МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: 122 «Web-додаток підтримки діяльності навчального процесу в ЗОШ»

_____

Здобувача (ки) групи ІТ-91 Обраменка Дмитра Сергійовича
(шифр групи) (Прізвище, ім'я та по батькові)

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

_____ Дмитро ОБРАМЕНКО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент кафедри ІТ, к.т.н., доцент Володимир НАГОРНИЙ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ

В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО

«__» _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Обраменку Дмитру Сергійовичу

1 Тема роботи *Web-додаток підтримки діяльності навчального процесу в ЗОШ*

керівник роботи *Нагорний Володимир В'ячеславович, к.т.н., доцент* ,

затверджені наказом по університету від «*29*» травня 2023 р. № 0588-IV

2 Строк подання студентом роботи «7» червня 2023 р.

3 Вхідні дані до роботи технічне завдання на розробку web-додатку підтримки діяльності навчального процесу в ЗОШ

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, моделювання та проектування, розробка web-додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Аналіз предметної області. Актуальність роботи. Дослідження аналогів. Постановка задачі. Діаграма IDEF0, діаграма декомпозиції, діаграма варіантів використання, діаграма діяльності. Логічна модель даних. Архітектура web-додатку. Демонстрація web-додатку.

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 15.10.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної області	20.03.2023-24.03.2023	Виконано
2	Оформлення планування робіт	27.03.2023-31.03.2023	Виконано
3	Оформлення технічного завдання	03.04.2023-06.04.2023	Виконано
4	Вибір засобів реалізації	07.04.2023-07.04.2023	Виконано
5	Моделювання та проектування web-додатку	10.04.2023-29.04.2023	Виконано
6	Розробка web-додатку	20.04.2023-25.05.2023	Виконано
7	Тестування web-додатку	26.05.2023-28.05.2023	Виконано
8	Оформлення пояснювальної записки	29.05.2023-05.06.2023	Виконано

Студент _____ Дмитро ОБРАМЕНКО
(підпис)

Керівник роботи _____ к.т.н., доц. Володимир НАГОРНИЙ
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Web-додаток підтримки діяльності навчального процесу в ЗОШ».

Пояснювальна записка складається зі вступу, трьох розділів, висновку, списку використаних джерел із 24 найменувань, трьох додатків. Загальний обсяг робіт – 88 сторінок, у тому числі 36 сторінок основного тексту, 3 сторінки списку використаних джерел, 48 сторінок додатків.

Кваліфікаційну роботу бакалавра присвячено розробці web-додатку підтримки діяльності навчального процесу в ЗОШ.

У першому розділі проведено аналіз предметної області, де визначено актуальність даної проблеми. Розглянути аналоги web-додатків підтримки діяльності навчального процесу. На основі розробки першого розділу було визначено мету та поставку задачі.

У другому розділі проведено структурно-функціональне моделювання. Було розроблено контекстну діаграму в нотації IDEF0 та її декомпозицію, діаграму варіантів використання для показу користувачів, процесів та взаємозв’язків між ними. Було створено діаграму діяльності, де показано, як кожен користувач може взаємодіяти з web-додаток. логічну модель даних. Також, розроблено логічну модель даних.

У третьому розділі описано архітектуру web-додатку та саму розробку. Також показано приклади використання web-додатку для різних типів користувачів.

Ключові слова: web-додаток, розробка, моделювання, навчальний процес, школа, журнал, щоденник, база даних, ASP.NET Core 6 MVC, Bootstrap, MS SQL.

ЗМІСТ

ВСТУП

У час популяризації інформаційних технологій ні одна галузь у світі не відстає від цього тренду. Використання смартфонів, комп'ютерів та інших гаджетів – це велика частина повсякдення для кожної людини, яка живе в цивілізованому світі.

Не виключенням є сфера освіти. Дана галузь дуже змінилися за останній час, зокрема з 2020 року, коли навчальний процес повністю перейшов на онлайн формат. Через сьогоденні події, більшість українських шкіл впровадили змішану форму навчання, але все ж таки велика частка залишається на постійному дистанційному навчанні. Ці умови підштовхують до розуміння того, що без сучасних технологій теперішній освітній процес неможливий.

Тут виникає проблема в пошуку якісних інструментів, за допомогою яких можна ефективно налагодити навчальний процес. Адміністрації шкіл шукають сервіси для зручного керування усіма функціями освітнього процесу, вчителі хочуть мати зручний інтерфейс аби записувати оцінки та домашні завдання, учні хочуть мати змогу переглядати свою успішність в будь-який момент часу. Увесь цей функціонал бажано тримати в одному місці, проте на просторах українського інтернету мало програмних продуктів, які б задовільнили ці потреби. Через це, учасники навчального процесу використовують різні інструменти для різних функцій. Це є незручною практикою.

Гарним рішенням даної проблеми був би web-додаток, який забезпечить усі вищезгадані потреби та функціонал, необхідний для ефективної діяльності навчального процесу [1].

Мета даного проекту – розробка зручного та простого в освоєнні web-додатку, що буде підтримувати навчальний процес в ЗОШ.

Для того, щоб мета була досягнута, необхідно виконати наступні задачі:

- Проаналізувати предметну область
- Виконати огляд web-додатків для електронних журналів, якими користуються школи
- Сформулювати функціональні вимоги
- Виконати моделювання та проектування
- Виконати програмну реалізацію

На основі результатів даної роботи, було опубліковано тезу на конференцію «ІМА – 2023» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень та публікацій

Інформаційні технології в галузі освіти використовуються давно: електронні підручники, інтерактивні онлайн завдання, відеоуроки тощо. Проте, внаслідок карантину за останні декілька років використання сучасних технологій в сфері освіти набуло ще більших масштабів.

Онлайн формат навчання зумовив необхідність використання електронних журналів та щоденників, хоча в деяких країнах світу вони вже давно впровадженні. Електронні бази даних про навчальний процес введено в розвинених європейських, азійських країнах, а також в США [2]. Проте, попри сьогоденні обставини, не всі українські школи використовують електронні сервіси для підтримки діяльності навчального процесу – всього 46 відсотків [3]. Це ускладнює можливість систематично відслідковувати успішність учнів під час дистанційного навчання. Виходячи з цього, близько половини вчителів ведуть паперовий журнал, рахуючи самостійно кількість пропущених занять, тематичні та семестрові оцінки. У свою чергу, учні не можуть переглянути свої оцінки в будь-який момент, так як не всі школярі в наш час ходять до школи. Це також дає певні незручності і батькам, так як щоб дізнатися відомості про присутність, дисципліну їхньої дитини треба напряму зв’язуватися з вчителем.

Перехід від паперових щоденників та журналів до електронних має наступні переваги [4]:

- Єдина база даних з усією інформацією про навчальний процес
- Конфіденційність. Кожен учень має доступ лише до своїх оцінок
- Відкритість. Батьки бачать за що виставлена оцінку з можливим коментарем

1.2 Аналіз програмних продуктів-аналогів

Web-додатків електронних журналів та щоденників дуже мало в просторах українського інтернету. Проте, вони мають досить велику популярність. Але, все ж таки, у адміністрації шкіл не великий вибір серед даних програмних продуктів.

Щоб визначити функціонал майбутнього програмного продукту було проведено дослідження існуючих аналогів web-додатків підтримки діяльності навчального процесу. Найпопулярнішою платформою серед таких web-додатків є публічний сайт «Нові знання», який використовується близько 55% шкіл [5], які використовують електронні журнали. Також, було розглянуто і інші, менш популярні аналоги, такі як «e-school» та «Smart School».

Почнемо з очевидного лідера серед використання школами – «Нові знання» [6]. Даний web-додаток має доволі застарілий інтерфейс та широкий спектр можливостей, що може ускладнити процес ознайомлення. Розподіляється 4 типи користувачів: адміністратор, вчителі, учні, батьки.

У відкритому доступі немає прикладів, як виглядає електронний журнал та щоденник, проте, функціонал web-додатку можна переглянути на відео-інструкціях користування ним на YouTube. Електронний журнал зображено на Рисунок 1.1. Як видно з рисунку, теми уроків описані знизу, немає наглядних розділень уроків по місяцях або тижнях. Також, що найбільше «кидається» в очі – так це велика кількість кнопок та бокове меню, яке виглядає негармонічно. Електронний щоденник має однаковий вигляд як для учнів так і для батьків. Як видно з Рисунок 1.2 він має доволі незручний вигляд, не як у паперовому форматі, відсутня колонка для оцінок та їх пояснень. На Рисунок 1.3 зображено сторінку адміністратора, на якій він може управляти навчальним процесом.

