Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Веб-орієнтований додаток обліку відвідування занять здобувачами освіти на платформі ASPI.NET»

Студента 4 курсу, 6 групи,
спеціальності 121 «Інженерія
програмного забезпечення»
освітньої програми «Інженерія
програмного забезпечення» Дорощука
Ігоря Олеговича

_____
підпис студента

Науковий керівник
старший викладач кафедри
інженерії програмного
забезпечення та кібербезпеки Гнатченко Дмитро
Дмитрович

_____
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки Рзаєва Світлана
Леонідівна

_____
підпис гаранта

КИЇВ – 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Дорощуку Ігорю Олеговичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Веб-орієнтований додаток обліку відвідування занять здобувачами освіти на платформі ASPI.NET»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту розробка веб-орієнтованого додатку ведення обліку відвідуваності занять здобувачами освіти, для допомагати ведення обліку студентів, результати їх успішності/неуспішності.

Об'єкт дослідження робота університету.

Предмет дослідження процес проектування та розробки веб орієнтованого додатку з базою даних для ведення обліку відвідуваності занять здобувачами освіти.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКУ

1.1 Опис проблеми

1.2 Недоліки програм аналогів

1.3 Технічне завдання

1.4 Висновок до розділу 1

РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Предметна область

2.2 Use Case діаграма додатку

2.3 Концептуальна модель

2.4 Логічна модель

2.5 Фізична модель

2.6 Висновок до розділу 2

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКУ

3.1. Перелік програмних модулів

3.2 Система управління даними

3.3 Вибір мови прогрмування

3.4 Вибір фреймворку

3.5 Технологія для роботи з даними

3.6 Огляд додатка

3.7 Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Аналіз предметної області застосування веб-орієнтованого додатку*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Проектування програмного забезпечення*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Реалізація веб-орієнтованого додатку*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту

Гнатченко Д.Д.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Дорощук І.О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____ Дорощука І.О.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Рзаєва С.Л.

(прізвище, ініціали, підпис)

Завідувач кафедри _____ Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці та реалізації веб-орієнтованого додатку для обліку відвідуваності занять здобувачами освіти на платформі ASP.NET.

Шляхом порівняльного аналізу існуючих рішень в області обліку відвідуваності занять було визначено основні функціональні вимоги до розроблюваного додатку, також було визначено що більшість програмних продуктів платні, мають надлишковий функціонал та низьку швидкодію.

Розробка серверної частини була здійснена з використанням платформи ASP.NET, що надає потужні можливості для розробки веб-орієнтованих додатків. Клієнтська частина була реалізована у вигляді інтуїтивно зрозумілого веб-інтерфейсу.

Отриманий веб-орієнтований додаток представляє ефективне рішення для ведення обліку відвідуваності занять, що дозволяє зручно та швидко керувати даними про відвідуваність, викладачів, курси, студентів . Цей додаток можна використовувати для полегшення процесу обліку в навчальних закладах.

Ключові слова: веб-орієнтований додаток, архітектура, інтерфейс, надійність, швидкодія, облік відвідуваності, студенти, функціональні вимоги.

ABSTRACT

In accordance with the purpose of the research, the work is devoted to the development and implementation of a web-oriented application for recording the attendance of classes by students on the ASP.NET platform.

By means of a comparative analysis of existing solutions in the field of class attendance accounting, the main functional requirements for the developed application were determined, and it was also determined that most of the software products are paid, have redundant functionality and low speed.

The development of the server part was carried out using the ASP.NET platform, which provides powerful opportunities for the development of web-oriented applications. The client part was implemented in the form of an intuitive web interface.

The resulting web-oriented application represents an effective solution for keeping records of class attendance, which allows you to conveniently and quickly manage data on attendance, teachers, courses, students. This application can be used to facilitate the accounting process in educational institutions.

Keywords: web-oriented application, architecture, interface, reliability, speed, attendance accounting, students, functional requirements.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій у наш час призводить до того, що застарілі підходи до роботи викладачів в закладах вищої освіти стають менш ефективними. Через це більшість викладачів починають використовувати різні інструменти для обліку студентів, контролю їх успішності та організації цих даних на кафедрах.

Через це розробка веб-орієнтованого додатку ведення обліку відвідуваності занять необхідна та актуальна в теперішніх умовах.

Всі дані про студентів зберігаються в деканаті, і кожного разу, коли викладачі мають потребу у певній інформації про студента, вони змушені звертатися до працівників деканату, що викликає зайві витрати їхнього часу.

В архівах закладів вищої освіти зберігається значна кількість даних про студентів. Тому для кожної кафедри важливо мати систему обліку студентів, яка є засобом моніторингу. Правильно розроблена система обліку студентів допомагає викладачам економити час на пошуку необхідної інформації.

На даний час існують універсальні системи для ведення обліку відвідуваності, але більша частина з них або розташовуються на платних інтернет ресурсах, або включають в себе надлишковий функціонал, це спричиняє виникнення різного виду проблем з налаштуванням цих систем для вирішення задач, пов'язаних з обліком студентів конкретної кафедри.

Було вирішено створити систему ведення обліку відвідуваності занять здобувачами освіти для допомоги викладачам та грамототного використання їх часу на роботі.

Для методів та досліжень було обрано метод «Порівняння та аналізу», цей метод передбачає порівняння різних альтернативних рішень та аналіз їх

недоліків, вартості та технічних можливостей. Основна мета порівняння та аналізу полягає в тому, щоб визначити оптимальний варіант для розробки системи відвідуваності, який найкраще відповідає потребам і обмеженням.

Створюваний програмний продукт покликаний полегшити роботу викладачів, зменшити кількість можливих помилок в обліку. Програма зменшить час пошуку потрібних даних та дозволить вести обліку успішності.

База даних веб-додатку створюється для надійності збереження інформації, необхідної для правильного обліку інформації про студента, та для досягнення максимальної зручності та підвищення ефективності запису, що, в свою чергу, буде мати позитивний вплив на роботу викладача.

Актуальність створеного веб-орієнтованого додатку полягає в стрімкому розвитку інформаційних технологій. Це спричиняє збільшення попиту для електронного журналу для заміни звичайних записників

Метою створення програмного продукту є розробка веб-орієнтованого додатку ведення обліку відвідуваності занять здобувачами освіти, який буде допомагати викладачам вести облік студентів, результати їх успішності/неуспішності.

Завданням дослідження є створення зручного, інтуїтивно зрозумілого веб-орієнтованого додатку.

Об'єктом дослідження є робота університету.

Предметом дослідження є процес проектування та розробки веб орієнтованого додатку з базою даних для університету.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКУ

1.1. Опис проблеми

Останні роки все значимішою стає проблема оптимізації навчання, зростає потреба в керуванні навчальним процесом.

Звісно ж дані можна зберігати у вигляді списку в електронній таблиці або будь-якому текстовому редакторі, проте з часом, коли інформації стає достатньо багато, керувати таким списком стає достатньо проблематично. Виникає велика кількість різного роду помилок, невідповідностей та повторів, дедалі важче стає отримати чи переглянути необхідну інформацію. Для вирішення цієї проблеми використано базу даних, покликану вирішити усі вищезгадані проблеми. Для керування базою даних обрано програму Microsoft SQL Server.

Бази даних являють собою максимально зручний та надійний спосіб зберігання необхідної інформації та є досить потужним інструментом в роботі будь-якої організації.

Своєї популярності вони набули за декількох найбільш важливих особливостей, а саме: швидкість обробки величезних обсягів інформації, доступність інформації. Описані властивості баз даних дозволяють максимізувати продуктивність роботи будь-якого підприємства або певної його частини та мінімізувати кількість помилок та невідповідностей, які зазвичай виникають через людський фактор.

1.2. Недоліки програм аналогів

Нижче наведено опис трьох програм для ведення обліку відвідуваності занять та описано їх недоліки:

1) «Attendance Tracker»:
   - Відсутність інтеграції з іншими системами, такими як система керування навчальним процесом або система особових даних.
   - Обмежені можливості налаштування та адаптації програми під унікальні вимоги конкретного навчального закладу.
   - Обмежена підтримка користувачів та відсутність широкої спільноти для обміну досвідом та підтримки.
   - Недостатня стабільність програми, можливі збої або помилки.
2) «Attendance Manager»:
   - Складний інтерфейс, що може вимагати додаткового часу на оволодіння навичками використання програми.
   - Відсутність автоматизованого імпорту даних з інших джерел, що вимагає ручного введення інформації.
   - Відсутність інструментів зі збору додаткової статистики та аналітики відвідуваності.
   - Необхідність постійного з'єднання з Інтернетом для доступу до програми.
3) «Attendance Pro»:
   - Висока вартість ліцензій або підписок на використання програми.
   - Відсутність можливості інтеграції з іншими популярними системами, що використовуються в навчальних закладах.
   - Відсутність локалізації та підтримки для мов, які використовуються

1.3. Технічне завдання

1. Загальні відомості

1.1. Найменування системи

1.1.1. Повне найменування системи - Attendance Manager

1.1.2. Скорочене найменування системи - Attendance Manager

1.2. Планові терміни початку та закінчення робіт

Початок робіт - 01.01.2023

Закінчення робіт - 01.05.2023

1.3. Порядок оформлення і пред'явлення результатів робіт

Результати робіт будуть оформлені у вигляді програмного коду, документації з поясненням алгоритмів, принципів роботи. Результати будуть представлені замовнику в електронному вигляді, а також у паперовому вигляді за запитом замовника.

Побудова системи для ведення обліку відвідуваності занять ділиться на розробку самої бази даних, і також розробку інтерфейсу для користування та взаємодії зі створеною базою даних. Виходячи з цього було сформульовано наступні завдання:

— спроектувати базу даних для ведення обліку відвідуваності занять;
— розробити інтерфейс для адмінстратора системи, який дозволятиме відображати та редагувати всю інформацію, що міститься в базі даних студентів кафедри;
— розробити інтерфейс для викладача, який дозволятиме відображати та редагувати інформацію яка не стосується інших викладачів, в базі даних студентів кафедри;
— розробити інтерфейс студента для перегляду та пошуку необхідної інформації.

База даних студентів кафедри повинна встановлювати взаємозв'язок між студентами, курсами та їх навчальними планами і надавати можливість до виконання наступних функцій:

— додавання, редагування інформації про студентів, групи студентів;

— додавання, редагування інформації про індивідуальні навчальні плани студентів

— формування звітів з виконання кожним студентом навчального плану.

База даних повинна створюватись в програмі для SQL Server від Microsoft.

