.7.).

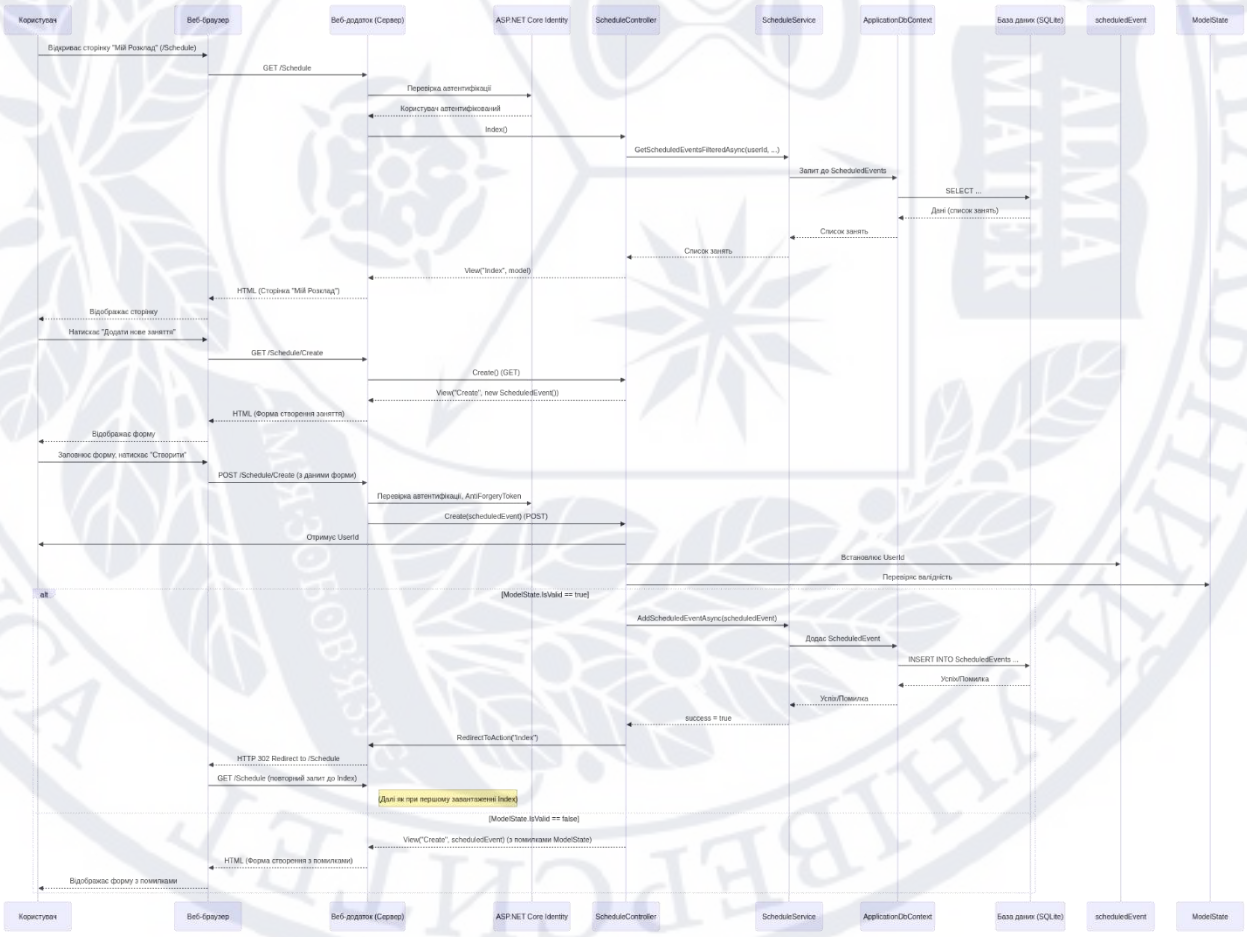

Рисунок 2.7 – Схема взаємодії користувача при додаванні нового заняття

Для забезпечення позитивного досвіду користувача використовуються повідомлення TempData (наприклад, "Заняття успішно створено!"), які відображаються після виконання успішних операцій. Кнопки та посилання мають чіткі назви та іконки, що полегшує їх ідентифікацію. Загалом, дизайн інтерфейсу спрямований на мінімізацію когнітивного навантаження на користувача та забезпечення швидкого та ефективного виконання поставлених завдань.

Для більш наочного представлення основних варіантів використання системи розроблено Use Case діаграму (Рис. 2.8). Ця діаграма відображає ключові дії, які може виконувати зареєстрований користувач веб-додатку. Основним актором є "Користувач", який взаємодіє з системою для управління своїм розкладом та нотатками. Варіанти використання включають реєстрацію та автентифікацію, повний набір CRUD-операцій (створення, перегляд, редагування, видалення) як для занять, так і для нотаток, а також можливість пошуку та фільтрації занять. Кожен варіант використання представляє собою окрему функціональну можливість, яку надає система.