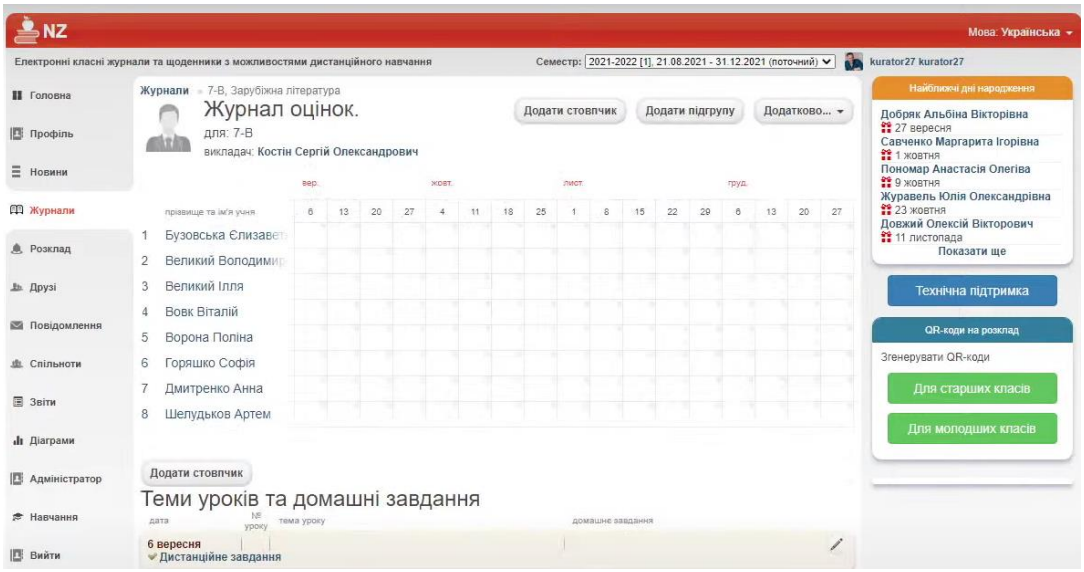

Рисунок 1.1 – Електронний журнал «Нові знання»

Рисунок 1.2 – Електронний щоденник «Нові знання»

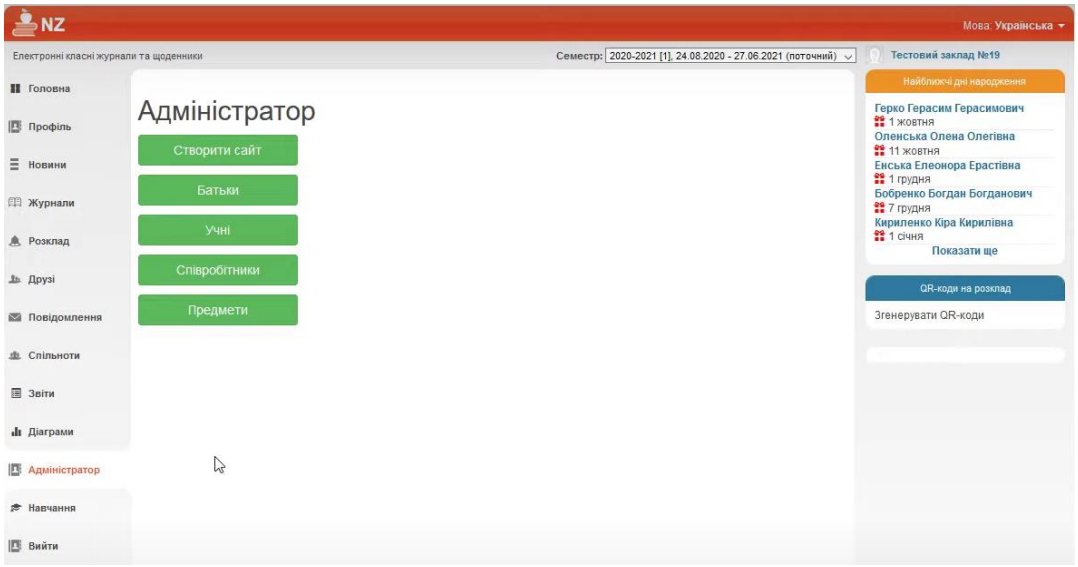

Рисунок 1.3 – Кабінет адміністратора «Нові знання»

Наступний аналог – проект від e-school [7]. Сторінки головних можливостей даного програмного продукту виконані в доволі примітивному, зрозумілому стилі, що є великим плюсом даного додатку. Проте, сам стиль деяких частин сайту не відповідає сучасним тенденціям.

Таблиця журналу (Рисунок 1.4) виконана дуже якісно і зрозуміло. Гарною особливістю є відображення середньої оцінки учнів.

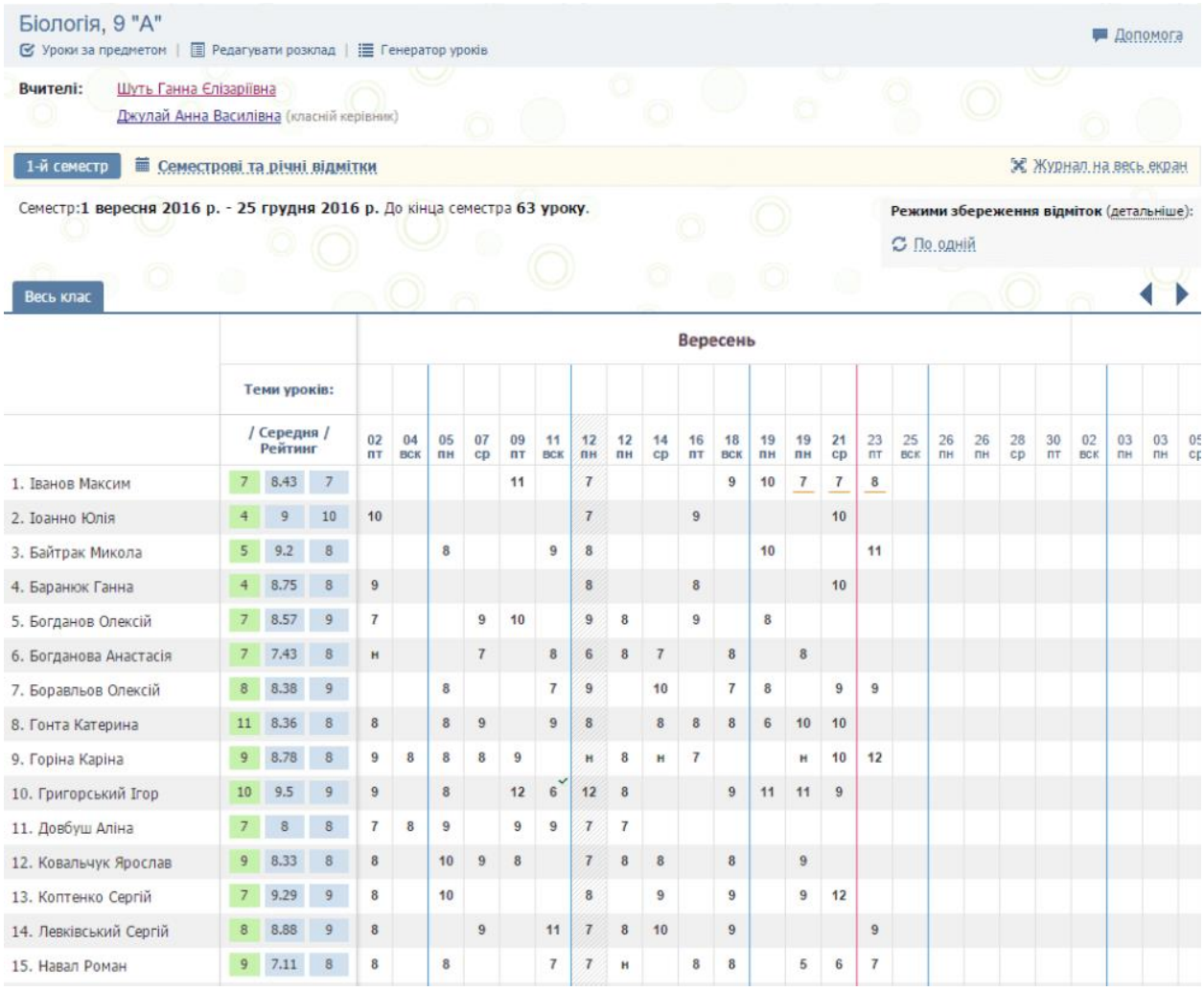

Рисунок 1.4 – Електронний журнал «e-school»

Електронний щоденник (Рисунок 1.5) також має доволі зрозумілу конструкцію: перегляд оцінок, розкладу та домашніх завдань. Усе так само, як і в паперовому аналогу. Проте, недоліком є те, що не вказано за що була поставлена оцінка.

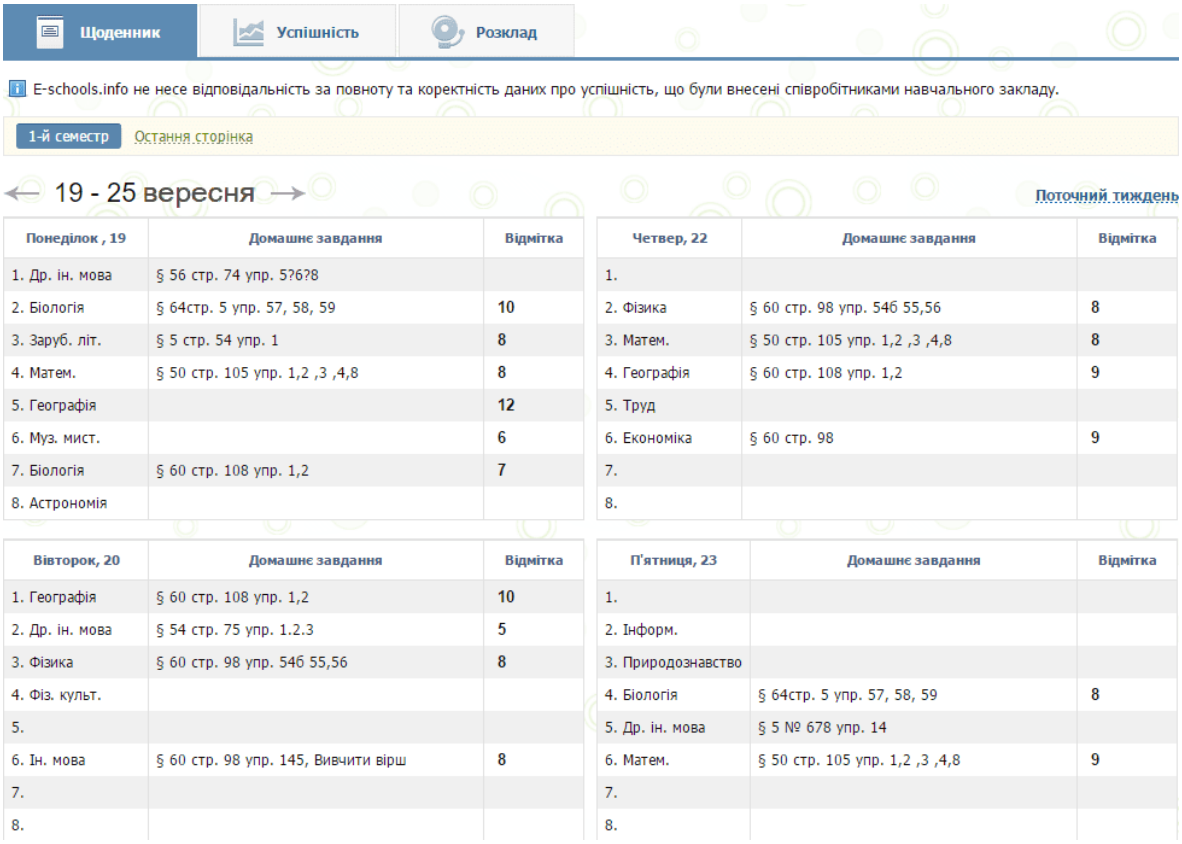

Рисунок 1.5 – Електронний щоденник «e-school»

Даний web-додаток має унікальну сторінку, під назвою «Успішність», яка зображена на Рисунок 1.6, що дає змогу переглянути всі власні бали по всім предметам на одній сторінці.