1. Створення інтерфейсу адміністрування системи

Інтерфейс для адміністратора системи має надавати можливість для редагування, видалення та редагування інформації з бази даних студентів кафедри, викладачів, та всю іншу інформацію. Адміністратор має абсолютний доступ до всієї інформації бази даних.

Інтерфейс викладача має зменшений функціонал. В нього є більшість дозволів адміністратора, окрім дій з викладачами.

Звичайний користувач має можливість лише для перегляду інформації з бази даних

Інтерфейс створюватиметься засобами середовища Microsoft Visual Studio за допомогою платформи ASP.NET Core.

2. Створення інтерфейсу користувача

У системі потрібно реалізувати перевірку користувача за допомогою автентифікації під час входу в систему.

Система потребує створення перевірки ролі користувача за допомогою аутентифікації.

Пошук в програмі повинен надавати змогу до швидкого пошуку необхідної інформації.

Інтерфейс повинен бути реалізований засобами програмного середовища Microsoft Visual Studio допомогою платформи ASP.NET Core.

Поставлено наступні завдання:

створити базу даних студентів кафедри, яка буде встановлювати взаємозв'язок між студентами кафедри і їх індивідуальними планами студентів;

— розробити інтерфейс для адміністрування системи, який надасть можливість для відображення та редагування всієї інформації, яка знаходиться у базі даних для ведення обліку відвідуваності занять та керування доступом користувачів до системи;
— розробити інтерфейс користувача системи для перегляду та пошуку необхідної інформації.
— розробити інтерфейс для викладача, який дозволятиме відображати та редагувати інформацію яка не стосується інших викладачів, в базі даних студентів кафедри;
— розробити інтерфейс студента для перегляду та пошуку необхідної інформації.

3. Потенційні користувачі системи

Система переважно розрахована для викладачів, які хочуть вести облік відвідуваності занять студентами в електронному форматі. Також для студентів надана можливість перегляду інфомації про викладачів і їх дані для зв’язку. Інтерфейс веб-додатку виконаний максимально зручним та інтуїтивно-зрозумілим для забезпечення максимальної продуктивності роботи клієнта.

4. Практична значимість

Дипломна робота надає можливість замінити ручне ведення обліку на автоматизовану систему, що значно спрощує процес та зменшує можливість допустити похибку. Використання веб-додатка дозволяє зберігати дані в

електронному вигляді, забезпечуючи легкий доступ до них і зручний пошук і аналіз.

Зручний доступ до інформації: Веб-додаток дозволяє викладачам, адміністраторам і студентам легко отримувати доступ до інформації про відвідування занять. Це може включати записи про присутність, відсутність, час відвідування, оцінки тощо. Вся ця інформація може бути доступна в режимі реального часу і зручно організована для користувачів.

1.4. Висновок до розділу 1

На основі проведених досліджень було сформовано детальне технічне завдання, яке допоможе розробити якісний продукт – веб-орієнтований додаток для ведення обліку відвідуваності занять. В даному завданні враховано всі основні вимоги і потреби закладу.

Для початку, були визначені функціональні вимоги до системи. Серед них належить можливість введення даних про учнів, викладачів , їх класи та інші необхідні дані. Крім того, система повинна включати можливість позначення відвідування, виставлення балів для студента.

Для забезпечення безпеки та обмеження доступу до даних, технічне завдання передбачає реалізацію системи аутентифікації та авторизації, що дозволить визначити рівні доступу до інформації для різних користувачів. Наприклад, адміністратор має повний доступ до всіх функцій системи, викладачі можуть мають функціонал адмінів, окрім можливості створення та видалення викладачів, а студенти можуть лише переглядати інформацію.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Предметна область

Було проведено огляд предметної області системи обліку відвідуваності занять. Виконано аналіз існуючих засобів та систем аналогів. Проаналізувавши програми для виконання поставлених задач, виявлено, що існуючі системи обліку мають надлишковий функціонал який зменшує швидкодію, та доступ лише з платною підпискою.

Проектування моделі бази даних для ведення обліку відвідуваності занять, процес створення бази даних спираючись на створені моделі та розбка веб-орієнтованого додатку для керування базою даних надасть змогу більш ефективно вирішувати завдання що зв'язані з обліком студентів кафедри.

Основним завданням даної роботи є розробка правильної моделі, а також та створення інтерфейсу користувача для зручного перегляду та маніпуляцій з даними.

Серед усіх можливих моделей даних найпоширенішою є реляційна, саме вона використовується при розробці бази даних відділу кадрів. Нижче розглянемо переваги цієї моделі.

Реляційна модель даних характеризується простотою структури даних та володіє достатньо зручним для користувача поданням даних – табличним. Тобто уся необхідна інформація зберігається у вигляді таблиць (відношень). Кожна таблиця складається з рядків (кортежів) та стовпчиків (атрибутів).

Водночас з простотою структури, реляційна модель даних володіє досить високим рівнем захищеності від некоректних змін, тобто включає в себе властивість цілісності даних.

Однією з основоположних ідей реляційної моделі даних у тому, що зв’язок між даними встановлюється у відповідності з внутрішньою структурою логічних взаємовідносин. Тобто, між таблицями бази даних існують відповідні зв’язки, які допомагають підтримувати цілісність даних. Саме тому визначення та встановлення правильних зв’язків є надзвичайно важливим етапом розробки бази даних.

Кожна таблиця реляційної моделі даних повинна задовольняти мінімальний перелік вимог:

- значення у таблиці повинні бути одиночними, тобто кожен клітинка таблиці є одним елементом даних;
- взаємозв’язки повинні подаватися у вигляді об’єктів;
- кожна таблиця відображає лише один об’єкт і включає в себе визначений перелік атрибутів цього об’єкту;
- кожен стовпчик таблиці має унікальне ім’я;
- однотипність записів одного стовпця;
- кожна таблиця містить первинний ключ (простий – з одного поля, складений – з двох або більше полів), що однозначно ідентифікує кожен рядок таблиці;
- таблиця не може містити двох однакових кортежів;

Основною перевагою реляційних баз даних є можливість створювати значущу інформацію шляхом об’єднання таблиць. Об'єднання таблиць дозволяє зрозуміти взаємозв'язок між даними або спосіб їх об’єднання. SQL включає можливість підрахунку, додавання, групування, а також об'єднання запитів. SQL може виконувати основні математичні та проміжні функції та логічні перетворення.

2.2. Use case діаграма додатку

Use Case діаграма - це вид діаграми в Unified Modeling Language, яка використовується для моделювання функціональної поведінки системи. Вона описує взаємодію системи з користувачами або іншими системами, що вона повинна робити для задоволення їх потреб.

Ця діаграма слугує ключовим інструментом для проведення аналізу вимог до системи. Вона дозволяє визначити, як система буде себе вести з іншими системами і користувачами, які процеси має виконувати та які функціональність має бути доступна для користувачів (рис.2.1.).

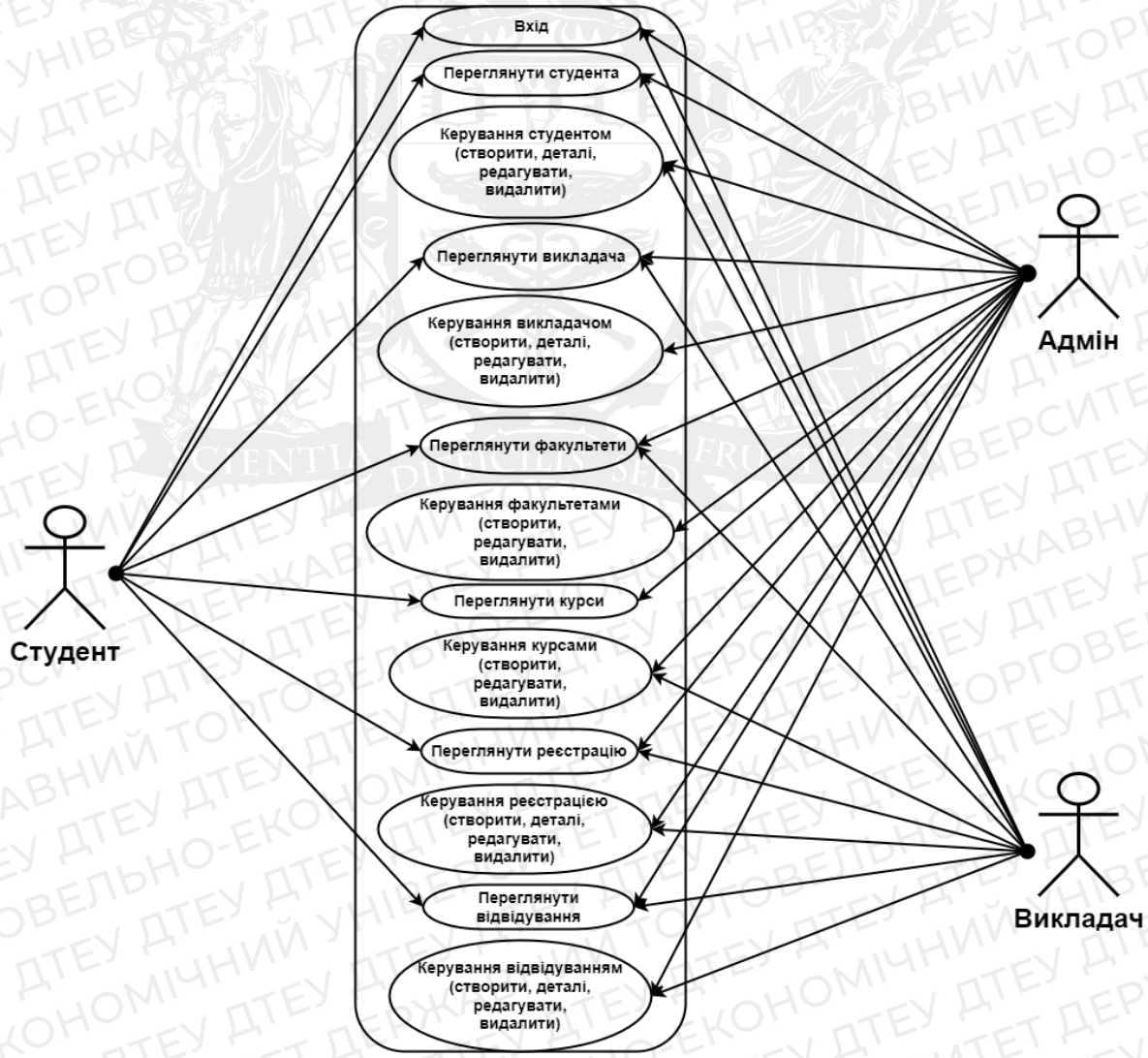

Рис. 2.1. Use Case діаграма

Джерело: побудовано автором

У ній актори зображаються як люди, інші системи або компоненти системи. Актори можуть тобто викликати функціональність системи. Use Case описує функціональну поведінку системи, яка відповідає на запит актора. Кожен Use Case може мати свій унікальний ідентифікатор та опис. Кожен Use Case може мати свої власні параметри та результати, які повертає система. Також в діаграмі можуть бути зображені залежності між Use Case, коли один Use Case викликає інший.