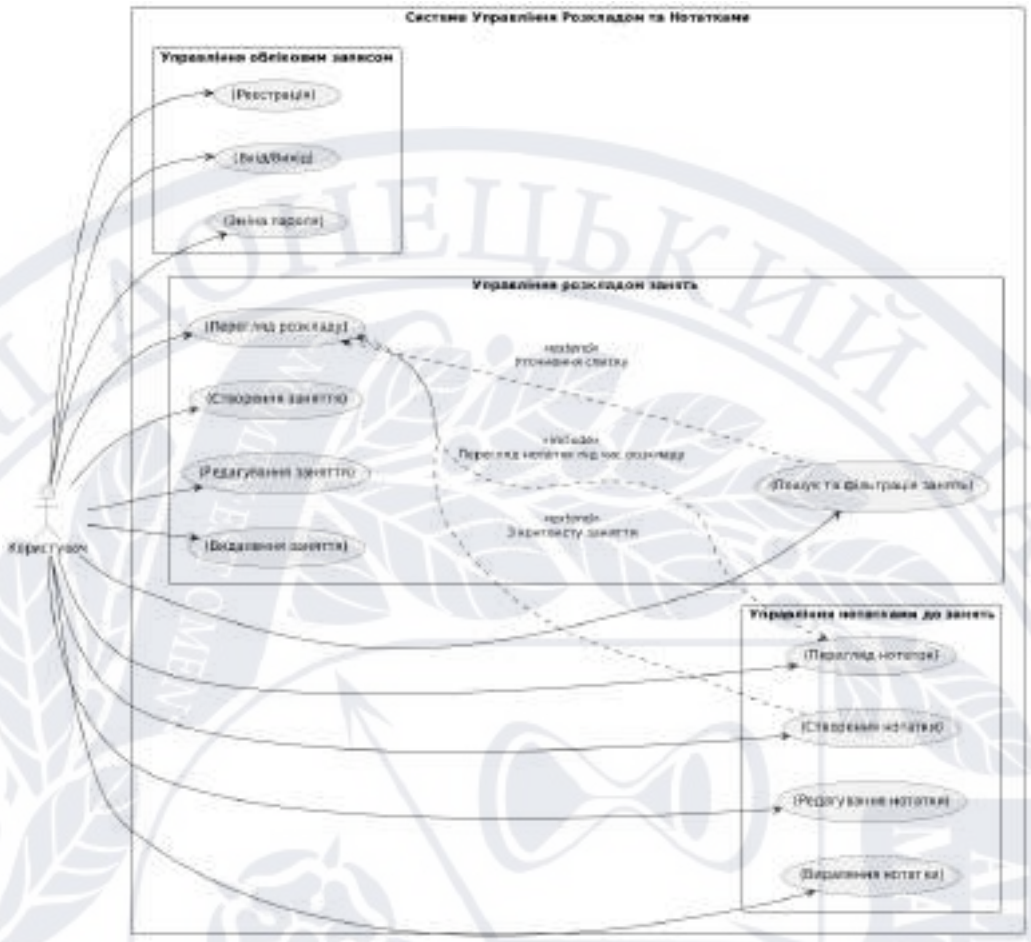

Рисунок 2.8 – Use Case діаграма взаємодії користувача з веб-додатком

Такий підхід до проєктування UI/UX, що поєднує стандартні практики веб-розробки, використання відомого CSS-фреймворку та логічну організацію потоків взаємодії, має забезпечити позитивний досвід для кінцевих користувачів веб-додатку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

3.1. Налаштування робочого середовища розробки

Ефективна розробка будь-якого програмного продукту починається з належного налаштування робочого середовища. Для реалізації веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток було обрано інтегроване середовище розробки (IDE) Microsoft Visual Studio [23], зокрема версію, що підтримує розробку на платформі .NET 8. Вибір Visual Studio обґрунтований її потужними інструментами для розробки, відладки, тестування та управління проєктами на C# та ASP.NET Core [24], а також тісною інтеграцією з системою контролю версій Git.

Першим кроком було встановлення актуальної версії Visual Studio з необхідними робочими навантаженнями, зокрема "ASP.NET and web development" та ".NET desktop development" (якщо планується робота з іншими типами проєктів .NET). Також було встановлено .NET 8 SDK, який є основою для розробки на ASP.NET Core 8 [25].

Після встановлення необхідного програмного забезпечення було створено новий проєкт ASP.NET Core MVC. При створенні проєкту було обрано шаблон "ASP.NET Core Web App (Model-View-Controller)" з використанням .NET 8 як цільової платформи. Важливим аспектом початкового налаштування було обрання типу автентифікації "Individual Accounts", що автоматично інтегрувало систему ASP.NET Core Identity в проєкт, забезпечивши базовий функціонал для реєстрації, входу та управління обліковими записами користувачів. Також було активовано опцію "Configure for HTTPS" для забезпечення безпечного з'єднання під час розробки та подальшого розгортання.

Після створення проєкту було налаштовано систему управління NuGet-пакетами для додавання необхідних бібліотек. Ключовими пакетами, встановленими на цьому етапі, були Microsoft.EntityFrameworkCore.Sqlite для роботи з базою даних SQLite, Microsoft.EntityFrameworkCore.Tools для виконання команд міграцій Entity Framework Core [26] (таких як Add-Migration та Update-Database) та Microsoft.EntityFrameworkCore.Design для підтримки інструментів проєктування EF Core. Версії цих пакетів були узгоджені з версією .NET 8 (наприклад, 8.0.5) для забезпечення сумісності та уникнення конфліктів.

Структура проєкту, згенерована Visual Studio для ASP.NET Core MVC, включає стандартні папки, такі як Controllers, Models, Views, wwwroot для статичних файлів, та Data для класів, пов'язаних з доступом до даних, включаючи ApplicationDbContext та ApplicationUser. Також було створено папку Services для розміщення класів сервісів, що інкапсулюють бізнес-логіку. На Рис. 3.1. продемонстровано структуру проєкту SheduleAppMVC в Solution Explorer Visual Studio після початкового налаштування та додавання основних папок і файлів.

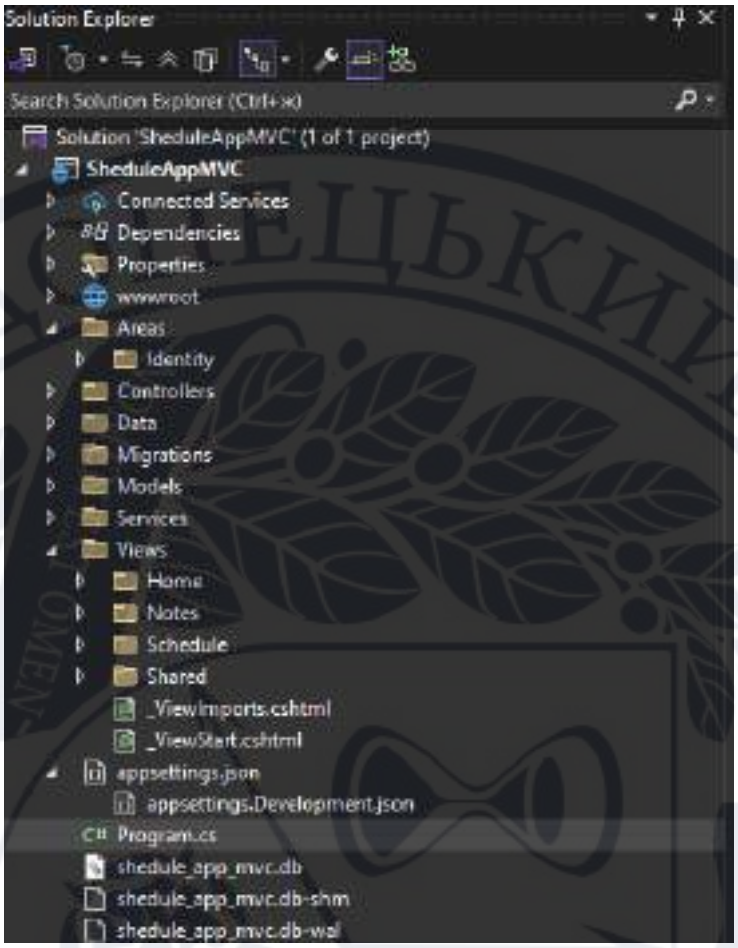

Рисунок 3.1 – Структура проєкту веб-додатку в Visual Studio

Для зберігання даних було налаштовано рядок підключення до бази даних SQLite [27] у файлі appsettings.json, вказавши ім'я файлу бази даних, наприклад, shedule_app_mvc.db.