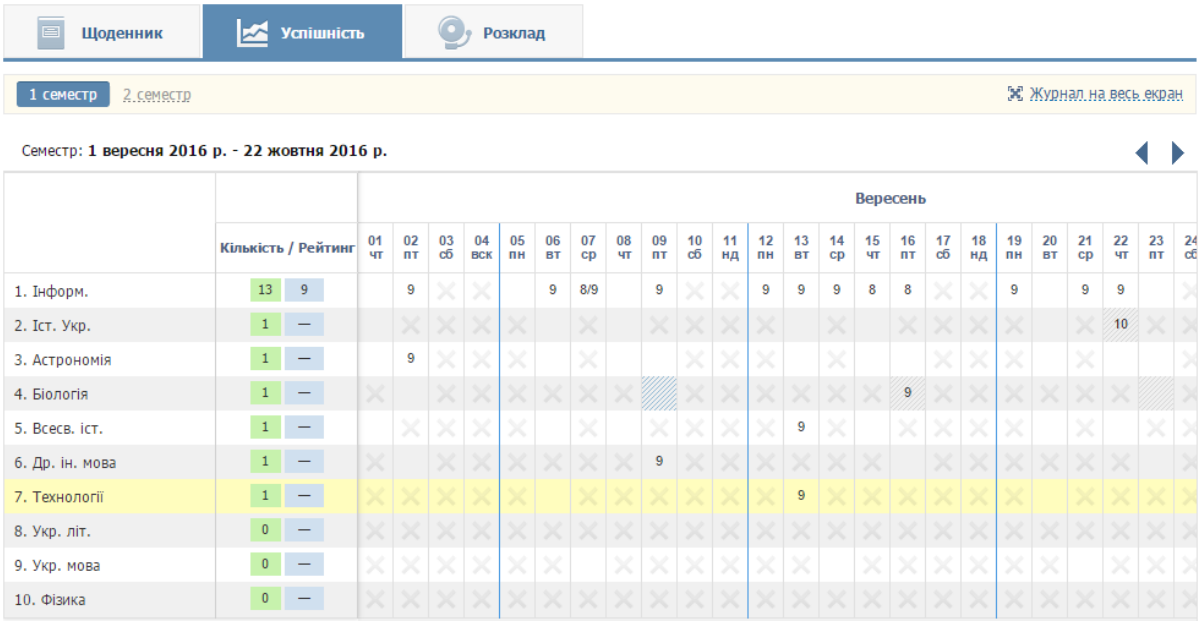

Рисунок 1.6 – Таблиця успішності «e-school»

Останній розглянутий аналог – це web-додаток Smart School [8]. Він виконаний в кольорах синього відтінку. Функціонал дуже схожий на той, який був у вищеописаних аналогах. Розділено 4 типи користувачів: адміністрація, вчителі, учні та батьки. У кожного різний функціонал, проте у батьків та учнів однаковий.

Електронний журнал, який зображено на Рисунок 1.7, виконано дуже красиво, проте є деякі недоліки: незручно наведено дату заняття та відсутня середня оцінка для кожного учня. Проте, слід зауважити, можна записати за урок як оцінку, так і будь-яке зауваження, наприклад відсутність на уроці або запізнення.

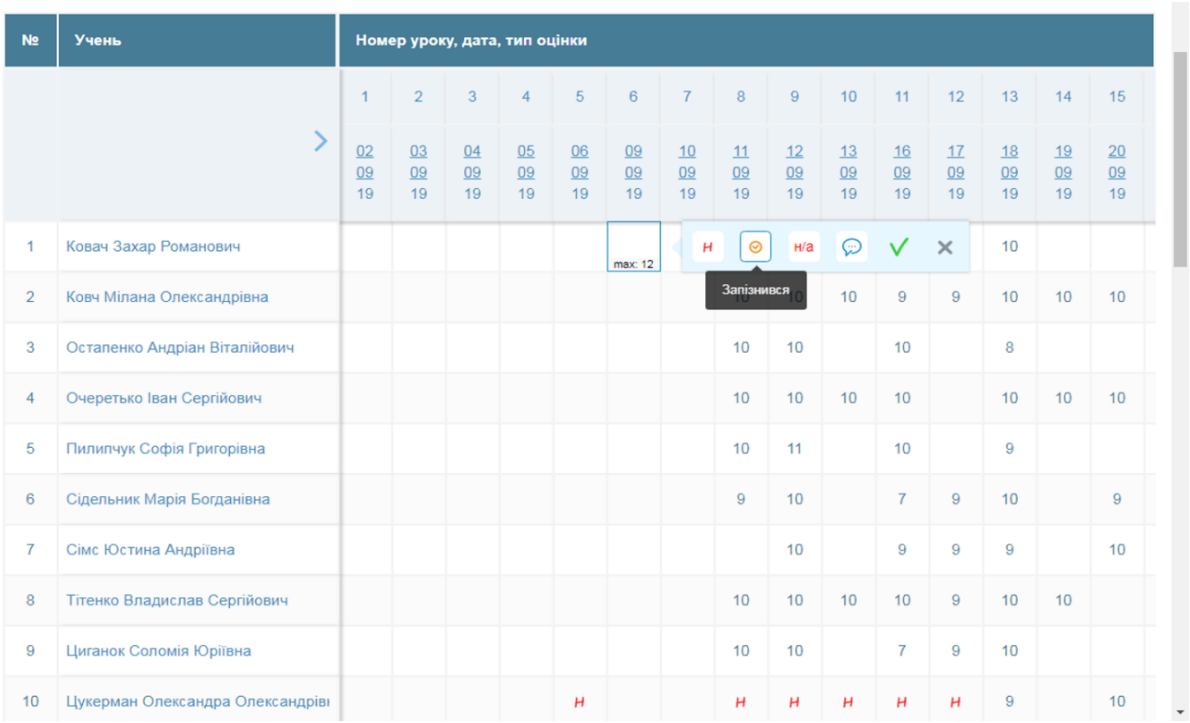

Рисунок 1.7 – Електронний журнал «Smart School»

Електронний щоденник, зображений на Рисунок 1.8, має менш зручніший інтерфейс ніж журнал, проте достатньо інтуїтивно зрозумілий.

Слід зауважити, що даний web-додаток має графіки успішності, проте окремої наглядної таблиці успішності немає.

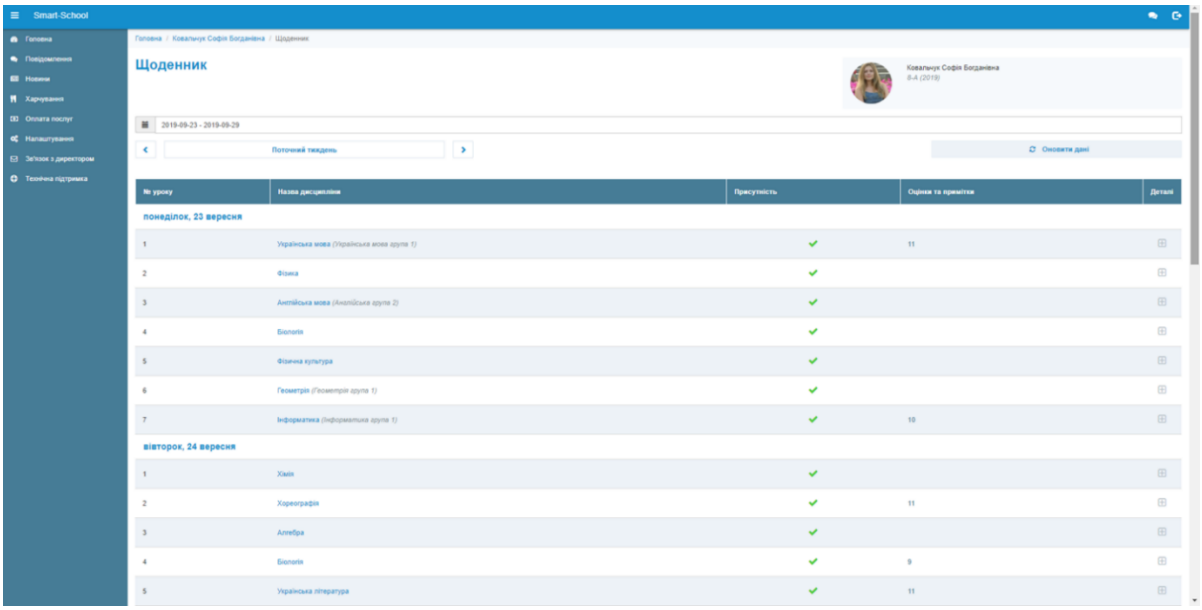

Рисунок 1.8 – Електронний щоденник «Smart School»

Після аналізу аналогів web-додатків підтримки діяльності навчального процесу в ЗОШ, було визначено їх переваги та недоліки. Результати представлені в Таблиця 1.1.