Ця діаграма може використовуватись для більш глибокого розуміння системи та її функціональності. Також її використовують для виявлення недоліків в системі та уточнити вимоги до неї. Вона може бути використовуватись для визначення тестових сценаріїв та для валідації системи на етапі розробки.

Загалом, Use Case діаграма є важливим інструментом потрібним щоб аналізувати вимоги до системи яку потрібно проектувати. Вона дозволяє краще розуміти функціональну поведінку системи та визначити, як вона повинна взаємодіяти з користувачами та іншими системами.

2.3. Концептуальна модель бази даних

Найпершою створюваною моделлю є концептуальна, яка являє собою модель предметної області, що складається з переліку взаємопов'язаних понять, які беруть участь для опису цієї області. Тобто це інформаційна модель найбільш високого рівня абстракції.

Концептуальна модель використовується на етапі аналізу вимог та проектування системи. Вона дозволяє детально вивчити предметну область, яку моделює система, та визначити її ключові поняття, відносини та атрибути. Ця модель допомагає уточнити вимоги до системи та визначити, які функціональність та процеси повинні бути реалізовані.

Ще один метод використана, це створення документації проекту, яка буде корисною для розробників, тестувальників та інших зацікавлених сторін. Вона допомагає уточнити вимоги та забезпечити зрозумілість між різними учасниками проекту, допомагає уточнити ключові аспекти предметної області та сприяє розумінню взаємодії між різними складовими системи. Концептуальна модель може допомогти зменшити складність проекту, дозволяючи зосередитися на головних функціях та залежностях в системі.

Крім того, концептуальна використовується як основа для розробки фізичної моделі - реалізації системи в обраному середовищі програмування. При цьому, концептуальна модель може слугувати основою для розробки бази даних. Важливо забезпечити належну документацію концептуальної моделі, що дозволить уникнути непорозумінь та сприятиме розумінню всіма учасниками проекту.

У загальному розумінні, концептуальна модель є ключовим етапом проектування системи та важливим інструментом для забезпечення розуміння вимог до системи, визначення функціональності та процесів та забезпечення співпраці між різними учасниками проекту.

Така модель створюється без орієнтації на якусь конкретну СУБД та незалежно від сприйняття її окремими користувачами та способів опису в комп’ютерній системі.

Концептуальна модель представлена на схемі нижче (рис.2.2.).

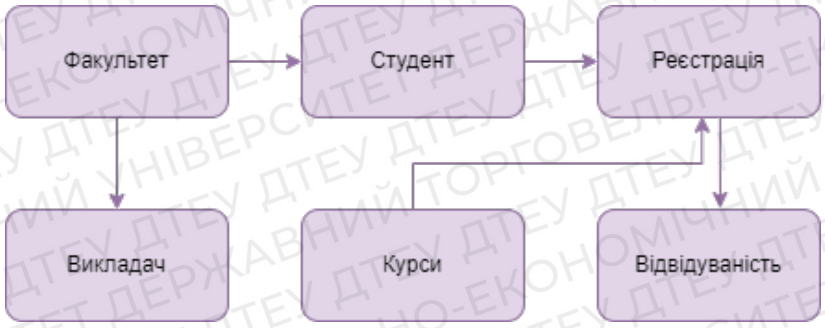

Рис. 2.2. Концептуальна модель

Джерело: побудовано автором

Після створення концептуальної моделі наступним кроком є розробка логічної моделі, яка також відображає предметну область, яка показується незважаючи на певну програму чи засіб чи технології зберігання, але вона містить структуру даних, такі як: реляційні таблиці , артифакти, відношення. Побудова логічної моделі допомагає уникнути надмірності даних та сприяє розумінню елементів даних та вимог.

2.4. Логічна модель бази даних

Логічна модель - це візуальне зображення структури та взаємозв'язків між даними у системі. Вона використовується для розуміння взаємодії між різними елементами даних та для визначення, які дані повинні бути збережені в системі. Ця модель є важливим інструментом для проектування баз даних та інших інформаційних систем а також для зображення структур даних на вищому рівні абстракції, ніж фізична модель. Вона описує, які дані повинні бути збережені та як вони пов'язані між собою, але не визначає, як саме ці дані зберігаються на диску або яким чином вони обробляються програмним забезпеченням.

Тут використовуються сутності, атрибути та відносини між сутностями. Сутність - це об'єкт або поняття, яке можна описати даними. Атрибут - це характеристика сутності, яка описує її властивості. Відносина - це зв'язок між двома або більше сутностями, який описує взаємозв'язок між ними.

Ця модель бере участь у проектування реляційних баз даних, які є найпоширенішою формою збереження даних у сучасних інформаційних системах. Вона може допомогти зрозуміти, які таблиці повинні бути створені, які поля повинні бути у кожній таблиці та як вони повинні бути пов'язані між собою.

У загальному розумінні, логічна модель є важливим інструментом для розуміння та проектування структури даних у системах та забезпечення їх ефективного та правильного збереження та обробки. Вона може бути використана як для проектування баз даних, і також для різних інформаційних систем, наприклад, систем керування вмістом, систем керування проектами тощо(рис.2.3.).

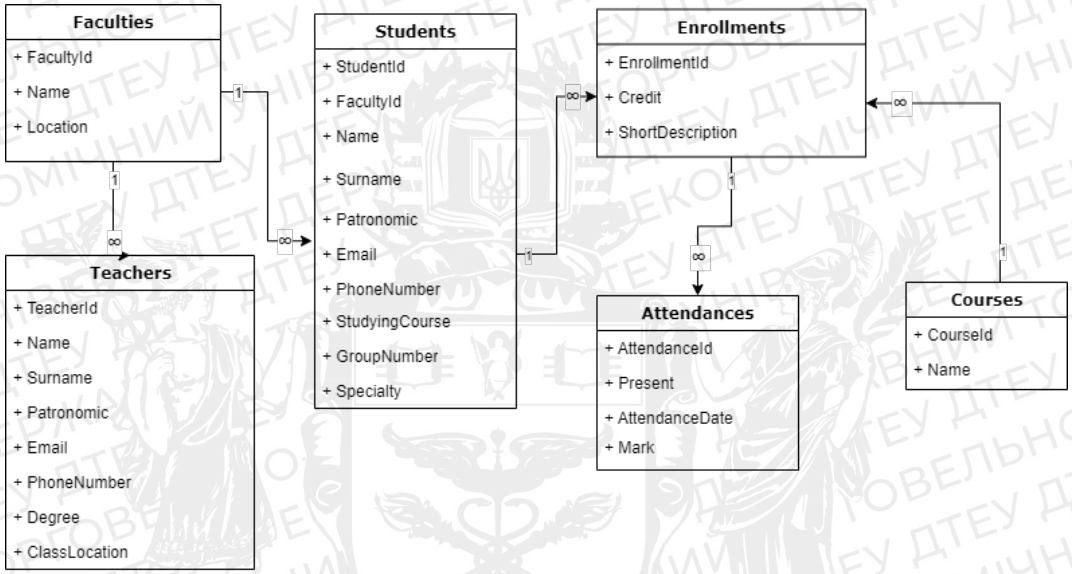

Рис. 2.3. Логічна модель

Джерело: побудовано автором

На основі логічної моделі створюємо фізичну модель даних. Дана модель найконкретніше та найповніше містить опис уявлення та зберігання інформації, вона містить в собі усі елементи бази даних, як потрібні щоб створити зв’язки поміж таблицями. Етап створення фізичної моделі включає в себе забезпечення цілісності бази даних, що досягається за допомогою правил допустимості даних.

2.5. Фізична модель бази даних

Фізична модель - це опис того, які саме дані будуть зберігатися на комп'ютері та як програмне забезпечення буде їх обробляти. Ця модель

показує нам кількість таблиць, колонок та ключів, а також типів даних та обмежень.

Ця модель є важливою складовою в розробці баз даних та інших інформаційних систем. Вона надає змогу розробникам здійснювати опис яким чином дані будуть зберігатися та оброблятися в програмі, це нає змогу для більш ефективного і коректного збереження і обробки даних.

Фізична модель включає в себе визначення структури бази даних, включаючи таблиці, колонки та ключі, типи даних та обмеження на дані. Вона також визначає, як дані будуть зберігатися на диску, включаючи розмір блоків даних, розташування файлів та інші деталі. Фізична модель також включає опис індексів, які використовуються для швидкого доступу до даних.

Фізична модель зазвичай створюється на основі логічної моделі, але має більш детальний рівень абстракції та визначає конкретність та розташування даних. Це означає, що фізична модель визначає, яким чином на диску будть зберігатись та як будуть оброблятися дані за допомогою програмного забезпеченням.

У підсумку, фізична модель є важливою складовою розробки програмного забезпечення, що дозволяє розробникам програмного забезпечення описати, як саме дані будуть зберігатися та оброблятися в програмі. Вона описує детальні характеристики бази даних та її взаємодію з програмним забезпеченням, забезпечує безпеку даних та оптимізацію запитів до бази даних. Фізична модель також допомагає забезпечити масштабованість та ефективність роботи системи.

Таблиця «Teacher» містить в собі іформацію про викладача та включає в себе 9 полів. Первинним ключем є поле TeacherId – ідентифікатор викладача. В полях Name, Surname, Patronomic, Email, PhoneNumber, Degree, ClassLocation відповідно у нас задаються ім'я , прізвище, по-батькові, емайл, номер телефону, освіта, розташування

кабінету викладача. Поле Facultyd представляє зовнішній ключ для зв'язку з таблицею Faculty.

Структура таблиці «Teacher» зображена на рисунку нижче (рис.2.4.).

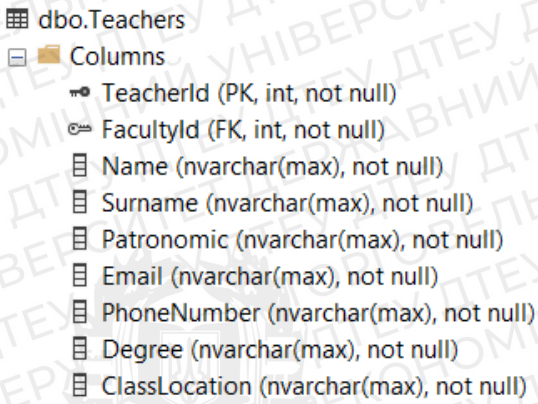

Рис. 2.4. Таблиця Teachers

Джерело: побудовано автором

Таблиця «Attendances» містить інформацію про відвідуваність та включає в себе 6 полів. Структура таблиці «Attendances» зображена на рисунку нижче (рис.2.5.).