У файлі Program.cs було зареєстровано ApplicationDbContext з використанням провайдера SQLite (options.UseSqlite(connectionString)), а також налаштовано сервіси ASP.NET Core Identity для роботи з кастомним класом користувача ApplicationUser.

Для контролю версій коду було ініціалізовано локальний Git-репозиторій, що дозволяє відстежувати зміни, створювати гілки для розробки окремих функцій та повертатися до попередніх станів коду за потреби.

Налаштоване робоче середовище забезпечило всі необхідні інструменти та конфігурації для подальшої розробки функціоналу веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток.

3.2. Реалізація ключових функціональних модулів

Після налаштування робочого середовища та проєктування архітектури, наступним етапом є безпосередня реалізація ключових функціональних модулів веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток. Реалізація відбувалася поетапно, з фокусом на забезпеченні основного функціоналу та поступовим додаванням розширених можливостей.

Основою для зберігання даних слугують моделі сутностей, визначені за допомогою класів C# та атрибутів Entity Framework Core для мапування на таблиці бази даних SQLite [28]. Наприклад, модель Note (Рис. 3.2), що представляє персоналізовану нотатку, включає поля для ідентифікатора, вмісту, дат створення та оновлення, а також зовнішні ключі для зв'язку з користувачем (UserId) та заняттям (ScheduledEventId). Атрибути [Key], [Required], [StringLength] та [ForeignKey] використовуються для визначення первинних ключів, обов'язковості полів, обмежень на довжину рядків та зовнішніх ключів відповідно.

```
38 references
public class Note
{
    [Key]
    11 references
    public int Id { get; set; }

    [Required(ErrorMessage = "Вміст нотатки є обов'язковим.")]
    [StringLength(2000, ErrorMessage = "Нотатка не може перевищувати 2000 символів.")]
    [Display(Name = "Вміст нотатки")]
    8 references
    public string Content { get; set; } = string.Empty;

    [Display(Name = "Дата створення")]
    6 references
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    [Display(Name = "Дата останнього оновлення")]
    10 references
    public DateTime LastUpdatedAt { get; set; } = DateTime.UtcNow;

    // Зв'язок з користувачем (власником цієї нотатки)
    [Required]
    18 references
    public string UserId { get; set; } = string.Empty;

    [ForeignKey("UserId")]
    1 reference
    public virtual ApplicationUser? User { get; set; }

    // Зв'язок з заняттям (нотатка може бути прив'язана до заняття, але не обов'язково)
    [Display(Name = "Заняття")]
    40 references
    public int? ScheduledEventId { get; set; }

    [ForeignKey("ScheduledEventId")]
    1 reference
    public virtual ScheduledEvent? ScheduledEvent { get; set; }
}
```

Рисунок 3.2 – Фрагмент коду моделі Note

Реалізація модуля управління користувачами базується на системі ASP.NET Core Identity [29]. Було використано стандартні сторінки для реєстрації, входу, виходу та управління профілем, з адаптацією класу ApplicationUser для додавання навігаційних властивостей до пов'язаних сутностей розкладу та нотаток. Це забезпечило надійний та безпечний механізм автентифікації та авторизації.

Модуль управління розкладом занять реалізовано через ScheduleController, який обробляє HTTP-запити від користувача. Контролер взаємодіє з сервісним шаром, представленим ScheduleService (реалізація

IScheduleService), для виконання бізнес-логіки та операцій з даними. Наприклад, для створення нового заняття, контролер отримує дані з форми, валідує їх, встановлює UserId поточного користувача та викликає відповідний метод сервісу для збереження заняття в базі даних. Для відображення даних використовуються Razor Views, які отримують моделі даних або моделі представлення від контролера.

Аналогічно реалізовано модуль управління персоналізованими нотатками. NotesController відповідає за обробку запитів, пов'язаних з нотатками. Наприклад, дія CreateForEvent (Рис. 3.3) у NotesController готує дані для відображення форми створення нової нотатки, прив'язаної до конкретного заняття. Вона отримує scheduledEventId як параметр, перевіряє належність цього заняття поточному користувачеві через ScheduleService, і передає у представлення новий об'єкт Note з уже встановленим ScheduledEventId.

```
// GET: Notes/CreateForEvent/5
0 references
public async Task<IActionResult> CreateForEvent(int scheduledEventId)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userId))
    {
        return Challenge();
    }

    var anEvent = await _scheduleService.GetScheduledEventByIdAsync(scheduledEventId, userId);
    if (anEvent == null)
    {
        TempData["ErrorMessage"] = "Заняття не знайдено або у вас немає до нього доступу.";
        return RedirectToAction("Index", "Schedule");
    }

    var note = new Note
    {
        ScheduledEventId = scheduledEventId,
        // UserId = userId // Встановимо в POST-дії
    };
    ViewBag.ScheduledEventName = anEvent.SubjectName;
    // Передаємо ім'я представлення "Create", оскільки у нас може бути одне представлення для
    return View("Create", note);
}
```

Рисунок 3.3 – Фрагмент коду методу CreateForEvent з NotesController

Сервісний шар, наприклад NoteService, інкапсулює логіку взаємодії з базою даних через ApplicationDbContext. Метод GetNotesForEventAsync (Рис. 3.4) у NoteService отримує ідентифікатор заняття та користувача, перевіряє належність заняття користувачеві, а потім вибирає з бази даних усі нотатки, що відповідають цим критеріям, сортуючи їх за датою останнього оновлення.