Таблиця 1.1 – Порівняльна таблиця характеристик аналогів web-додатків супроводження начального процесу

Web-додаток / Характеристика	e-school	Нові знання	Smart School
Сучасний дизайн	+	-	+
Навігація	+	-	+
Зручний інтерфейс	+	+	+
Електронний щоденник	+	+	+
Електронний журнал	+	+	+
Наявність розкладу	+	+	+
Таблиця успішності	+	-	-
Можливість входу як вчитель	+	+	+
Можливість входу як учень	+	+	+
Можливість входу для батьків	-	+	+

Продовження таблиці 1.1

Web-додаток / Характеристика	e-school	Нові знання	Smart School
Можливість запису відомостей про учня (поведінка, присутність тощо)	-	-	-

З Таблиця 1.1 чітко видно, що необхідно мати програмному продукту, а також на який функціонал треба звернути увагу, розроблюючи новий web-додаток. Він повинен мати сучасний, зручний, інтуїтивно зрозумілий дизайн та гарну навігацію. Щодо функціоналу, то попри основні можливості слід виділити наявність таблиці успішності, кабінету батьків, можливість запису відомостей про учнів.

Далі описано вимоги, яких необхідно дотримуватися при створенні web-додатку:

- Розподілити користувачів на 4 типи з різними правами: адміністратор, вчитель, учень, батьки
- Організувати роботу адміністратора: створення нових користувачів, управління класами та розкладом
- Забезпечити можливість ведення електронного журналу, як аналогу паперового формату
- Забезпечити можливість ведення електронного щоденника: перегляд оцінок та домашніх завдань
- Забезпечити застосування функції інтерактивної організації розкладу
- Створити таблицю успішності для перегляду учнями та їх батьками

У ДОДАТОК А представлено технічне завдання, де описано всі вимоги до проекту. Планування робіт наведено у ДОДАТОК Б.

1.3 Постановка задачі

Мета даного проекту – розробка web-додатку підтримки діяльності навчального процесу в ЗОШ. Функціонал повинен бути мінімальним, але в той же час задовольняти усі потреби. Даний web-додаток повинен забезпечити належну роботу електронного журналу та щоденника, що надасть змогу вчителям та учням спростити їх роботу.

Щоб мета проекту була досягнута, необхідно виконати нижчеописані задачі:

- Дослідити предметну область для визначення актуальності розробки
- Проаналізувати аналоги, виділивши їх переваги та недоліки
- Сформулювати функціональні вимоги
- Виконати моделювання та проектування
- Створити базу даних
- Розробити back-end частину
- Розробити front-end частину
- Виконати тестування

Web-додаток буде реалізований за допомогою технології ASP.NET Core 6 MVC. Дана платформа дозволяє розробляти як front-end частину, використовуючи JavaScript та інші різні інструменти, наприклад Bootstrap, так і back-end частину. Також за допомогою цього інструменти можна легко організувати авторизацію та реєстрацію користувачів, розбиваючи їх на ролі за допомогою системи Identity. Уся інформація буде зберігатися в СУБД MS SQL. Організація роботи з базою даних буде виконана через бібліотеку Entity Framework Core.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

2.1 Структурно-функціональне моделювання

Важливим етапом при розробці програмного продукту є структурно-функціональне моделювання. Воно допомагає краще зрозуміти систему та оптимізувати її, зекономивши час та ресурси.

Щоб представити функціональне моделювання використовують такий інструмент як IDEF0. Дана методологія створена для опису систем і процесів, які взаємопов’язані між собою [9]. Вона складається з блоків (процеси та функції) та стрілок (вхідні та вихідні дані).

На Рисунок 2. зображена функціональна діаграма, процесом якої є підтримка діяльності навчального процесу в ЗОШ.

Рисунок 2.1 – Діаграма IDEF0

Як видно з Рисунок 2., визначено наступні елементи діаграми IDEF0:

- Вхідні дані: логін та пароль, запит на управління навчальним процесом (від адміністратора: управління класами, розкладом та користувачами), запит на редагування журналу, запит на перегляд щоденника та(або) таблиці успішності
- Управління: інструкція по використанню web-додатка
- Механізми: web-додаток, база даних, користувач та апаратне забезпечення
- Вихід: авторизований користувач, нові та(або) оновлені дані, редагований журнал, перегляд щоденника та(або) таблиці успішності

Для відображення детальних процесів використовують декомпозицію функціонального проектування. Це дозволяє розбити функції на менші та прості елементи. Декомпозиція може мати не тільки один рівень. Даний процес не є обмеженим та може тривати доки, поки не буде виділено конкретні підпроцеси.

Декомпозиція функціональної моделі зображена на рРисунок 2..

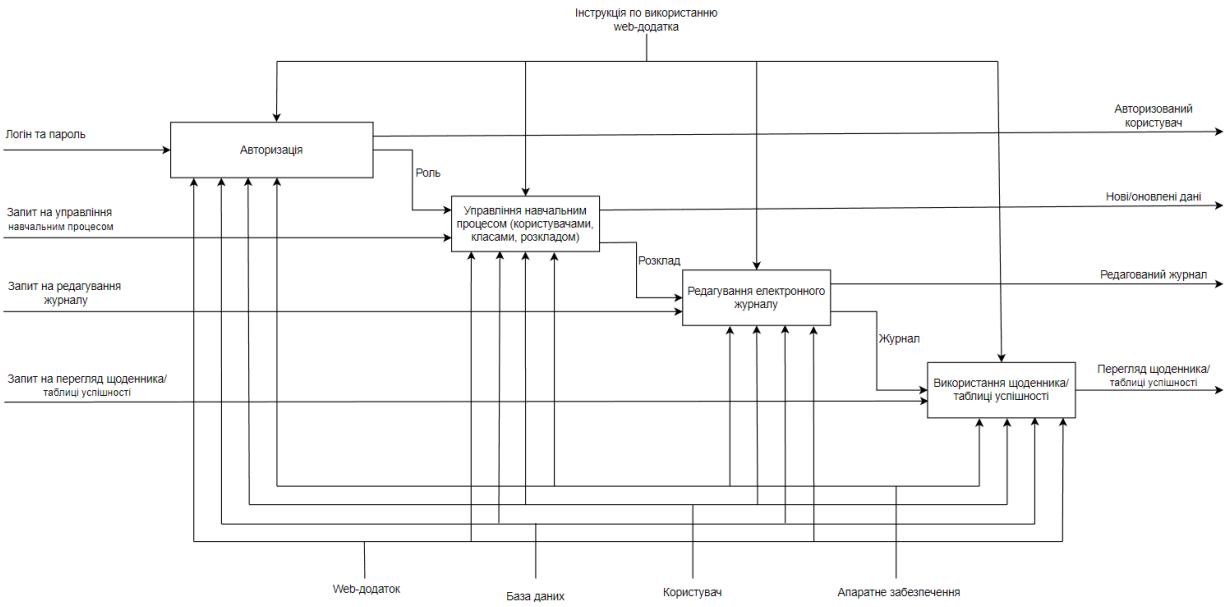

Рисунок 2.2 – Діаграма декомпозиції функціональної моделі

Дана декомпозиція розбита на 4 процеси, які йдуть послідовно і не можуть існувати без попереднього процесу: авторизація, управління навчальним процесом (користувачами, класами, розкладом), редагування журналу, використання щоденника та таблиці успішності.

2.2 Проектування web-додатку

Діаграма варіантів (Рисунок 2.) використання є важливим елементом при проектування інформаційної системи. Вона показує взаємозв'язки між акторами та системою, а також відображає функціональні вимоги зі сторони користувача [10].

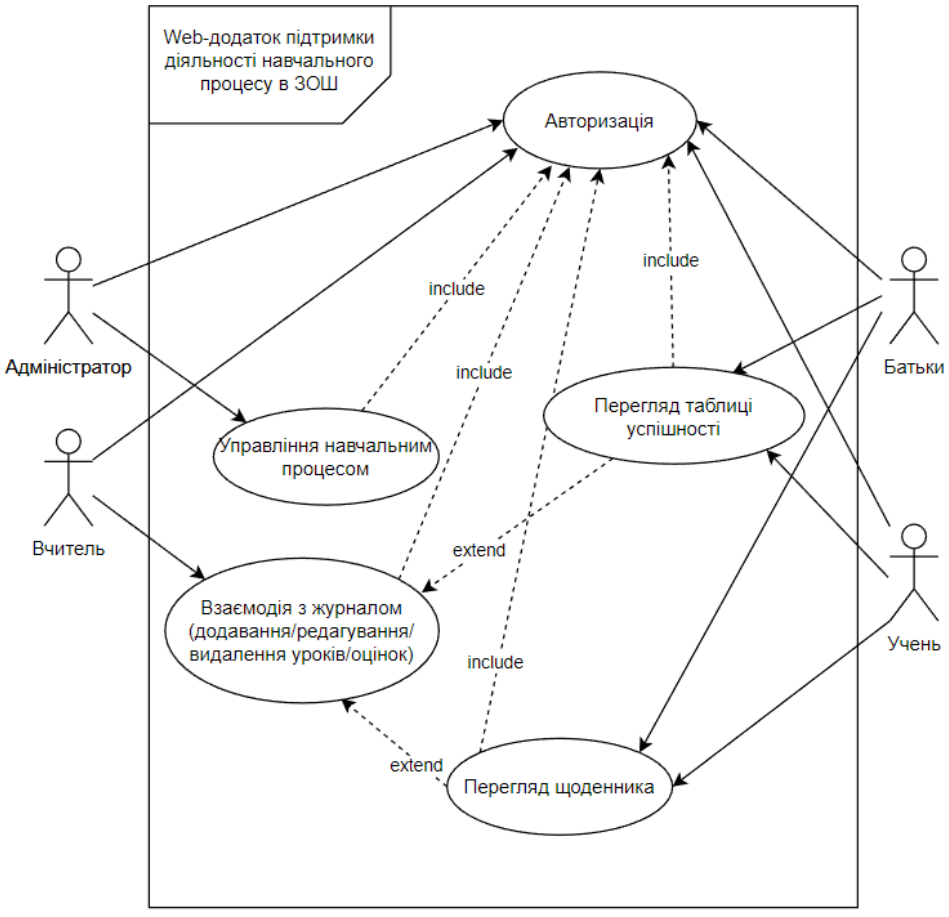

Рисунок 2.3 – Діаграма варіантів використання

Як видно з Рисунок 2. для web-додатку було виділено наступні типи користувачів:

- Адміністратор
- Вчитель
- Учень
- Батьки

Варіанти використання наведено нижче:

- Авторизація
- Управління навчальним процесом (управління класами, розкладом, іншими користувачами)
- Взаємодія з журналом
- Перегляд таблиці успішності
- Перегляд щоденника

2.3 Проектування моделі бази даних

Логічна модель даних створюється для подачі абстрактної структури інформаційної області [11]. На ній подаються сутності, їх атрибути та взаємозв’язки між ними.