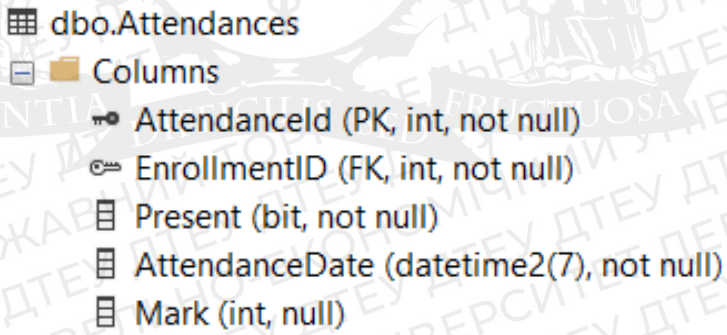

Рис. 2.5. Таблиця Attendance

Джерело: побудовано автором

Первинним ключем є поле AttendanceId – ідентифікатор відвідуваності. Такі містить такі поля як Present, AttendanceDate, Mark відповідно означають чи присутній учень, дата, оцінка. Поле EnrollmentId представляє зовнішній ключ для зв'язку з таблицею Enrollment.

Таблиця «Courses» містить інформацію про курси(дисципліни) доступні студентам кафедри. Структура таблиці зображена на нижче. (рис.2.6.).

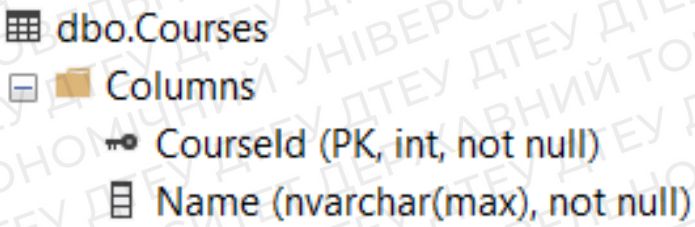

Рис. 2.6. Таблиця Courses

Джерело: побудовано автором

Первинним ключем таблиці є поле CourseId – ідентифікатор контактних даних. Поле Name містить інформацію про назву дисципліни.

Таблиця «Enrollments» містить інформацію реєстрацію студентів. Структура таблиці зображена на нижче (рис.2.7.).

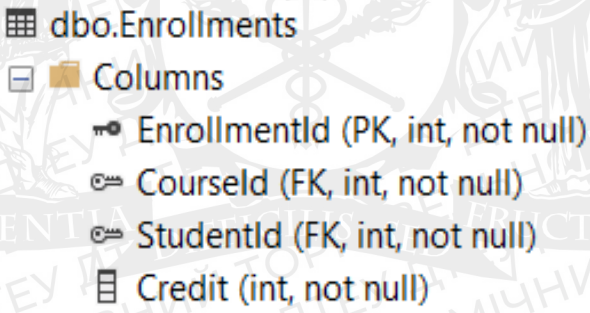

Рис. 2.7. Таблиця Enrollment

Джерело: побудовано автором

Первинним ключем таблиці є поле EnrollmentId. Таблиця містить інформацію про кількість занять за семестер. Поле CourseId представляє зовнішній ключ для зв'язку з таблицею Course. А поле StudentId в свою чергу виступає зовнішнім ключем для таблиці Student.

Таблиця «Faculties» містить інформацію факультети університету. Структура таблиці зображена на на рисунку нижче (рис.2.8.).

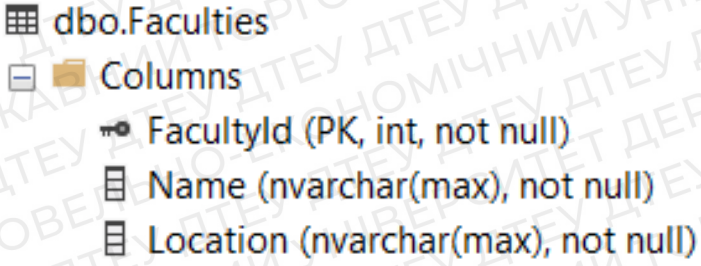

Рисунок 2.8. Таблиця Faculty

Джерело: побудовано автором

Первинним ключем таблиці є поле FacultyId. Таблиця містить інформацію про факультети навчального закладу. Поле Name означає назву факультету, а поле Location показує розташування головного кабінету.

Таблиця «Students» містить інформацію про студентів університету. Структура таблиці зображена на рисунку нижче (рис.2.9.).

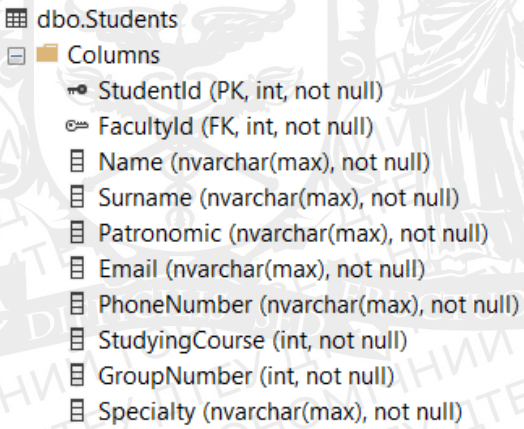

Рисунок 2.9. Таблиця Students

Джерело: побудовано автором

Таблиця «Students» включає в себе 10 полів. Первинним ключем є поле StudentId – ідентифікатор викладача. В полях Name, Surname, Patronomic, Email, PhoneNumber, StudyingCourse, GroupNumber, Specialty відповідно у нас задаються ім'я , прізвище, по-батькові, емайл, номер телефону, номер курсу, спеціальність. Поле Facultyd представляє зовнішній ключ для зв'язку з таблицею Faculty.

Фізична модель представлена сукупністю описуваних об’єктів, кожен з яких відповідає окремій таблиці майбутньої бази даних. Кожен об’єкт містить назву таблиці, перелік полів, а також ключ, індексоване поле, тип даних та null-значення кожного атрибуту.

2.6. Висновок до розділу 2

Отже, підсумовуючи вище викладене, можно дійти до висновку, що даний веб-орієнтований додаток дозволяє спростити процес ведення обліку відвідуваності занять. В даному розділі було проведено огляд предметної області системи обліку відвідуваності занять. Проаналізувавши програми для виконання поставлених задач, виявлено, що існуючі системи обліку мають надлишковий функціонал який зменшує швидкодію, та доступ лише з платною підпискою. Також у розділі було виконано проектування додатку, створено концептуальну, логічну та фізичну модель, Use case діаграму.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОГО ДОДАТКУ

3.1. Перелік програмних модулів

При розробці будь-якої системи важливо правильно вибрати мову програмування, середовище розробки та базу даних, які найкраще відповідатимуть потребам проекту. У даному випадку, була обрана мова програмування C# - це мова програмування, яка є частиною технології .NET, розробленої компанією Microsoft. C# є популярною мовою програмування, яка забезпечує високу продуктивність, безпеку та простоту розробки.

Для розробки веб-додатку, використовувалося середовище розробки Microsoft Visual Studio 2022. Це потужне інтегроване середовище розробки (IDE), яке підтримує різні мови програмування та має різноманітні інструменти для розробки веб-додатків.

Для керування базою даних було використано Microsoft SQL Server. Це система керування базами даних, розроблена компанією Microsoft, яка забезпечує високу продуктивність, масштабованість та безпеку. SQL Server має широкий функціонал, включаючи можливість створення процедур та функцій для керування даними, які можуть бути використані в коді програми.

3.2. Система управління даними

Microsoft SQL Server - це програмне забезпечення для управління базами даних, яке дозволяє зберігати, обробляти і аналізувати великий обсяг інформації в бізнес-середовищах. Воно надає різноманітні функції для

здійснення операцій з транзакціями, аналізу даних і бізнес-аналітики в корпоративних системах IT.

Одна з провідних систем на ринку технологій баз даних, що працюють разом із Oracle Database і DB2 IBM, - це Microsoft SQL Server. Вона базується на мові програмування SQL, яку використовують адміністратори баз даних та IT-фахівці для управління базами даних і виконання запитів до даних. SQL Server використовує реалізацію SQL від Microsoft, відому як Transact-SQL, яка має додаткові можливості програмування порівняно зі стандартною мовою.

Ця програма побудована на основі реляційної моделі СУБД, де дані зберігаються в таблицях із пов'язаними рядками. Це дозволяє уникнути дублювання даних і забезпечує цілісність даних та інші обмеження для забезпечення точності та надійності. Вона має компонент, відомий як SQL Server Database Engine, який відповідає за зберігання, обробку та безпеку даних. Він включає реляційний механізм для обробки команд і запитів, а також механізм зберігання, який керує файлами бази даних і об'єктами, такими як таблиці, індекси та транзакції, а також використовує операційну систему SQL Server або SQLOS для керування функціями нижчого рівня, такими як керування пам'яттю, планування завдань та блокування даних. На рівні мережевого інтерфейсу використовується протокол потоку табличних даних Microsoft для взаємодії з серверами баз даних.

Адміністратори баз даних та розробники SQL Server використовують мову T-SQL для створення та зміни структур бази даних, маніпулювання даними та виконання інших завдань, таких як забезпечення безпеки та резервного копіювання.

SQL Server також надає різноманітні служби та інструменти для керування даними, включаючи служби інтеграції SQL Server, служби якості даних SQL Server і служби основних даних SQL Server. Для адміністрування баз даних і розробки існують SQL Server Data Tools і SQL Server Management

Studio, які допомагають у розробці, розгортанні та керуванні базами даних (рис.3.1.).