Використання AsNoTracking() для запиту на перевірку існування заняття оптимізує продуктивність, оскільки Entity Framework Core не відстежує зміни для цієї сутності [30].

```
2 references
public async Task<List<Note>> GetNotesForEventAsync(int scheduledEventId, string userId)
{
    if (string.IsNullOrEmpty(userId))
    {
        return new List<Note>();
    }

    var anEvent = await _context.ScheduledEvents
                                .AsNoTracking()
                                .FirstOrDefaultAsync(e => e.Id == scheduledEventId && e.UserId == userId);
    if (anEvent == null)
    {
        return new List<Note>(); // Подія не знайдена або не належить користувачеві
    }

    return await _context.Notes
                         .Where(n => n.ScheduledEventId == scheduledEventId && n.UserId == userId)
                         .OrderByDescending(n => n.LastUpdatedAt)
                         .ToListAsync();
}
```

Рисунок 3.4 – Фрагмент коду методу GetNotesForEventAsync з NoteService

Для забезпечення зручного користувацького досвіду на сторінці відображення розкладу (Views/Schedule/Index.cshtml) було реалізовано функціонал пошуку та фільтрації занять. Це включає додавання форми з полями для введення пошукового терміну, вибору типу заняття та діапазону дат. Відповідна дія Index у ScheduleController була модифікована для приймання цих параметрів фільтрації та передачі їх у метод GetScheduledEventsFilteredAsync сервісу ScheduleService. Цей метод динамічно будує LINQ-запит [30] до бази даних, застосовуючи умови Where на основі наданих фільтрів, що дозволяє ефективно відбирати потрібні заняття. Інтерфейс користувача оновлюється для відображення

відфільтрованих результатів, а також зберігає поточні значення фільтрів у формі для зручності користувача. Також було реалізовано візуальне оформлення рядків таблиці розкладу відповідно до обраного користувачем кольору для кожного заняття, що покращує візуальне сприйняття та організацію інформації.

Реалізація CRUD-операцій (Create, Read, Update, Delete) для занять та нотаток супроводжувалася створенням відповідних представлень Razor (.cshtml) з формами для введення даних, таблицями для їх відображення та елементами управління (кнопки, посилання). Валідація даних здійснюється як на стороні клієнта за допомогою jQuery Unobtrusive Validation, так і на стороні сервера за допомогою атрибутів валідації в моделях та перевірок у контролерах. Повідомлення про успішні операції або помилки відображаються користувачеві за допомогою TempData [31].

Реалізація ключових функціональних модулів забезпечила створення працездатного прототипу веб-додатку, що відповідає основним поставленим вимогам щодо управління розкладом занять та персоналізованими нотатками.

3.3. Проведення функціонального тестування

Після завершення етапу реалізації ключових функціональних модулів веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток було проведено функціональне тестування. Метою тестування була перевірка коректності роботи реалізованого функціоналу відповідно до сформульованих вимог, виявлення та усунення можливих дефектів. Тестування проводилося вручну шляхом виконання типових сценаріїв взаємодії користувача з системою.

Процес тестування охоплював наступні основні аспекти. Першочергово перевірялася працездатність системи управління користувачами. Тестування розпочалося з головної сторінки веб-додатку, де користувачеві пропонуються

опції реєстрації нового облікового запису або входу до системи, як показано на Рис. 3.5.

Рисунок 3.5 – Головна сторінка веб-додатку з можливістю реєстрації та входу

Було успішно створено тестовий обліковий запис користувача. Перевірялася валідація даних на формі реєстрації, зокрема обов'язковість заповнення полів, коректність формату електронної пошти та співпадіння введених паролів. Після успішної реєстрації здійснювався вхід до системи з використанням новостворених облікових даних, що продемонстровано на Рис. 3.6. Функціонал виходу із системи та базові можливості управління профілем користувача (наприклад, зміна пароля через стандартні сторінки ASP.NET Core Identity) [32] також були перевірені та функціонували коректно.

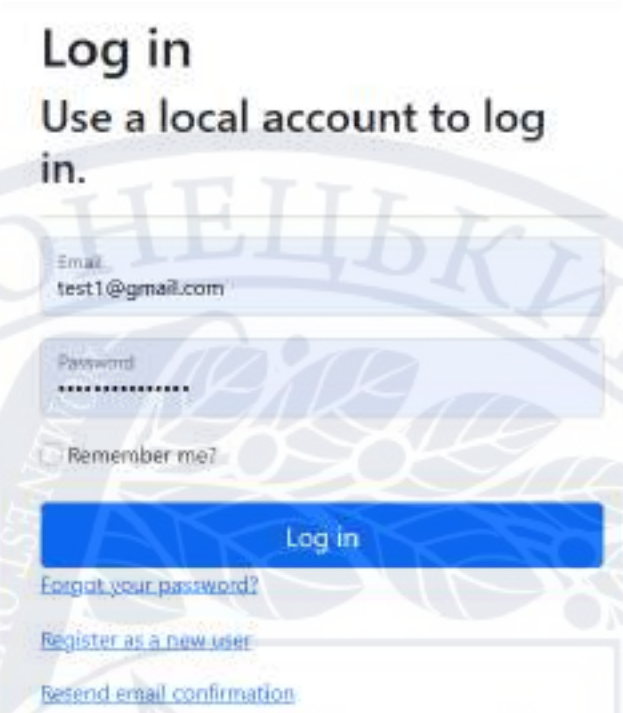

Рисунок 3.6 – Форма входу до системи

Наступним етапом було тестування основного модуля – управління розкладом занять. Після успішної автентифікації користувач отримував доступ до сторінки "Мій Розклад". Було протестовано всі CRUD-операції для занять. Створення нового заняття відбувалося через відповідну форму, де перевірялася валідація введених даних, таких як назва предмету, тип заняття, час початку та закінчення (з умовою, що час закінчення має бути пізнішим за час початку). Після успішного збереження, нове заняття коректно відображалося у списку розкладу, а користувач отримував повідомлення про успішне створення, як показано на Рис. 3.7.

Рисунок 3.7 – Сторінка "Мій Розклад" після успішного створення заняття

Перегляд списку занять поточного користувача, сортування за часом початку та візуальне оформлення рядків таблиці відповідно до обраного кольору заняття функціонували належним чином. Також перевірялася можливість переходу на сторінку деталей конкретного заняття. На цій сторінці (Рис. 3.8) відображалася вся введена інформація про заняття, а також передбачалася секція для пов'язаних нотаток та кнопки для подальших дій, таких як редагування заняття або додавання нотаток.