Далі розглянемо кожну сутність:

- User: узагальнена сутність для усіх типів користувачів, яка має спільні атрибути
- Адміністратор (Admin)
- Вчитель (Teacher)
- Учень (Student)
- Батьки (Parent)
- Клас (Classes) – містить опис класу

- Предмет (Subjects) – зберігає опис предмету
- Розклад (Schedule) – таблиця розкладу: день тижня, номер уроку, клас, предмет
- Домашнє завдання (Homework) – зберігає опис домашнього завдання
- Оцінка (Grades) – містить опис оцінки
- Зауваження (Remarks) – описує відомості про учня (такі як поведінку, присутність)

На Рисунок 2. зображена логічна модель бази даних.

Для роботи з базою даних було обрано MS SQL.

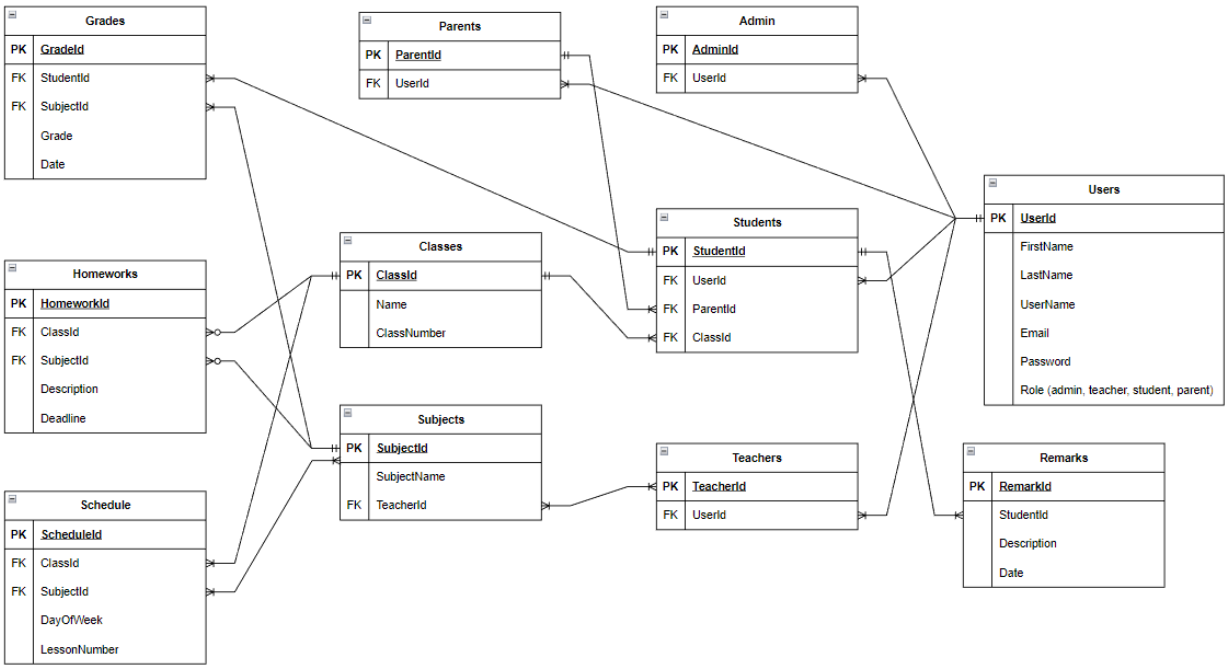

Рисунок 2.4 – Логічна модель бази даних

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Архітектура програмного продукту

Розроблюваний програмний продукт має багаторівневу структуру. Така архітектура дозволяє розділити логіку web-додатку на різні шари, що спрощує підтримку таких програмних продуктів.

Рівень Domain містить опис сутностей. Він має бути незалежним від технології та може в подальшому бути використаний в інших проектах. Даний шар створюється першим, так як тут описуються усі моделі, які будуть використовуватися в інших шарах.

Рівень DAL (Data Access Layer) створений для роботи з базою даних. Даний шар відповідає за взаємодію з даними у базі даних. Він включає два елементи:

- Контекст бази даних. Тут визначається доступ до бази даних
- Репозиторії – зчитування та запис в базу даних

Також, слід зауважити, в цьому ж рівні знаходяться міграції, які можуть виконуватися в будь-який момент часу розробки.

Рівень Service. Це середній шар, що містить усю бізнес логіку програми [12]. Він взаємодіє з репозиторієм, що знаходиться в рівні DAL та рівнем Presentation, який описано нижче.

Presentation Layer – це найвищий рівень, де користувачі напряму взаємодіють з web-додатком [12]. У випадку з даним розроблюваним web-додатком цей рівень приміняє паттерн MVC [13]. У даному шарі головними елементами є views (представлення) та controllers (контролери). Views відповідають за візуальну частину web-додатку, тобто інтерфейс. Controllers, в свою чергу, приймають вхідні дані від користувача та повертають відповідь. Models зберігають класи для роботи з представленнями.

На Рисунок 3. зображено архітектуру web-додатку [12].

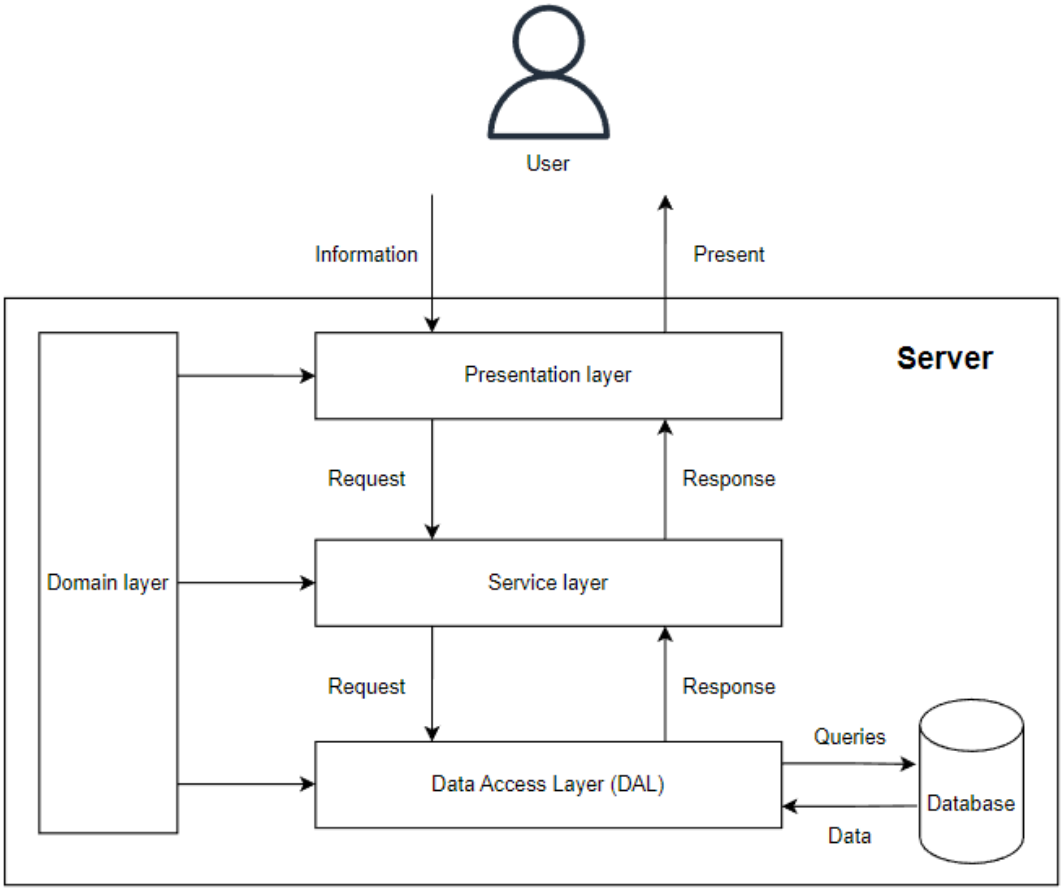

Рисунок 3.1 – Архітектура web-додатку

3.2 Програмна реалізація

Розробка web-додатку здійснювалася у середовищі Visual Studio 2022, технологією ASP.NET Core 6. Робота з базою даних здійснювалася через MS SQL. Використовувалося асинхронне програмування, що дозволяє виконувати декілька операцій паралельно.