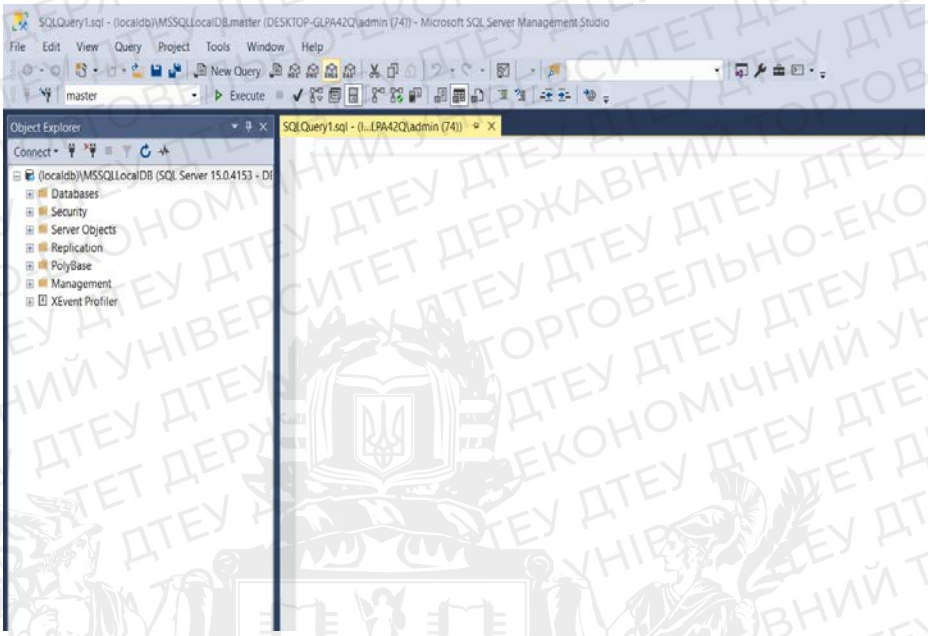

Рис. 3.1. Microsoft SQL Server.

Джерело: побудовано автором

3.3. Вибір мови програмування.

C# - це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft для платформи .NET. Ця мова програмування з'явилася в 2000 році і швидко стала дуже популярною в середовищі розробки програмного забезпечення для Windows.

Мова має багатий синтаксис, який дозволяє розробникам ефективно створювати складні програми з великою кількістю функцій. Особливістю мови є те, що вона підтримує багато концепцій об'єктно-орієнтованого програмування, таких як наслідування, інкапсуляція та поліморфізм.

Однією з переваг C# є його широке застосування для розробки різноманітних додатків, від простих програм до складних веб-додатків, сервісів та ігор. Мова C# також має потужну базу класів .NET, що дозволяє розробникам швидко і ефективно створювати програми.

Для дипломної роботи була обрана мова програмування C# дипломної роботи, оскільки ця мова програмування має потужний функціонал, що дозволить мені ефективно реалізувати поставлені завдання. Також, середовище розробки Visual Studio, яке підтримує C#, має багатий інструментарій для розробки додатків, що значно полегшує процес програмування.

C# є однією з кращих мов програмування для розробки веб-додатків з кількох причин:

Інтеграція з .NET Framework та .NET Core: C# підтримується .NET Framework та .NET Core, що дозволяє розробникам використовувати багато бібліотек та інструментів, що допомагають зробити розробку швидкою та ефективною.

Безпека та надійність: C# підтримує безпеку типів, що допомагає уникнути помилок та зменшити ризики злому безпеки. Крім того, C# підтримує обробку винятків, що дозволяє обробляти помилки та забезпечувати надійність додатків.

Продуктивність: C# є компільованою мовою програмування, що дозволяє зменшити час виконання програм та збільшити продуктивність додатків. Крім того, C# підтримує багатопоточність, що дозволяє використовувати потужність багатоядерних процесорів та підвищувати продуктивність додатків.

Масштабованість: C# дозволяє розробляти додатки будь-якої складності та масштабувати їх залежно від потреб користувачів та бізнес-вимог.

Підтримка багатьох платформ: C# є кросплатформеною мовою програмування, що дозволяє розробляти додатки для різних платформ, включаючи Windows, Linux та MacOS.

У загальному, C# має багато переваг для розробки веб-додатків, що робить його популярним вибором серед розробників.

3.4. Вибір фреймворку

ASP.NET Core — це кросплатформний фреймворк з відкритим вихідним кодом для створення сучасних додатків, підключених до Інтернету, таких як веб-програми, мобільні серверні програми. Програми ASP.NET Core можуть працювати на .NET Core або на повній версії .NET Framework. Він був розроблений, щоб забезпечити оптимізовану розробку програм, які розгортаються в хмарі або виконуються локально. Він складається з модульних компонентів які забезпечують мінімальні навантаження на систему та забезпечують швидкодію, тож ви зберігаєте гнучкість під час створення своїх рішень.

Перша версія ASP.NET вийшла майже 15 років тому як частина .NET Framework. Відтоді мільйони розробників використовували його для створення та запуску веб-програм, і протягом багатьох років було додано багато функцій та нових можливостей.

Величезний плюс ASP.NET Core – кросплатформеність. Це дозволяє розробляти та запускати свої додатки ASP.NET Core між платформами на Windows, Mac і Linux.

ASP.NET Core має низку архітектурних змін, які призводять до набагато компактнішоїструктури. ASP.NET Core більше не базується на System.Web.dll. Він заснований на наборі детальних і добре розбитих пакетів NuGet. Це дозволяє оптимізувати вашу програму, щоб включити лише ті пакети NuGet, які вам потрібні. Переваги меншої площі програми включають більш сувору безпеку, скорочення обслуговування, покращену продуктивність і зниження витрат у моделі «оплата за те, що ви використовуєте».

ASP.NET Core був випущений з численними покращеннями в порівнянні з попередніми версіями ASP.NET. Ось 15 основних покращень, які отримав ASP.NET Core:

- Підтримка багатьох платформ - ASP.NET Core працює на Windows, Linux та Mac.
- Оптимізація продуктивності - ASP.NET Core має вбудовану підтримку асинхронних запитів, що знижує час відповіді сервера та збільшує продуктивність додатку.
- Вбудована підтримка Docker - можна легко запустити та розгорнути додаток ASP.NET Core у контейнері Docker.
- Вбудована підтримка SignalR - ASP.NET Core має вбудовану підтримку SignalR для створення реального часу додатків.
- Незалежність від сервера - можна запустити додаток ASP.NET Core на будь-якому сервері, включаючи IIS, Kestrel та Apache.
- Інтеграція з Angular, React та іншими JavaScript-бібліотеками - ASP.NET Core має вбудовану підтримку для інтеграції з JavaScript-бібліотеками, що дозволяє створювати багатофункціональні додатки.
- Менше коду - ASP.NET Core має менше коду порівняно з попередніми версіями, що полегшує розробку та підтримку додатків.
- Розширена підтримка API - ASP.NET Core має вбудовану підтримку RESTful API, що дозволяє легко створювати API.
- Нова система конфігурації - ASP.NET Core має нову систему конфігурації, що дозволяє легко настроювати додаток на різних етапах розробки.
- Нова система Middleware - ASP.NET Core має нову систему Middleware, що дозволяє додавати та налаштовувати Middleware у ланцюжку обробки запитів, що дозволяє легко додавати нову функціональність до додатку.

Нижче наведено порівняння швидкодії .Net з іншими популярними фреймворками (рис.3.2.).

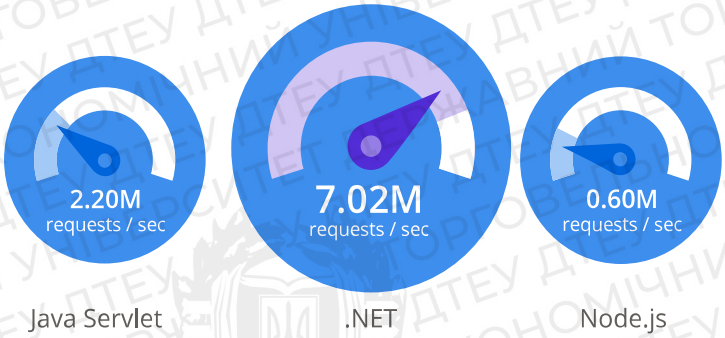

Рис. 3.2. Порівняння швидкодії

Джерело: побудовано автором

3.5. Технологія для роботи з даними

Для роботи з даними обрано спеціальну об'єктно-орієнтовану технологію Entity Framework на базі фреймворку створеного Microsoft. Так як звичайні інструменти .NET надають змогу виконувати з'єднання, команди та інші дії що надють змогу взаємодіюти з базами даних, тобто це є найвищим рівнем абстракції, який дозволяє абстрагуватися від самої бази даних та взаємодіяти із інформацією незважаюти на тип сховища. Так як фізично розробник управляє таблицями, взаємодіє із їх ключами , проте Entity Framework нам пропонує взаємодію на концептуальному рівні.

Основною концепцією Entity Framework є поняття сутності. Сутність це набір даних, асоційованих з певним об'єктом. Через це ця технологія взаємодіє з об'єктами та їх наборами.

Усі сутності так як і будь-який об'єкт з нашого світу, має ряд властивостей. Для прикладу, якщо сутність описує людину, ми можемо виділити основні властивості, як ім'я, дату народження, зріст, стать, вагу. Ці

властивості можуть бути представляють як простими типами даних таких як integer, також вони можуть бути представлені у вигляді різної збірної інформації. Кожна таблиця може містити в собі одна або декілька властивостей, котрі визначають її унікальність та показують її відмінність.

Сутності можуть бути пов'язані асоціативними зв'язками один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як це відбувається зі зв'язками через зовнішні ключі в реальних базах даних.

Особливістю Entity Framework є використання запитів LINQ для вибірки даних з бази даних. За допомогою LINQ ми можемо не лише отримувати певні рядки, які зберігають об'єкти, з бази даних, але й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Ще одним ключовим поняттям є Entity Data Model. Ця модель встановлює відповідність між класами сутностей та реальними таблицями в базі даних. Entity Data Model складається з трьох рівнів: концептуального рівня, рівня сховища та рівня зіставлення (мапінгу).На концептуальному рівні відбувається визначення класів сутностей, які використовуються в додатку.

Рівень сховища визначає таблиці, стовпці, відносини між таблицями та типи даних, з якими пов'язана використовувана база даних. Рівень зіставлення (мапінгу) діє як посередник між попередніми двома рівнями, визначаючи відповідності між властивостями класу сутності та стовпцями таблиць. Отже, через класи, визначені в додатку, ми можемо взаємодіяти з таблицями бази даних.

Entity Framework надає три можливі способи взаємодії з базою даних:

Database first: Entity Framework створює набір класів, що відображають модель конкретної бази даних.

Model first: спочатку розробник створює модель бази даних, на основі якої Entity Framework створює реальну базу даних на сервері.

Code first: розробник створює класи моделі даних, які будуть зберігатися в базі даних, а потім Entity Framework, використовуючи цю модель, генерує базу даних та її таблиці.

У нашому додатку буде використовуватись найбільш популярний підхід Code first.