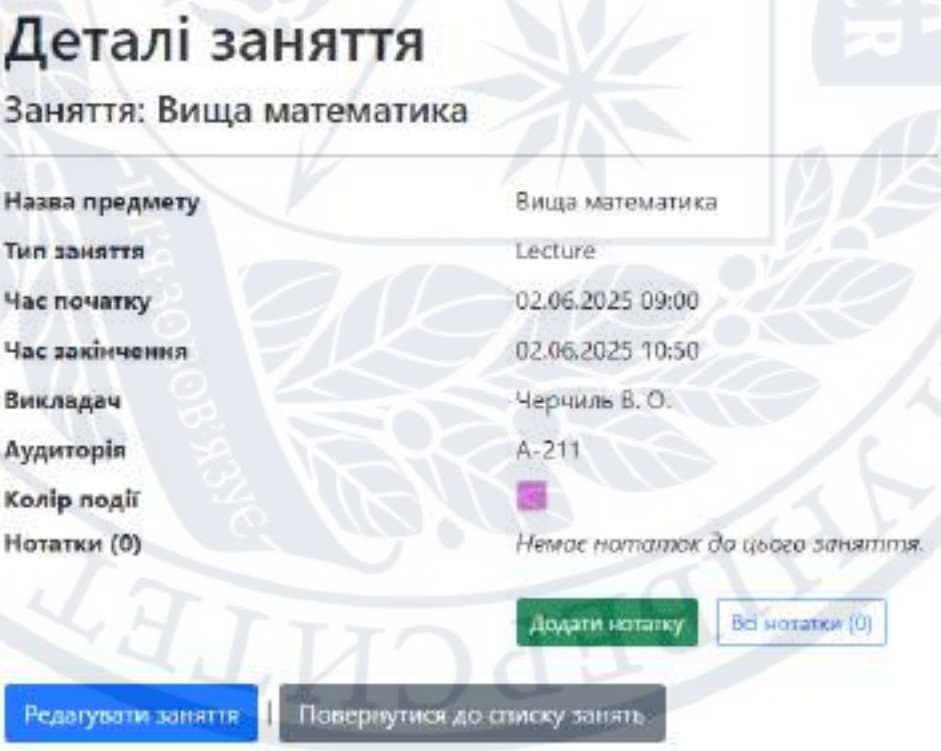

Рисунок 3.8 – Сторінка деталей заняття

Функції редагування та видалення існуючих занять також були ретельно протестовані. Форма редагування коректно заповнювалася даними обраного заняття, дозволяла вносити зміни та зберігати їх. Процес видалення включав сторінку підтвердження, після чого заняття успішно видалялося з бази даних та зникало зі списку розкладу.

Далі тестувався модуль управління персоналізованими нотатками. Зі сторінки деталей заняття користувач мав можливість перейти до створення нової нотатки для цього заняття. Форма створення нотатки (Рис. 3.9) дозволяла ввести текстовий вміст нотатки.

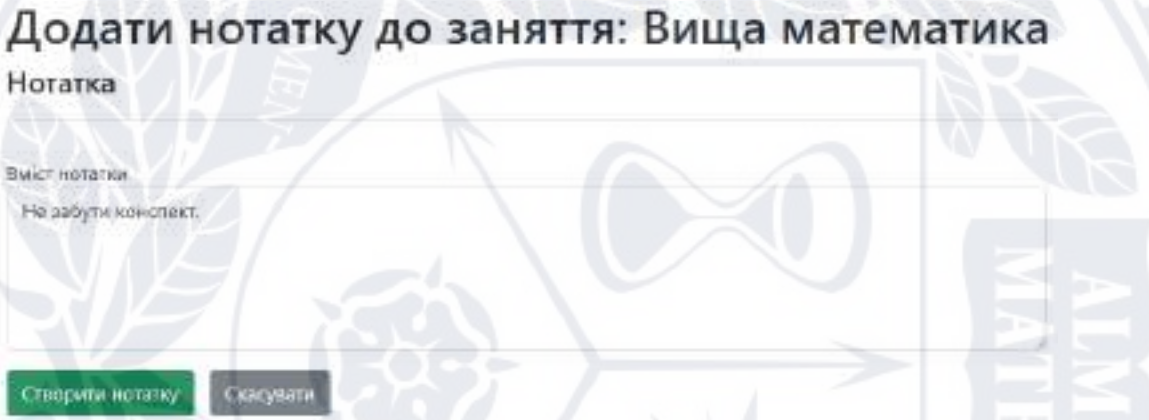

Рисунок 3.9 – Форма додавання нової нотатки до заняття

Після збереження, нотатка коректно прив'язувалася до відповідного заняття та відображалася у списку нотаток для нього, як це показано на Рис. 3.10. Також було перевірено можливості редагування та видалення існуючих нотаток.

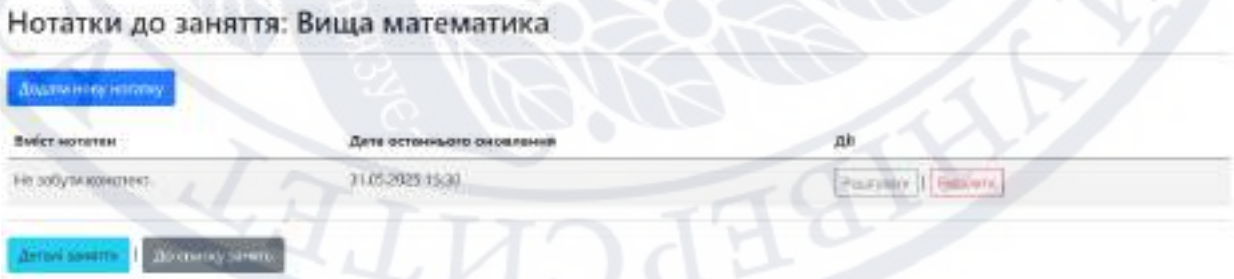

Рисунок 3.10 – Сторінка зі списком нотаток для конкретного заняття

Функціонал пошуку та фільтрації на сторінці "Мій Розклад" перевірявся шляхом введення різних пошукових запитів, вибору типів занять та

встановлення діапазонів дат. Система коректно фільтрувала та відображала список занять відповідно до заданих критеріїв, а також дозволяла скидати фільтри для повернення до повного списку.

Під час функціонального тестування також зверталася увага на нефункціональні аспекти, такі як швидкість завантаження сторінок, коректність відображення інтерфейсу на різних пристроях (базове тестування адаптивності завдяки Bootstrap), інтуїтивність навігації та зрозумілість повідомлень для користувача. Було виявлено та виправлено низку дрібних недоліків, пов'язаних переважно з валідацією даних на формах та коректним відображенням інформації після виконання операцій. Загалом, тестування підтвердило, що реалізовані ключові функціональні модулі веб-додатку працюють відповідно до поставлених вимог, забезпечуючи користувачеві можливість ефективно управляти своїм розкладом занять та персональними нотатками.

3.4. Розробка короткої інструкції для користувачів веб-додатку

Для забезпечення ефективного та комфортного використання веб-додатку для управління розкладом занять із можливістю персоналізованих нотаток розроблено наступну коротку інструкцію, що охоплює основні функціональні можливості системи.