Першим етапом розробки було створення бази даних. Для роботи з базою даних було обрано бібліотеку Entity Framework Core, а також модель розробки «Code First» [14]. Даний підхід дозволяє спочатку створити сутності (на рівні Domain), а потім створити контекст бази даних (рівень

DAL). Останнім кроком даного підходу є міграції. Їх можна виконувати незалежно від етапу розробки програмного продукту.

Слід зауважити, що для роботи з користувачами було обрано бібліотеку Identity [15], яка керує користувачами, паролями, ролями. Було створено сутність User, яка унаслідується від IdentityUser та додано властивості ім’я та прізвище:

```
public class User : IdentityUser
{
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
}
```

Усі типи користувачів, будуть унаслідуватися від класу User, тобто мати ті властивості, які визначені в User та IdentityUser. Наприклад, розглянемо модель Admin:

```
public class Admin : User
{
}
```

Зв’язки між сутностями Entity Framework Core розпізнає автоматично. Наприклад, нижче показано зв’язок один до багатьох між сутностями студент та шкільний клас:

```
public class SchoolClass
{
        public int Id { get; set; }
        public virtual ICollection<Student> Students { get; set;}
}
```

```
public class Student : User
{
        public virtual SchoolClass SchoolClass { get; set; }
}
```

Один клас повинен мати щонайменше одного студента; один студент повинен відноситися до одного класу.

Важливим елементом є визначення ролей. Для їх налаштування було створено окремий клас RoleConfiguration який унаслідується від IEntityTypeConfiiguration<IdentityRole>. У даному класу перевизначається метод Configure, у якому створюються нові ролі. Наступним чином створюється роль адміністратора:

```
public class RoleConfiguration
:IEntityTypeConfiguration<IdentityRole>{
        public void Configure(EntityTypeBuilder<IdentityRole>
builder){
            builder.HasData(
                new IdentityRole
                {
                    Name = "Admin",
                    NormalizedName = "ADMIN"
                }}}
```

Аналогічним чином, через метод HasData створюються інші ролі, такі як вчитель, учень, батьки.

Так як використовується Identity, клас контексту бази даних (**Error! Reference source not found.**) повинен унаслідуватися від IdentityDbContext<User>. Щоб зробити додаткові налаштування, необхідно перевизначити метод OnModelCreating, де вказати необхідну конфігурацію. Нижче показано як визнаються таблиці в базі даних у класі контексту на прикладу 5 сутностей:

```
public DbSet<Admin> Admin { get; set; }
public DbSet<Student> Student { get; set; }
public DbSet<Teacher> Teacher { get; set; }
public DbSet<Parent> Parent { get; set; }
public DbSet<SchoolClass> SchoolClass { get; set; }
```

Перед останнім кроком налаштування бази даних, тобто міграцією, необхідно додати рядок підключення до БД у файлі appsettings.json:

```
"ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-
APHBGUT\\SQLEXPRESS;Database=School;Trusted_Connection=True;Multi
pleActiveResultSets=True"
}
```

Також треба налаштувати сервіси в головному файлі програми Program.cs:

```
string connection =
builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(connection));
builder.Services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Далі необхідно створити міграцію та оновити базу даних. Після цих дій у SQL Server Management Studio 2019, повинні відображатися усі таблиці, як зображено на Рисунок 3..

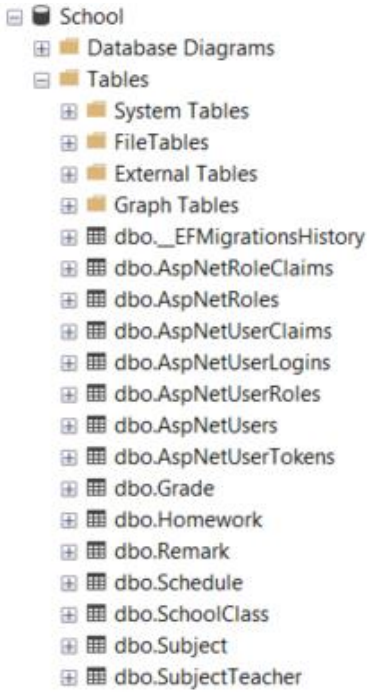

Рисунок 3.2 – Відображення бази даних в SQL Server Management Studio 2019

Наступним кроком є створення репозиторіїв. Кожен репозиторій реалізовує інтерфейс. При розробці було створено базовий репозиторій, який унаслідується усіма іншими, так як має основні методи, такі як додавання, видалення, оновлення та отримання. Слід зауважити, що даний репозиторій має загальний тип. Репозиторії мають тільки одне поле – це контекст бази

даних, який визначається в конструкторі. Нижче наведено один з методів базового репозиторію – CreateAsync:

```
public async Task<bool> CreateAsync(TEntity entity)
{
      await _db.Set<TEntity>().AddAsync(entity);
      await _db.SaveChangesAsync();
      return true;
}
```

Потім, від базового репозиторії для кожної сутності унаслідуються від базового і можуть мати власні методи. Нижче наведено репозиторій для сутності Admin:

```
    public class AdminRepository : BaseUsersRepository<Admin>,
IAdminRepository
    {
        public AdminRepository(ApplicationDbContext db) : base(db)
        {
        }
    }
```

Далі наводиться рівень бізнес логіки. Сервіси створюються аналогічним чином як і репозиторії: спочатку інтерфейс для кожної сутності, а далі сервіс, що реалізовує цей інтерфейс. У кожному сервіси є приватне поле свого репозиторію, яке визначається в конструкторі. Методи сервісів викликають методи репозиторіїв і так взаємодіють з ними. Нижче наведено приклад методу витягування з бази даних усіх студентів з одного класу:

```
public async Task<List<Student>> GetStudentsFromClass(int id)
{
      var students = await _studentRepository.SelectAsync();
      var studentsFromOneClass = students.Where(s =>
s.SchoolClass.Id == id).ToList();
      return studentsFromOneClass;
}
```

Наступним етапом є створення контролера. Для кожної сутності є свій контролер, який відповідає за взаємодію web-додатку та користувача. Він взаємодіє з сервісами, які визначаються в його конструкторі.

Контролер передає та приймає значення з представлення. Методи можуть мати свої типи, які визначаються за допомогою атрибутів. Здебільшого це HttpGet – передача даних в представлення та HttpPost – отримання даних та їх обробка.

В основному передача та отримання значень відбувається через моделі ViewModel. Дані моделі знаходяться в папці Models головного проекту. Ці класи допомагають передавати декілька типів даних у представлення та отримувати їх. У цій же папці знаходяться моделі для реєстрації та авторизації. Вони створені для роботи з формами. Кожна властивість має атрибут. Далі показано декілька властивостей з моделі UserRegister.cs:

```
[EmailAddress]
public string Email { get; set; }
Required(ErrorMessage = "Необхідно вказати пароль")]
[DataType(DataType.Password)]
public string Password { get; set; }
```

У коді вище показано, що властивість Email має такий атрибут, що при введенні форми буде проводитися перевірка на правильність введення електронної пошти. Аналогічно і до властивості Password, буде відбуватися перевірка пароля, а саме довжина, наявність спеціальних символів, цифр та літер верхнього регістру.

У представленні (View) у формі наступні рядки визначають для яких властивостей існують поля вводу:

```
<div class="form-outline flex-fill mb-0">
      <input asp-for=" Email " placeholder=" Email" class="form-
      control" />
      <span asp-validation-for=" Email " class="text-
      danger"></span>
</div>
```

Як показано вище, це здійснюється за допомогою tag helpers, таких як «asp-for» та «asp-validation-for».

Перевірка на правильність введення даних здійснюється за допомогою коду JavaScript:

```
@section Scripts{
    @{
        await
Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

Щоб модель реєстрації перетворювати в конкретну сутність необхідно використовувати AutoMapper [16]. Він перетворює вхідний об’єкт у вихідний прирівнюючи новому спільну властивості, а також ті, що визначені у спеціальному класі:

```
public class UserProfile : Profile
{
   public UserProfile()
   {
       CreateMap<UserRegister, Admin>()
           .ForMember(u => u.UserName, opt => opt.MapFrom(x =>
x.Email));
   }
}
```

Використання AutoMapper наведено нижче:

```
var admin = _mapper.Map<Admin>(userRegister);
```

Створення нових користувачів відбувається за допомогою UserManager та RoleManager [17] у методі контролера:

```
var result = await _userManager.CreateAsync(admin,
userRegister.Password);
 if (result.Succeeded) {
       role = await _roleManager.FindByNameAsync(WC.AdminRole);
      if (role != null) {
          await _userManager.AddToRoleAsync(admin, role.Name);
 }
}
```

Усі інші модулі та повний програмний код web-додатку знаходиться в ДОДАТОК В.

3.3 Використання програмного додатку

Робота web-додатку починається з авторизації (Рисунок 3.). Сторінка входу створена на основі шаблону [18]. Реєстрації немає, так як адміністратор сам створює нових користувачів.

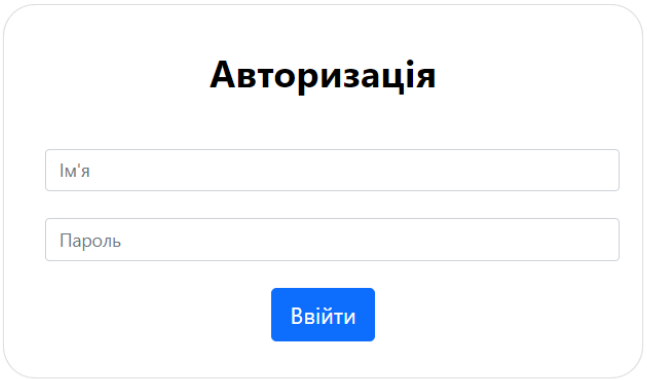

Рисунок 3.3 – Форма авторизація

На Рисунок 3. зображена сторінка адміністратора, на якій видно усіх адміністраторів, а також є можливість додати нового або видалити наявного. У меню є кнопки переходу на «Управління розкладом», «Класи» та випадаючий список з іншими типами користувачів.