3.6. Опис веб-орієнтованого додатку

У веб-орієнтованому додатку можна виділити головне вікно, форму для авторизації, інтерфейс для адміністратора системи, інтерфейс для викладача, інтерфейс для студента.

Після запуску програми відкриється вікно зображене на рисунку нижче. Дане вікно є головним, з якого починається вхід користувача в систему (рис.3.3.).

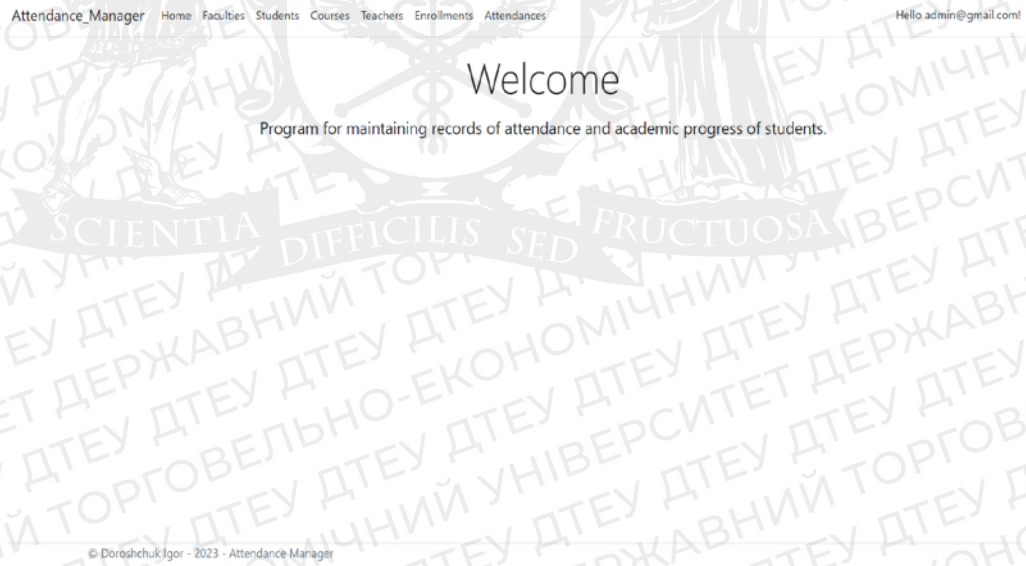

Рис. 3.3. Головне вікно

Джерело: побудовано автором

Форма авторизації зображена на рисунку нижче . Вона містить запит на введення логіну і паролю. Введені значення логіну і паролю

відправляються в базу даних студентів кафедри для пошуку користувача бази даних (рис.3.4.).

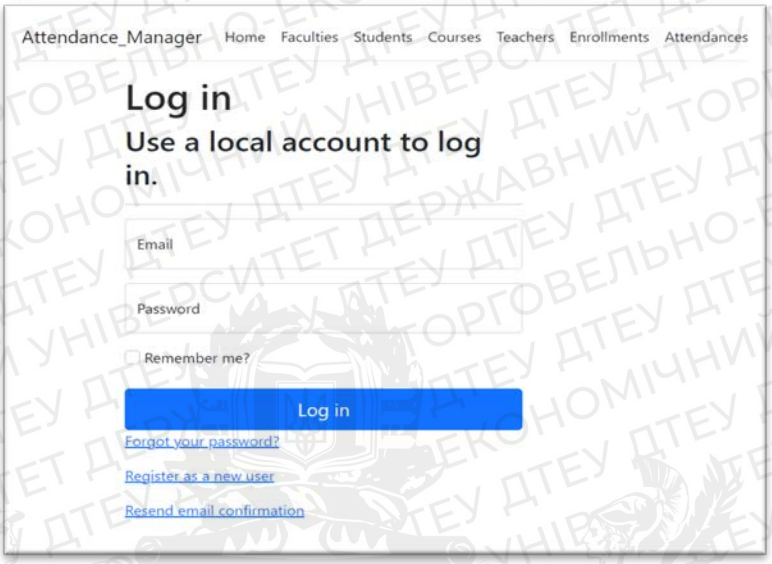

Рис. 3.4. Форма авторизації

Джерело: побудовано автором

Після введення неправильного логіну або паролю користувач отримає попередження (рис.3.5.).

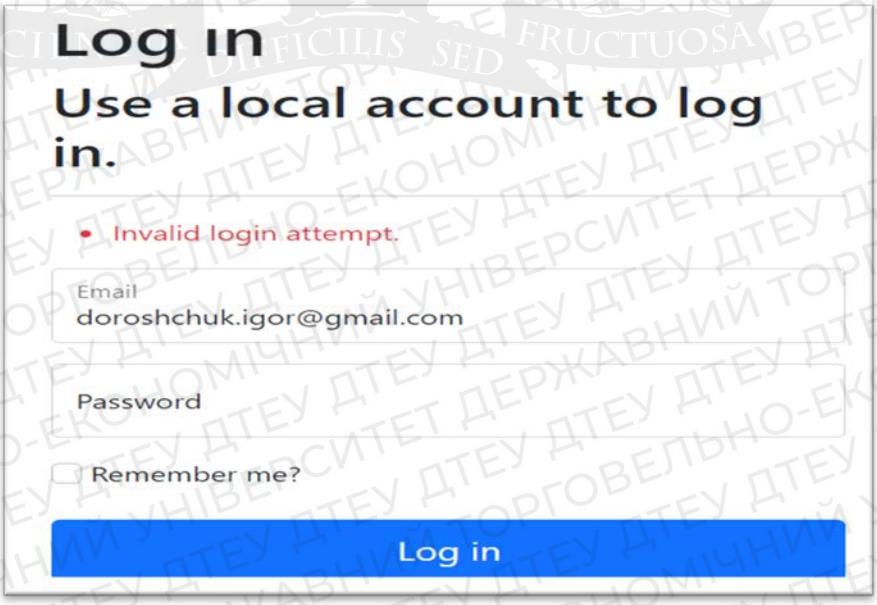

Рис. 3.5. Невірні дані

Джерело: побудовано автором

У футері програми знаходиться сім ссилок (рис.3.6).

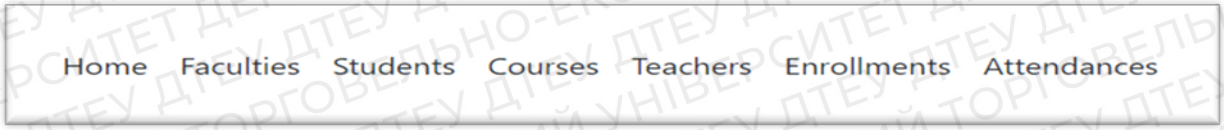

Рисунок 3.6. Футер

Джерело: побудовано автором

При переході на Факультети користувач бачe вікно нижче. (рис.3.7.).

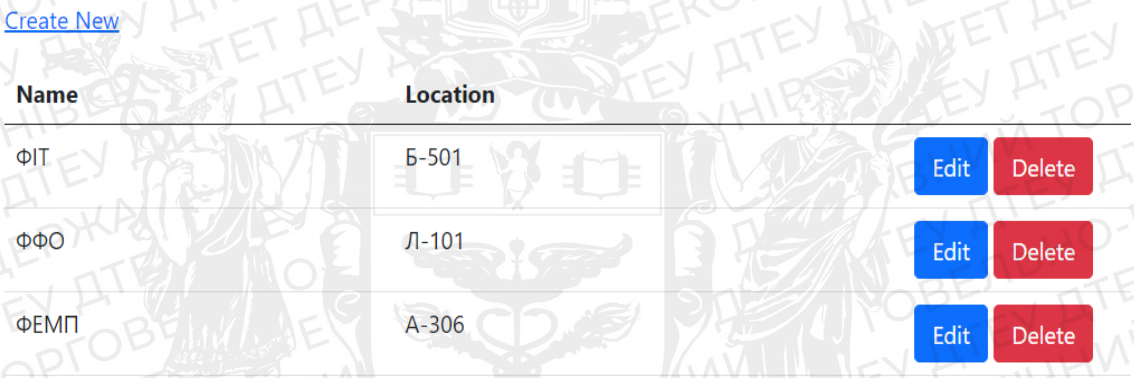

Рис. 3.7. Факультети

Джерело: побудовано автором

В даному вікні можна додавати нові факультети, редагувати вже створені а також видаляти їх.

При переході на Студентів користувач бачe вікно зображене на рисунку нижче (рис.3.8.).

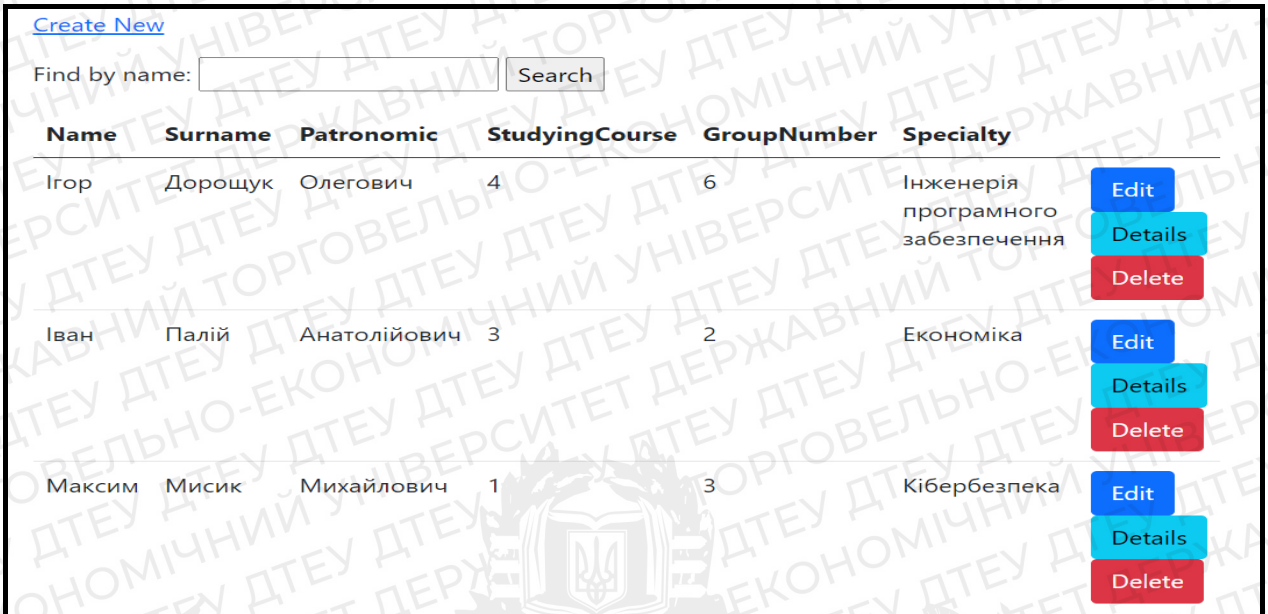

Рис. 3.8. Студенти

Джерело: побудовано автором

При переході на дисципліни користувач бачe вікно зображене на рисунку нижче. (рис.3.9.).