Початок роботи: Для початку роботи з веб-додатком необхідно перейти на його головну сторінку (Рис. 3.5). Новим користувачам слід пройти процедуру реєстрації, натиснувши кнопку "Зареєструватися" та заповнивши відповідні поля (електронна пошта, пароль). Після успішної реєстрації або якщо обліковий запис вже існує, необхідно здійснити вхід до системи, натиснувши кнопку "Увійти" та ввівши свої облікові дані (Рис. 3.6). Успішний вхід надасть доступ до персоналізованого функціоналу управління розкладом.

Управління розкладом занять: Після входу користувач потрапляє на сторінку "Мій Розклад" (Рис. 3.7), де відображаються всі його заплановані заняття. Для додавання нового заняття необхідно натиснути кнопку "Додати заняття". Відкриється форма, де потрібно заповнити інформацію про заняття: назву предмету, тип (лекція, практика тощо), час початку та закінчення, ім'я викладача (необов'язково), аудиторію (необов'язково) та обрати колір для візуального маркування. Після заповнення форми та натискання кнопки "Створити", нове заняття з'явиться у загальному списку розкладу. Кожне заняття у списку має кнопки для швидких дій: "Редагувати" (для внесення змін до існуючого заняття), "Деталі" (для перегляду повної інформації про заняття та пов'язаних нотаток, як показано на Рис. 3.8), "Нотатки" (для переходу до управління нотатками цього заняття) та "Видалити" (для видалення заняття після підтвердження). Для зручності пошуку потрібних занять на сторінці "Мій Розклад" передбачена панель "Фільтри та Пошук", яка дозволяє відбирати заняття за назвою, викладачем, аудиторією, типом заняття та діапазоном дат.

Управління персоналізованими нотатками: До кожного заняття користувач може додавати персональні нотатки. Для цього зі сторінки деталей заняття (Рис. 3.8) або зі сторінки списку нотаток для заняття (Рис. 3.10) необхідно натиснути кнопку "Додати нову нотатку". Відкриється форма (Рис. 3.9), де можна ввести текст нотатки та зберегти її. Створені нотатки відображатимуться у списку, прив'язаному до відповідного заняття. Кожну нотатку можна редагувати або видаляти за допомогою відповідних кнопок дій.

Завершення роботи: Для виходу з системи необхідно скористатися посиланням "Logout", яке зазвичай розташоване у верхній частині сторінки поруч з іменем користувача.

Ця коротка інструкція охоплює основні сценарії використання веб-додатку. Інтерфейс розроблено таким чином, щоб бути максимально інтуїтивно зрозумілим, тому користувачі з базовими навичками роботи з веб-додатками зможуть легко освоїти всі його можливості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. – [Чинний від 2017-07-01]. – К.: ДП "УкрНДНЦ", 2016. – 26 с.
2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. – [Чинний від 2016-07-01]. – К.: ДП "УкрНДНЦ", 2016. – 16 с.
3. Архітектурні шаблони програмного забезпечення: навч. посібник / О. В. Коротун, Т. А. Вакалюк, В. М. Франчук. – Житомир: Вид-во ЖДУ ім. І. Франка, 2021. – 198 с.
4. Бази даних та інформаційні системи: підручник / В. В. Литвин, В. В. Пасічник, Ю. В. Яцишин. – Львів: Видавництво Львівської політехніки, 2022. – 500 с.
5. Веб-програмування: навчальний посібник / І. О. Завадський, Т. В. Прокопенко. – К.: Видавничий дім "Києво-Могилянська академія", 2023. – 340 с.
6. Гавилюк А. Bootstrap 5. Розробка адаптивних веб-сайтів / А. Гавилюк. – Харків: Фоліо, 2022. – 312 с.
7. Гаврилова І. В. Організація робочого місця програміста: ергономічні аспекти / І. В. Гаврилова // Охорона праці. – 2022. – № 4. – С. 28-35.
8. Глибовець М. М. Основи програмування / М. М. Глибовець, С. С. Гороховський, О. В. Олецький. – К.: Видавничий дім "Києво-Могилянська академія", 2023. – 624 с.
9. Гриценко В. І. Інформаційні технології: сучасний стан та перспективи / В. І. Гриценко // Вісник НАН України. – 2022. – № 7. – С. 44-53.
10. Економіка програмного забезпечення: навч. посібник / О. М. Пасічник, О. В. Пасічник, І. В. Стецiв. – Львів: Магнолія 2006, 2021. – 264 с.

11. Ергономіка робочого місця користувача персонального комп'ютера: методичні рекомендації / уклад. М. О. Полумацканич. – Тернопіль: ТНТУ, 2020. – 48 с.
12. Завадський І. О. Основи веб-програмування: JavaScript і Node.js / І. О. Завадський. – К.: Видавничий дім "Києво-Могилянська академія", 2022. – 456 с.
13. Інформаційні технології в освіті: монографія / за ред. О. М. Спіріна, Ю. О. Жука. – К.: ЦП "Компринт", 2022. – 486 с.
14. Карпенко С. Г. Інформаційні системи і технології: навч. посібник / С. Г. Карпенко, В. В. Попов, Ю. А. Тарнавський. – К.: МАУП, 2021. – 544 с.
15. Ковальчук А. М. Основи програмування веб-додатків: підручник / А. М. Ковальчук. – Вінниця: ВНТУ, 2023. – 412 с.
16. Козловський В. О. Сучасні технології розробки програмного забезпечення / В. О. Козловський, Ю. В. Триус. – Черкаси: ЧНУ, 2021. – 380 с.
17. Коротун О. В. Хмарні технології в освіті: навч. посібник / О. В. Коротун. – Житомир: Вид-во ЖДУ ім. І. Франка, 2020. – 242 с.
18. Кривий С. Л. Дискретна математика: підручник / С. Л. Кривий, Г. Г. Цегелик. – Львів: Видавництво Львівської політехніки, 2021. – 568 с.
19. Лукін В. М. Охорона праці в ІТ-галузі: навчальний посібник / В. М. Лукін, О. В. Березуцький. – Харків: НТУ "ХПІ", 2021. – 216 с.
20. Мельник Р. А. Розробка веб-додатків на платформі .NET / Р. А. Мельник. – Львів: Видавництво Львівської політехніки, 2023. – 340 с.
21. Методологія розробки програмного забезпечення: підручник / В. С. Харченко, В. В. Скляр, О. М. Тарасюк. – Харків: НАУ "ХАІ", 2022. – 428 с.
22. Нікітченко М. С. Основи програмування: підручник / М. С. Нікітченко, С. С. Шкільняк. – К.: ВПЦ "Київський університет", 2020. – 287 с.
23. Основи веб-технологій: навчальний посібник / П. І. Жежнич, І. Ю. Шубин. – Львів: Видавництво Львівської політехніки, 2022. – 336 с.