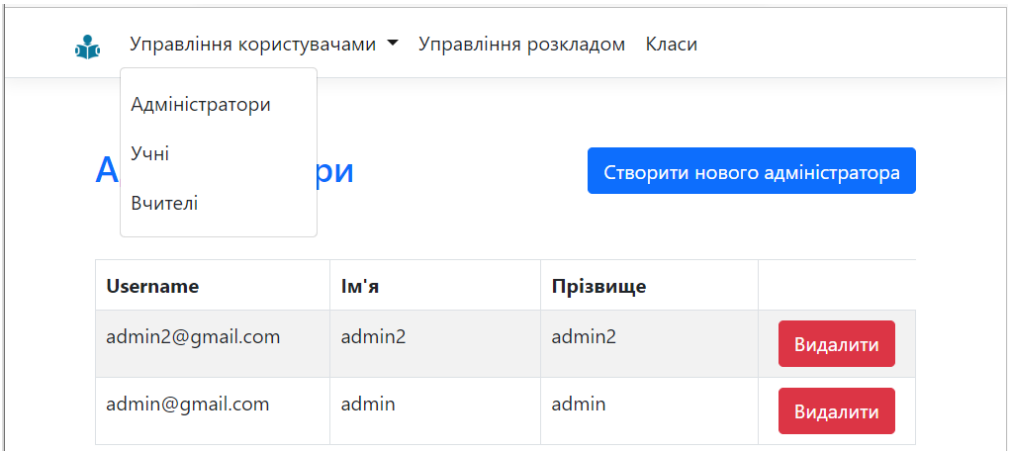

Рисунок 3.4 – Управління адміністраторами

Управління учнями (Рисунок 3.) та вчителями (Рисунок 3.) зображено нижче.

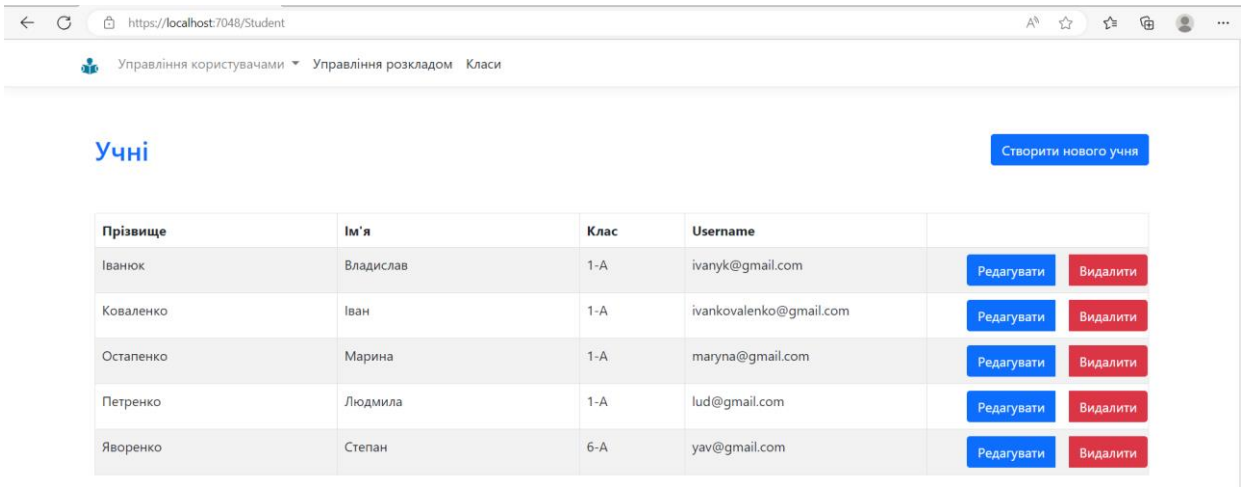

Рисунок 3.5 – Відображення усіх учнів

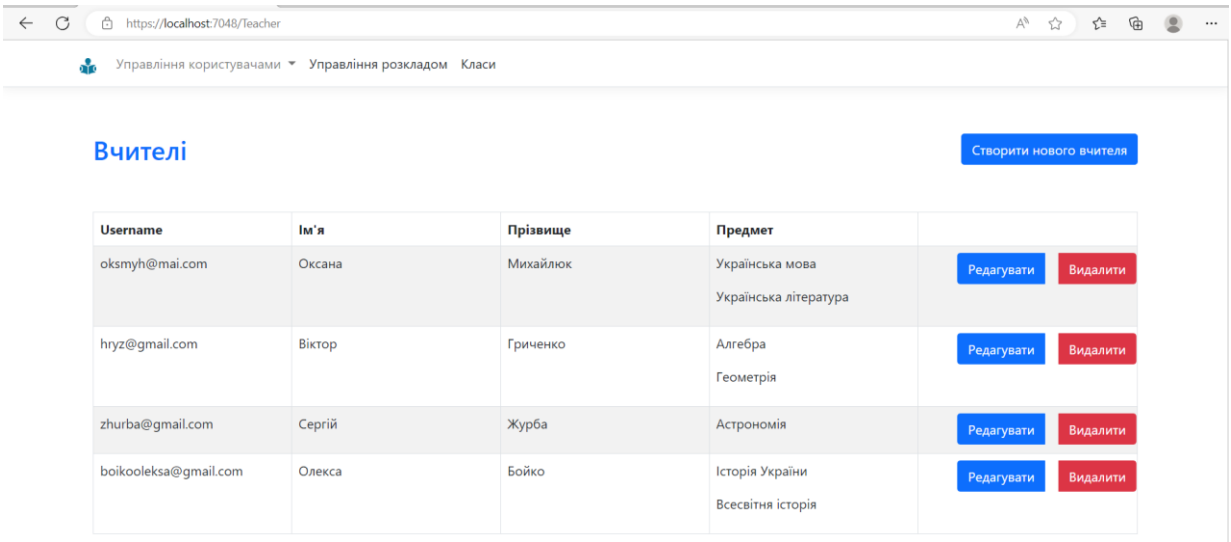

Рисунок 3.6 – Відображення усіх вчителів

Як видно з попередніх рисунків, на кожній сторінці управління користувачами є можливість створення нового користувача.

Шаблони сторінок авторизації було взято з готових рішень [1919]. Введені дані проходять валідацію та при неуспішному введені користувач буде повідомлений про це. На Рисунок 3. зображено сторінку створення нового вчителя, а на Рисунок 3. – нового адміністратора.