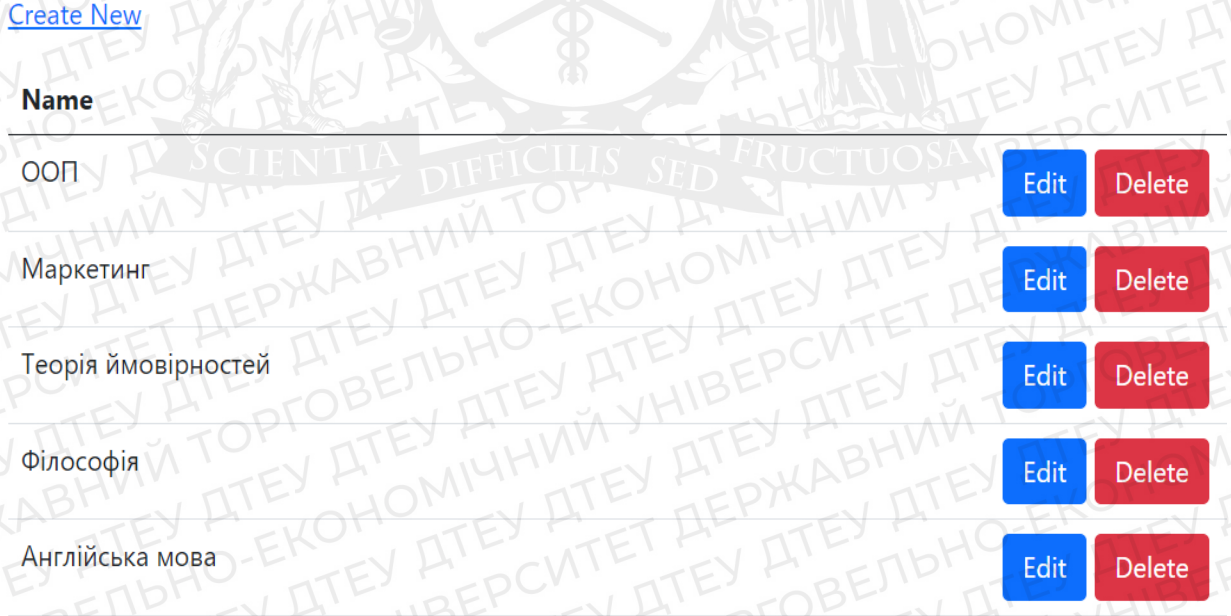

Рис. 3.9. Список дисциплін

Джерело: побудовано автором

При переході на викладачів користувач бачe вікно зображене нижче. (рис.3.10.).

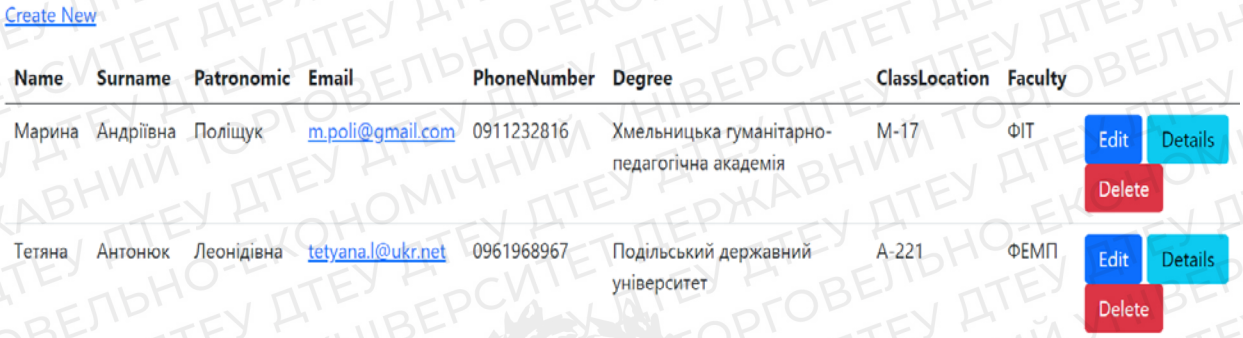

Рис. 3.10. Викладачі

Джерело: побудовано автором

При переході на список дисциплін користувач бачe вікно зображене на рисунку нижче (рис.3.11.).

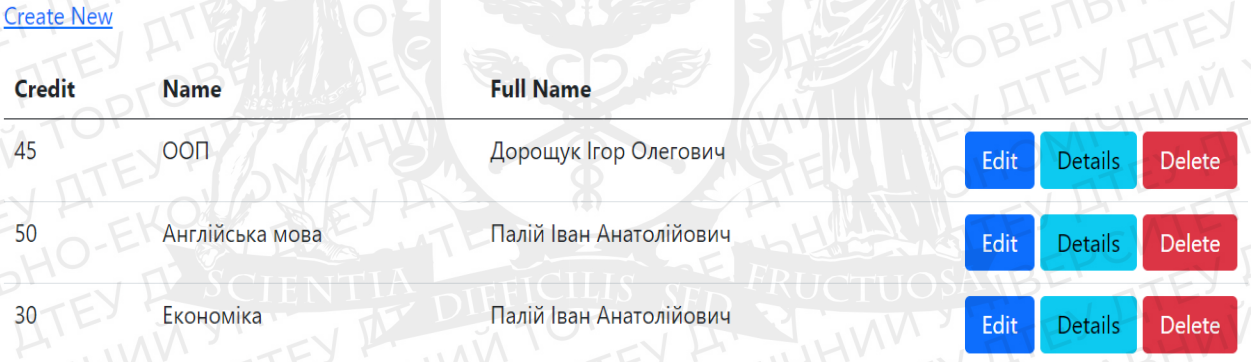

Рис. 3.11. Список дисциплін

Джерело: побудовано автором

При переході на список відвідуваності користувач бачe вікно зображене нижче (рис.3.12.).

Create New

Full Name	Name	Present	AttendanceDate	Mark	
Дорощук Ігор Олегович	ООП	☑	18.04.2023 22:39:00	4	Edit Details Delete
Дорощук Ігор Олегович	Англійська мова	☑	18.04.2023 8:20:00	3	Edit Details Delete
Палій Іван Анатолійович	Економіка	☑	14.04.2023 16:00:00	2	Edit Details Delete
Палій Іван Анатолійович	Економіка	☑	14.04.2023 12:38:00	2	Edit Details Delete
Палій Іван Анатолійович	Англійська мова	☑	13.04.2023 15:15:00	2	Edit Details Delete
Дорощук Ігор Олегович	ООП	☑	09.04.2023 15:28:00	6	Edit Details Delete

Рис. 3.12. Список відвідуваності

Джерело: побудовано автором

3.7. Висновок до розділу 3

Розробка інтерфейсу користувача досить важливий етап створення веб-орієнтованих застосунків. Адже саме з ним взаємодіє користувач при відкритті сайту. Тому інтерфейс було виконано максимально зручним та інтуїтивно зрозумілим для забезпечення максимальної продуктивності роботи. Веб-додаток працює з базою даних, що забезпечує швидкий та надійний доступ до усіх даних. Було проведено аналіз та обґрунтування вибору конкретних компонентів для розробки додатку.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У результаті виконання даної випускної кваліфікаційної роботи було розроблений веб-орієнтований додаток обліку відвідування занять здобувачами освіти на платформі ASP.NET. Додаток був створений з використанням сучасних технологій веб-розробки.

Робота включала проектування бази даних, порівняння з аналогами. Було використано мову програмування C# та фреймворк ASP.NET для розробки серверної частини додатку, а також мову розмітки HTML, мову стилів CSS , технологію Entity Framework для генерації таблиць.

В результаті виконаної роботи було досягнуто поставлені цілі і завдання, а саме створено працездатний веб-орієнтований додаток обліку відвідування занять. Додаток показує коректну роботу всіх функцій, а також забезпечує високу продуктивність та швидкий відгук.

Пропозиції:

- Розширити функціонал додатку, додавши можливість генерації звітів по відвідуваності для кожного окремого студента.
- Вдосконалити інтерфейс додатку.
- Розробити мобільну версію додатку
- Розглянути можливості інтеграції додатку з іншими освітніми системами або платформами,

Реалізація цих пропозицій дозволить покращити функціональність, ефективність та популярність розробленого додатку, забезпечуючи зручний і надійний інструмент для обліку відвідування занять здобувачами освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Core MVC [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-gb/aspnet/core/mvc/overview?view=aspnetcore-5.0 (дата звернення 11.04.2023)
2. Entity Framework [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-gb/ef/ (дата звернення 26.04.2023)
3. Pro C# 7: With .NET and .NET Core, 8th Edition, 2017.
4. System Diagnostics [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics?view=net-7.0 (дата звернення 26.04.2023)
5. Features. [Електронний ресурс] Retrieved from Visual Paradigm: - – Режим доступу: https://www.visual-paradigm.com/features (дата звернення 27.04.2023)
6. What is Software Testing? - Definition from Techopedia. [Електронний ресурс] – Режим доступу: https://www.techopedia.com/definition/17681/software-testing (дата звернення 28.04.2023)
7. Blackburn P., Vaughn W. ADO.NET Examples and Best Practices for C# Programmers. Apress, 2002.
8. Richter J. CLR via C# Developer Reference. 4th Edition. Pearson Education.
9. Ahmed S. Use Case Diagram Generation Through Parsed Use Case Text. Master's thesis collection, Dept. of Computer Engineering and Computer Science, Long Beach California State University, 2013.