24. Павлова О. О. Проектування баз даних: навчальний посібник / О. О. Павлова, В. М. Чернишов. – Харків: ХНУРЕ, 2021. – 272 с.
25. Пасічник В. В. Організація баз даних та знань / В. В. Пасічник, В. А. Резніченко. – К.: Видавнича група BHV, 2020. – 384 с.
26. Петренко А. І. Основи автоматизованого проектування програмного забезпечення / А. І. Петренко, О. А. Семенов. – К.: КПІ ім. Ігоря Сікорського, 2022. – 318 с.
27. Приходько С. Б. Управління ІТ-проектами: навч. посібник / С. Б. Приходько, О. В. Малахов. – Одеса: Фенікс, 2021. – 286 с.
28. Проектування інформаційних систем: підручник / за ред. М. З. Згуровського. – К.: НТУУ "КПІ", 2023. – 512 с.
29. Редько В. Н. Бази даних та інформаційні системи: підручник / В. Н. Редько, Ю. Й. Батюк, Н. Е. Кунанець. – Львів: Видавництво Львівської політехніки, 2022. – 484 с.
30. Рижко О. М. Інформаційні системи та технології: підручник / О. М. Рижко, Л. М. Рибак, М. М. Пасічник. – Львів: Новий Світ-2000, 2022. – 444 с.
31. Савчук Т. О. Безпека веб-додатків: навчальний посібник / Т. О. Савчук, Ю. В. Бойко. – Вінниця: ВНТУ, 2022. – 198 с.
32. Сучасні інформаційні системи і технології: навчальний посібник / за ред. В. С. Пономаренка. – Харків: ХНЕУ, 2021. – 492 с.
33. Триус Ю. В. Комп'ютерно-орієнтовані методичні системи навчання: монографія / Ю. В. Триус. – Черкаси: Брама-Україна, 2022. – 400 с.
34. Шевченко В. Л. Основи веб-дизайну: навч. посібник / В. Л. Шевченко. – К.: КНЕУ, 2021. – 298 с.
35. Шинкаренко В. І. Основи програмної інженерії: підручник / В. І. Шинкаренко, О. М. Куротчин. – Дніпро: НГУ, 2022. – 356 с.
36. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages / A. Freeman. – 9th ed. – Berkeley, CA: Apress, 2022. – 1299 p.

37. Lock A. ASP.NET Core in Action / A. Lock. – 2nd ed. – Greenwich, CT: Manning Publications, 2021. – 832 p.
38. Price M. J. C# 11 and .NET 7 – Modern Cross-Platform Development: Build apps, websites, and services / M. J. Price. – 7th ed. – Birmingham, UK: Packt Publishing, 2022. – 818 p.
39. Smith J. Entity Framework Core in Action / J. Smith. – 2nd ed. – Greenwich, CT: Manning Publications, 2021. – 656 p.
40. Troelsen A. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming / A. Troelsen, P. Japikse. – 11th ed. – Berkeley, CA: Apress, 2022. – 1638 p.

ДОДАТКИ

ДОДАТОК А

Лістинг коду «NotesController.cs»

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore; // Для DbUpdateConcurrencyException
using SheduleAppMVC.Data; // Для ApplicationUser
using SheduleAppMVC.Models; // Для Note, ScheduledEvent
using SheduleAppMVC.Services; // Для INoteService, IScheduleService
using System.Security.Claims;
using System.Threading.Tasks;

namespace SheduleAppMVC.Controllers
{
    [Authorize]
    public class NotesController : Controller
    {
        private readonly INoteService _noteService;
        private readonly IScheduleService _scheduleService;
        private readonly UserManager<ApplicationUser> _userManager;

        public NotesController(INoteService noteService, IScheduleService scheduleService, UserManager<ApplicationUser> userManager)
        {
            _noteService = noteService;
            _scheduleService = scheduleService;
            _userManager = userManager;
        }

        // GET: Notes/ForEvent/5
        public async Task<IActionResult> IndexForEvent(int scheduledEventId)
        {
            var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                return Challenge();
            }

            var anEvent = await _scheduleService.GetScheduledEventByIdAsync(scheduledEventId, userId);
            if (anEvent == null)
            {
```

```
                TempData["ErrorMessage"] = "Заняття не знайдено
або у вас немає до нього доступу.";
                return RedirectToAction("Index", "Schedule");
            }

            ViewBag.ScheduledEventId = scheduledEventId;
            ViewBag.ScheduledEventName = anEvent.SubjectName;
            var notes = await
_noteService.GetNotesForEventAsync(scheduledEventId, userId);
            return View(notes);
        }

        // GET: Notes/CreateForEvent/5
        public async Task<IActionResult> CreateForEvent(int
scheduledEventId)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                return Challenge();
            }

            var anEvent = await
_scheduleService.GetScheduledEventByIdAsync(scheduledEventId,
userId);
            if (anEvent == null)
            {
                TempData["ErrorMessage"] = "Заняття не знайдено
або у вас немає до нього доступу.";
                return RedirectToAction("Index", "Schedule");
            }

            var note = new Note
            {
                ScheduledEventId = scheduledEventId,
                // UserId = userId // Встановимо в POST-дії
            };
            ViewBag.ScheduledEventName = anEvent.SubjectName;
            // Передаємо ім'я представлення "Create", оскільки у
нас може бути одне представлення для створення нотаток
            return View("Create", note);
        }


        // POST: Notes/Create або Notes/CreateForEvent
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Content,ScheduledEventId")] Note note) // UserId не
має приходити з форми
        {
```

```
            var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                ModelState.AddModelError(string.Empty, "Не вдалося визначити користувача.");
                // Потрібно підготувати ViewBag.ScheduledEventName, якщо повертаємо View
                if (note.ScheduledEventId.HasValue)
                {
                    var tempEvent = await _scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventId.Value, User.FindFirstValue(ClaimTypes.NameIdentifier) ?? ""); // Потрібен userId, але якщо він null, то подія все одно не знайдеться
                    ViewBag.ScheduledEventName = tempEvent?.SubjectName;
                }
                return View(note);
            }

            note.UserId = userId; // Встановлюємо UserId ДО перевірки ModelState.IsValid

            // Очищаємо помилку для UserId, оскільки ми встановили його вручну
            if (ModelState.ContainsKey(nameof(Note.UserId)))
            {
                ModelState.Remove(nameof(Note.UserId));
            }

            // Перевірка, чи подія (якщо вказана) належить користувачеві
            if (note.ScheduledEventId.HasValue)
            {
                var anEvent = await _scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventId.Value, userId);
                if (anEvent == null)
                {
                    ModelState.AddModelError("ScheduledEventId", "Вказане заняття не знайдено або недоступне.");
                }
            }

            if (ModelState.IsValid)
            {
                var createdNote = await _noteService.AddNoteAsync(note);
                if (createdNote != null)
                {
```

```
                    TempData["SuccessMessage"] = "Нотатку успішно
створено.";
                    if (note.ScheduledEventId.HasValue)
                    {
                        return
RedirectToAction(nameof(IndexForEvent), new { scheduledEventId =
note.ScheduledEventId.Value });
                    }
                    // TODO: Додати логіку для повернення, якщо це
була загальна нотатка користувача (наприклад, на сторінку всіх
нотаток)
                    return RedirectToAction("Index", "Schedule");
// Тимчасово
                }
                else
                {
                    ModelState.AddModelError(string.Empty,    "Не
вдалося створити нотатку. Спробуйте ще раз.");
                }
            }

            // Якщо ModelState не валідний, потрібно знову
отримати назву події для ViewBag
            if (note.ScheduledEventId.HasValue)
            {
                var          anEvent          =          await
_scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventI
d.Value, userId);
                ViewBag.ScheduledEventName                    =
anEvent?.SubjectName;
            }
            return View(note);
        }

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

            var                        userId                    =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                return Challenge();
            }

            var             note             =             await
_noteService.GetNoteByIdAsync(id.Value, userId);
            if (note == null)
```

```
            {
                return NotFound(); // Нотатка не знайдена або не
належить користувачеві
            }

            if (note.ScheduledEventId.HasValue)
            {
                var anEvent = await
_scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventI
d.Value, userId);
                ViewBag.ScheduledEventName =
anEvent?.SubjectName;
            }
            return View(note);
        }

        // POST: Notes/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,Content,ScheduledEventId,UserId,CreatedAt")] Note note)
        {
            if (id != note.Id)
            {
                return NotFound();
            }

            var currentUserId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            // Переконуємося, що UserId, який прийшов з форми,
співпадає з UserId поточного користувача
            // і що він не порожній. Це важливо для безпеки та
цілісності даних.
            if (string.IsNullOrEmpty(currentUserId) ||
note.UserId != currentUserId)
            {
                return Forbid();
            }
            // Завжди встановлюємо/перепризначаємо UserId з
поточного автентифікованого користувача для безпеки
            note.UserId = currentUserId;

            // Очищаємо помилку для UserId, оскільки ми встановили
його вручну
            if (ModelState.ContainsKey(nameof(Note.UserId)))
            {
                ModelState.Remove(nameof(Note.UserId));
            }

            // Перевірка, чи подія (якщо вказана) належить
користувачеві
            if (note.ScheduledEventId.HasValue)
```

```
            {
                var anEvent = await
_scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventI
d.Value, currentUserId);
                if (anEvent == null)
                {
                    ModelState.AddModelError("ScheduledEventId",
"Вказане заняття не знайдено або недоступне.");
                }
            }

            if (ModelState.IsValid)
            {
                try
                {
                    // CreatedAt має бути збережено з
оригінального запису, не з форми
                    var existingNote = await
_noteService.GetNoteByIdAsync(id, currentUserId);
                    if (existingNote != null)
                    {
                        note.CreatedAt = existingNote.CreatedAt;
                    }
                    else
                    {
                        return NotFound(); // Якщо раптом нотатку
видалили під час редагування
                    }

                    bool success = await
_noteService.UpdateNoteAsync(note, currentUserId);
                    if (success)
                    {
                        TempData["SuccessMessage"] = "Нотатку
успішно оновлено.";
                        if (note.ScheduledEventId.HasValue)
                        {
                            return
RedirectToAction(nameof(IndexForEvent), new { scheduledEventId =
note.ScheduledEventId.Value });
                        }
                        return RedirectToAction("Index",
"Schedule"); // Тимчасово
                    }
                    else
                    {
                        ModelState.AddModelError(string.Empty,
"Не вдалося оновити нотатку. Спробуйте ще раз.");
                    }
                }
                catch (DbUpdateConcurrencyException)
                {
```

```
                    if (await
_noteService.GetNoteByIdAsync(note.Id, currentUserId) == null)
                    {
                        return NotFound();
                    }
                    else
                    {
                        ModelState.AddModelError(string.Empty,
"Запис був змінений іншим користувачем. Будь ласка, оновіть
сторінку та спробуйте ще раз.");
                    }
                }
            }

            if (note.ScheduledEventId.HasValue)
            {
                var anEvent = await
_scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventI
d.Value, currentUserId);
                ViewBag.ScheduledEventName =
anEvent?.SubjectName;
            }
            return View(note);
        }

        // GET: Notes/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                return Challenge();
            }

            var note = await
_noteService.GetNoteByIdAsync(id.Value, userId);
            if (note == null)
            {
                return NotFound();
            }
            if (note.ScheduledEventId.HasValue)
            {
                var anEvent = await
_scheduleService.GetScheduledEventByIdAsync(note.ScheduledEventI
d.Value, userId);
                ViewBag.ScheduledEventName =
anEvent?.SubjectName;
```

```
            }
            return View(note);
        }

        // POST: Notes/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var                    userId                    =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(userId))
            {
                return Challenge();
            }

            var          noteToDelete          =          await
_noteService.GetNoteByIdAsync(id, userId);
            if (noteToDelete == null)
            {
                TempData["ErrorMessage"] = "Не вдалося знайти
нотатку для видалення.";
                return RedirectToAction("Index", "Schedule");
            }

            int?              scheduledEventId                =
noteToDelete.ScheduledEventId;

            bool success = await _noteService.DeleteNoteAsync(id,
userId);
            if (success)
            {
                TempData["SuccessMessage"] = "Нотатку успішно
видалено.";
            }
            else
            {
                TempData["ErrorMessage"] = "Не вдалося видалити
нотатку.";
            }

            if (scheduledEventId.HasValue)
            {
                return   RedirectToAction(nameof(IndexForEvent),
new { scheduledEventId = scheduledEventId.Value });
            }
            return RedirectToAction("Index", "Schedule");
        }
    }
}
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____
_____
_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)