ДОДАТКИ

ДОДАТОК А

Код класу Attendance

```
public class Attendance
  {
    public int AttendanceId { get; set; }
    public int EnrollmentID { get; set; }
    public bool Present { get; set; }
    public DateTime AttendanceDate { get; set; }
    [Range(1, 50)]
    public int? Mark { get; set; }
    public Enrollment Enrollment { get; set; }
  }
```

ДОДАТОК Б

Код класу Course

```
public class Course
  {
    public int CourseId { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Enrollment> Enrollments { get; set; }
  }
```

ДОДАТОК В

Код класу Enrollment

```
  public class Enrollment{
    public int EnrollmentId { get; set; }
    public int CourseId { get; set; }
    public int StudentId { get; set; }
    public int Credit { get; set; }
    public virtual Course Course { get; set; }
    public virtual Student Student { get; set; }
```

```
    public virtual ICollection<Attendance> Attendances { get; set; } }
```

ДОДАТОК Г

Код класу Student

```
public class Student
  {
    public int StudentId { get; set; }
    public int FacultyId { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
    public string Patronomic { get; set; }
    public string FullName => $"{Surname} {Name} {Patronomic}";
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }
    [DataType(DataType.PhoneNumber)]
    public string PhoneNumber { get; set; }
    [Range(1,6)]
    public int StudyingCourse { get; set; }
    public int GroupNumber { get; set; }
    public string Specialty { get; set; }
    public Faculty Faculty { get; set; }
    public virtual ICollection<Enrollment> Enrollments { get; set; }
  }
```

ДОДАТОК Д

Код класу Teacher

```
public class Teacher
  {
    public int TeacherId { get; set; }
    public int FacultyId { get; set; }
    public string Name { get; set; }
```

```
    public string Surname { get; set; }
    public string Patronomic { get; set; }
    public string FullName => $"{Surname} {Name} {Patronomic}";
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }
    [DataType(DataType.PhoneNumber)]
    public string PhoneNumber { get; set; }
    public string Degree { get; set; }
    public string ClassLocation { get; set; }
    public Faculty Faculty { get; set; }
  }
```

ДОДАТОК Е

Код класу Faculty

```
  public class Faculty
  {
    public int FacultyId { get; set; }
    public string Name { get;set; }
    public string Location { get; set; }
    public virtual ICollection<Student> Students { get; set; }
    public virtual ICollection<Teacher> Teachers { get; set; }
  }
```

ДОДАТОК Ж

Код класу AttendancesController

```
public class AttendancesController : Controller
  {
    private readonly ApplicationDbContext _context;

    public AttendancesController(ApplicationDbContext context)
    {
      _context = context;
    }
```

```
        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Attendances.Include(a =>
a.Enrollment)
                .ThenInclude(b => b.Student)
                .Include(c => c.Enrollment.Course)
                .OrderByDescending(d => d.AttendanceDate);
            return View(await applicationDbContext.ToListAsync());
        }

        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Attendances == null)
            {
                return NotFound();
            }

            var attendance = await _context.Attendances
                .Include(a => a.Enrollment)
                .FirstOrDefaultAsync(m => m.AttendanceId == id);
            if (attendance == null)
            {
                return NotFound();
            }

            return View(attendance);
        }
        public IActionResult Create()
        {
            ViewData["EnrollmentID"] = new
SelectList(_context.Enrollments.Select(e => new
        {
            EnrollmentId = e.EnrollmentId,
            ShortDescription = $"{e.Student.FullName} - {e.Course.Name}"
        }), "EnrollmentId", "ShortDescription");
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("AttendanceId,EnrollmentID,Present,AttendanceDate,Mark")]
Attendance attendance)
```

```
        {
            _context.Add(attendance);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Attendances == null)
            {
                return NotFound();
            }
            var attendance = await _context.Attendances.FindAsync(id);
            if (attendance == null)
            {
                return NotFound();
            }
            ViewData["EnrollmentID"] = new
SelectList(_context.Enrollments.Select(e => new
            {
                EnrollmentId = e.EnrollmentId,
                ShortDescription = $"{e.Student.FullName} - {e.Course.Name}"
            }), "EnrollmentId", "ShortDescription");
            return View(attendance);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("AttendanceId,EnrollmentID,Present,AttendanceDate,Mark")]
Attendance attendance)
        {
            if (id != attendance.AttendanceId)
            {
                return NotFound();
            }
            try
            {
                _context.Update(attendance);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!AttendanceExists(attendance.AttendanceId))
                {
                    return NotFound();
```

```
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null || _context.Attendances == null)
        {
            return NotFound();
        }

        var attendance = await _context.Attendances
            .Include(a => a.Enrollment)
            .FirstOrDefaultAsync(m => m.AttendanceId == id);
        if (attendance == null)
        {
            return NotFound();
        }
        return View(attendance);
    }
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        if (_context.Attendances == null)
        {
            return Problem("Entity set 'ApplicationDbContext.Attendances'  is
null.");
        }
        var attendance = await _context.Attendances.FindAsync(id);
        if (attendance != null)
        {
            _context.Attendances.Remove(attendance);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    private bool AttendanceExists(int id)
```

```
        {
            return (_context.Attendances?.Any(e => e.AttendanceId ==
id)).GetValueOrDefault();
        }
    }
}
```

ДОДАТОК И

Код класу CoursesController

```
public class CoursesController : Controller
    {
        private readonly ApplicationDbContext _context;
        public CoursesController(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<IActionResult> Index()
        {
            return _context.Courses != null ?
                        View(await _context.Courses.ToListAsync()) :
                        Problem("Entity set 'ApplicationDbContext.Courses'  is
null.");
        }
        [Authorize(Roles = "Admin,Teacher")]
        public IActionResult Create()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("CourseId,Name")]
Course course)
        {
            _context.Add(course);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        [Authorize(Roles = "Admin,Teacher")]
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Courses == null)
            {
                return NotFound();
            }
```

```
            var course = await _context.Courses.FindAsync(id);
            if (course == null)
            {
                return NotFound();
            }
            return View(course);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("CourseId,Name")] Course course)
        {
            if (id != course.CourseId)
            {
                return NotFound();
            }

            try
            {
                _context.Update(course);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!CourseExists(course.CourseId))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        [Authorize(Roles = "Admin,Teacher")]
        // GET: Courses/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Courses == null)
            {
                return NotFound();
            }
```

```
            var course = await _context.Courses
                .FirstOrDefaultAsync(m => m.CourseId == id);
            if (course == null)
            {
                return NotFound();
            }

            return View(course);
        }

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

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

        private bool CourseExists(id)
        {
            return (_context.Courses.Any(e => e.CourseId ==
id)).GetValueOrDefault();
        }
    }
}
```

ДОДАТОК К

Код класу StudentsController

```
public class StudentsController : Controller
    {
        private readonly ApplicationDbContext _context;
```

```
        public StudentsController(ApplicationDbContext context)
        {
            _context = context;
        }
        public ViewResult Index(string searchString)
        {
            var students = from s in _context.Students
                           select s;
            if (!String.IsNullOrEmpty(searchString))
            {
                students = students.Where(s => s.Name.Contains(searchString)
                                       || s.Surname.Contains(searchString)
                                       || s.Patronomic.Contains(searchString));
            }
            return View(students.ToList());
        }

        [Authorize(Roles = "Admin,Teacher")]
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Students == null)
            {
                return NotFound();
            }

            var student = await _context.Students
                .Include(s => s.Faculty)
                .FirstOrDefaultAsync(m => m.StudentId == id);
            if (student == null)
            {
                return NotFound();
            }
            return View(student);
        }
        [Authorize(Roles = "Admin,Teacher")]
        public IActionResult Create()
        {
            ViewData["FacultyId"] = new SelectList(_context.Faculties,
"FacultyId", "Name");
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> Create(Student student)
        {
            _context.Students.Add(student);
            await _context.SaveChangesAsync();

            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var student = await _context.Students.FindAsync(id);
            if (student == null)
            {
                return NotFound();
            }
            ViewBag.FacultyId = new SelectList(_context.Faculties,
"FacultyId", "Name", student.FacultyId);
            return View(student);
        }
[HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("StudentId,FacultyId,Name,Surname,Patronomic,Email,PhoneNumbe
r,StudyingCourse,GroupNumber,Specialty")] Student student)
        {
            if (id != student.StudentId)
            {
                return NotFound();
            }
            try
            {
                _context.Update(student);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!StudentExists(student.StudentId))
                {
                    return NotFound();
                }
```

```
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }


    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null || _context.Students == null)
        {
            return NotFound();
        }

        var student = await _context.Students
            .Include(s => s.Faculty)
            .FirstOrDefaultAsync(m => m.StudentId == id);
        if (student == null)
        {
            return NotFound();
        }

        return View(student);
    }

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

        await _context.SaveChangesAsync();
```

```
            return RedirectToAction(nameof(Index));
        }

        private bool StudentExists(int id)
        {
            return (_context.Students?.Any(e => e.StudentId ==
id)).GetValueOrDefault();
        }
    }
}
```

ДОДАТОК Л

Код класу EnrollmentsController

```
public class EnrollmentsController : Controller
    {
        private readonly ApplicationDbContext _context;

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

        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Enrollments.Include(e =>
e.Course).Include(e => e.Student);
            return View(await applicationDbContext.ToListAsync());
        }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Enrollments == null)
            {
                return NotFound();
            }

            var enrollment = await _context.Enrollments
                .Include(e => e.Course)
                .Include(e => e.Student)
                .FirstOrDefaultAsync(m => m.EnrollmentId == id);
            if (enrollment == null)
            {
                return NotFound();
            }
            return View(enrollment);
```

```
    }
    [Authorize(Roles = "Admin,Teacher")]
    public IActionResult Create()
    {
        ViewData["CourseId"] = new SelectList(_context.Courses,
"CourseId", "Name");
        ViewData["StudentId"] = new SelectList(_context.Students,
"StudentId", "FullName");
        return View();
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("EnrollmentId,CourseId,StudentId,Credit")] Enrollment
enrollment)
    {
        _context.Add(enrollment);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    [Authorize(Roles = "Admin,Teacher")]

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

        var enrollment = await _context.Enrollments.FindAsync(id);
        if (enrollment = null)
        {
            return NotFound();
        }
        ViewData["CourseId"] = new SelectList(_context.Courses,
"CourseId", "Name", enrollment.CourseId);
        ViewData["StudentId"] = new SelectList(_context.Students,
"StudentId", "Name", enrollment.StudentId);
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> Edit(int id,
[Bind("EnrollmentId,CourseId,StudentId,Credit,ShortDescription")]
Enrollment enrollment)
        {
            if (id != enrollment.EnrollmentId)
            {
                return NotFound();
            }
                try
                {
                    _context.Update(enrollment);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!EnrollmentExists(enrollment.EnrollmentId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
        }

        [Authorize(Roles = "Admin,Teacher")]
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Enrollments == null)
            {
                return NotFound();
            }

            var enrollment = await _context.Enrollments
                .Include(e => e.Course)
                .Include(e => e.Student)
                .FirstOrDefaultAsync(m => m.EnrollmentId);
            if (enrollment == null)
            {
                return NotFound();
            }
            return View(enrollment);
```

```
        }

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

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

        private bool EnrollmentExists(int id)
        {
          return (_context.Enrollments?.Any(e => e.EnrollmentId ==
id)).GetValueOrDefault();
        }
    }
}
```