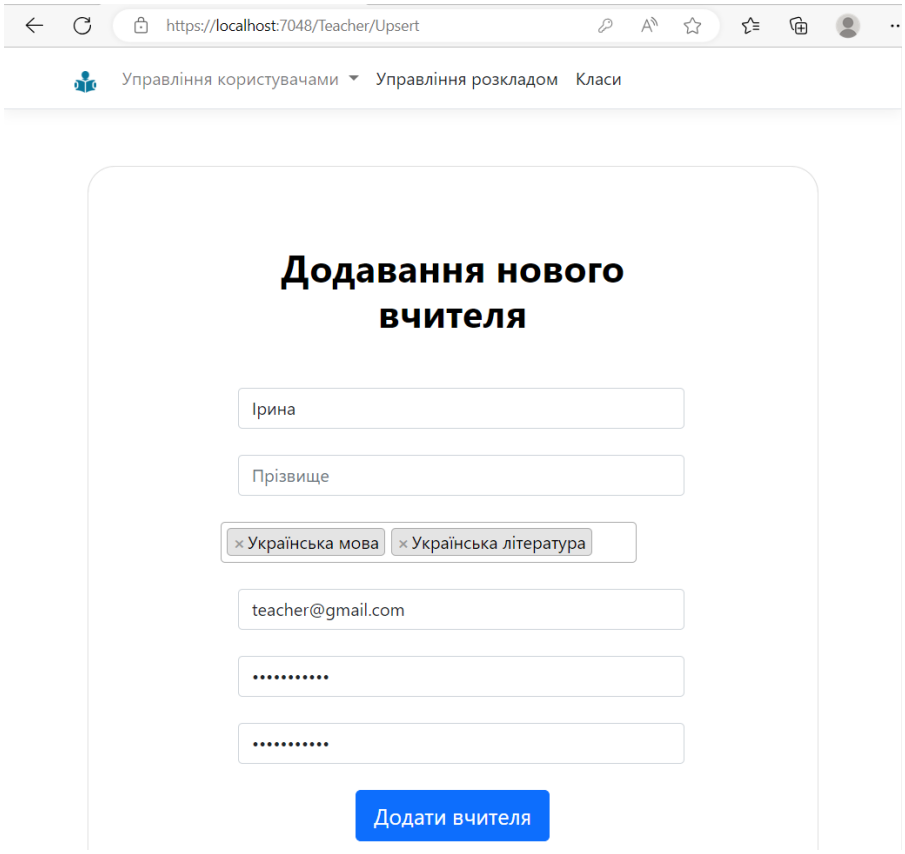

Рисунок 3.7 – Створення нового вчителя

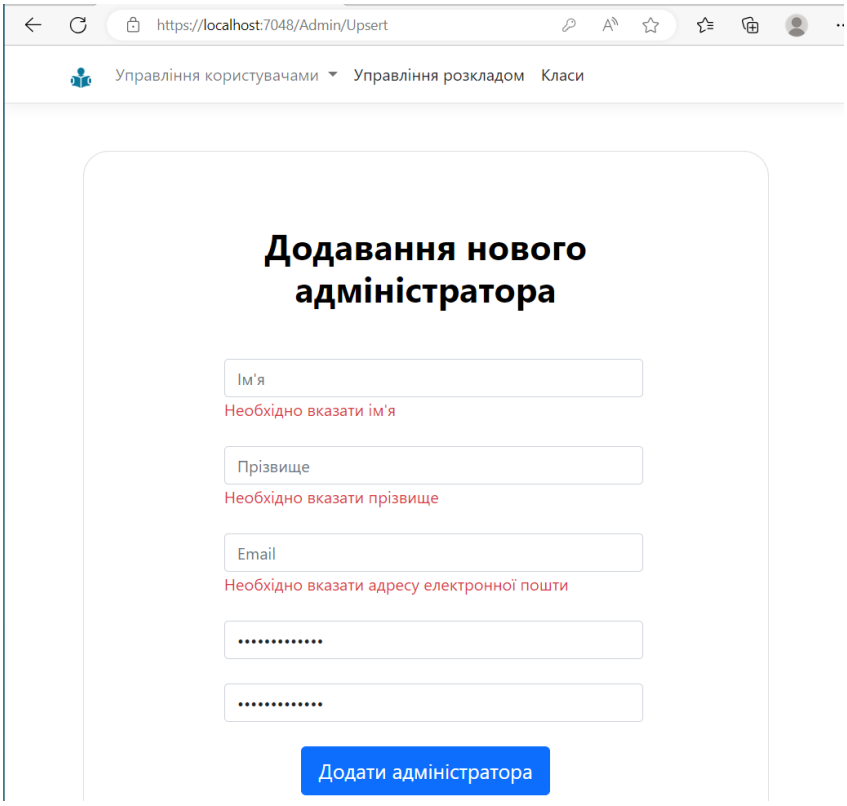

Рисунок 3.8 – Створення нового адміністратора

При створенні нового учня (Рисунок 3.), необхідно також створити аккаунт для одного з його батьків.

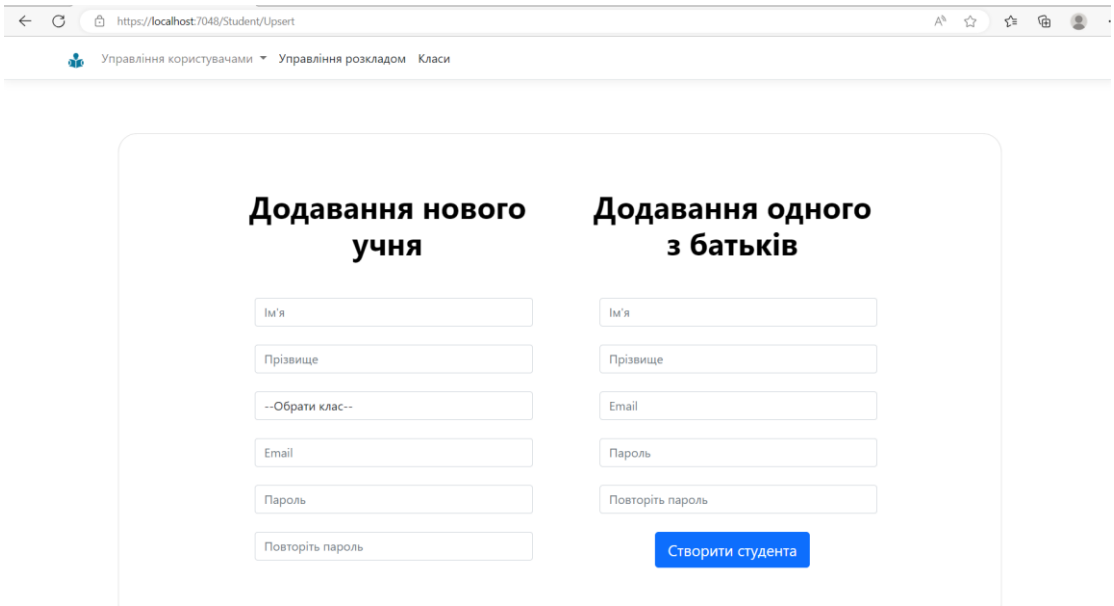

Рисунок 3.9 – Створення нового учня

При натисканні на кнопку «Класи» відкривається сторінка з усіма класами навчального закладу, як зображено на Рисунок 3..

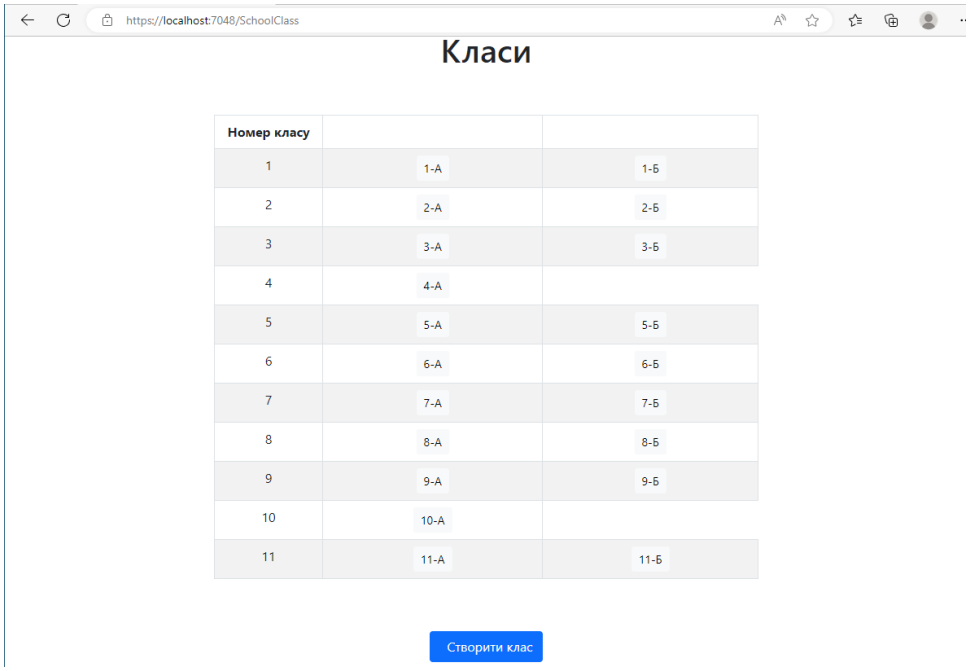

Рисунок 3.10 – Управління класами

На сторінці управління класами можна побачити усі класи, а також створити новий. При натисканні на клас, відкривається сторінка з учнями конкретного класу, аналогічно до сторінки, яка зображена на Рисунок 3..

Якщо в меню обрати пункт «Редагування розкладу», відкриється сторінка, аналогічна до сторінки «Класи», проте після натискання на клас, відкриється розклад (Рисунок 3.).

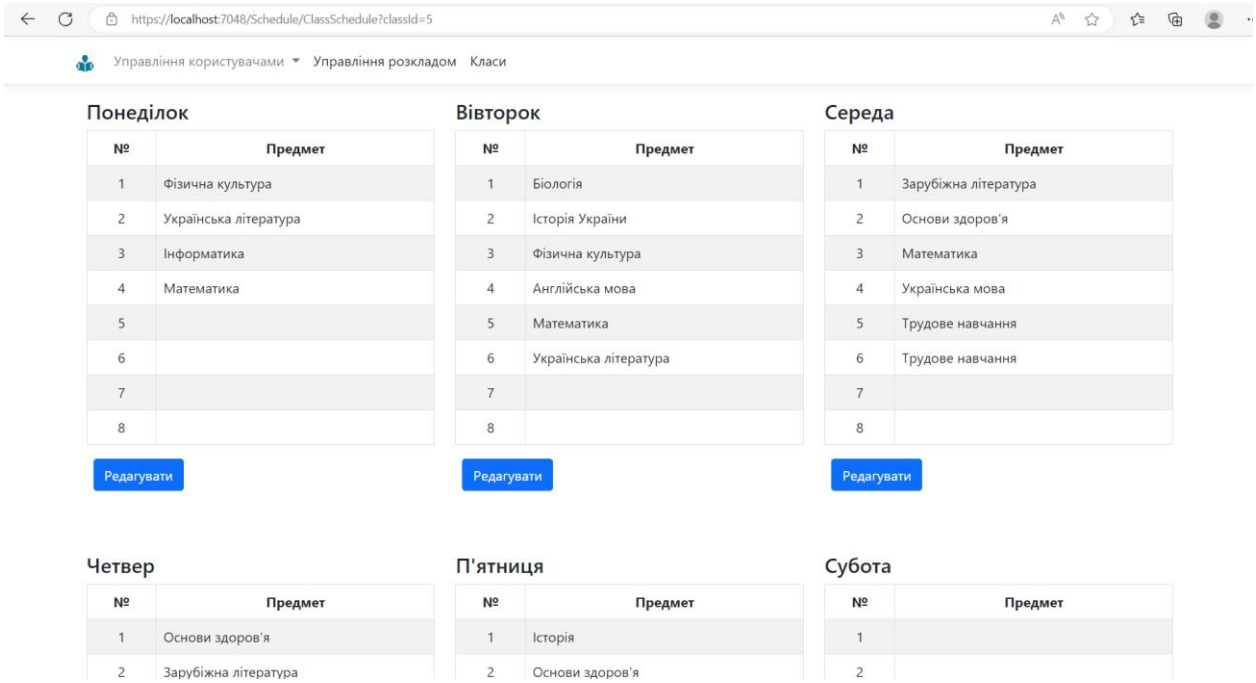

Рисунок 3.11 – Управління розкладом

Як видно з Рисунок 3., під кожним днем є кнопка «Редагувати». При натисканні на дану кнопку відкривається вікно редагування розкладу, зображено на Рисунок 3.. Дана сторінка дає змогу додавати нові уроки, змінювати їх та видаляти.

Коли в систему увійшов учитель, йому пропонується обрати клас та предмет (тільки той, який за ним закріплений). Сторінка вибору журналу зображена на Рисунок 3..

Після того, як вчитель обрав клас та предмет, його перенаправляє на сторінку журналу.

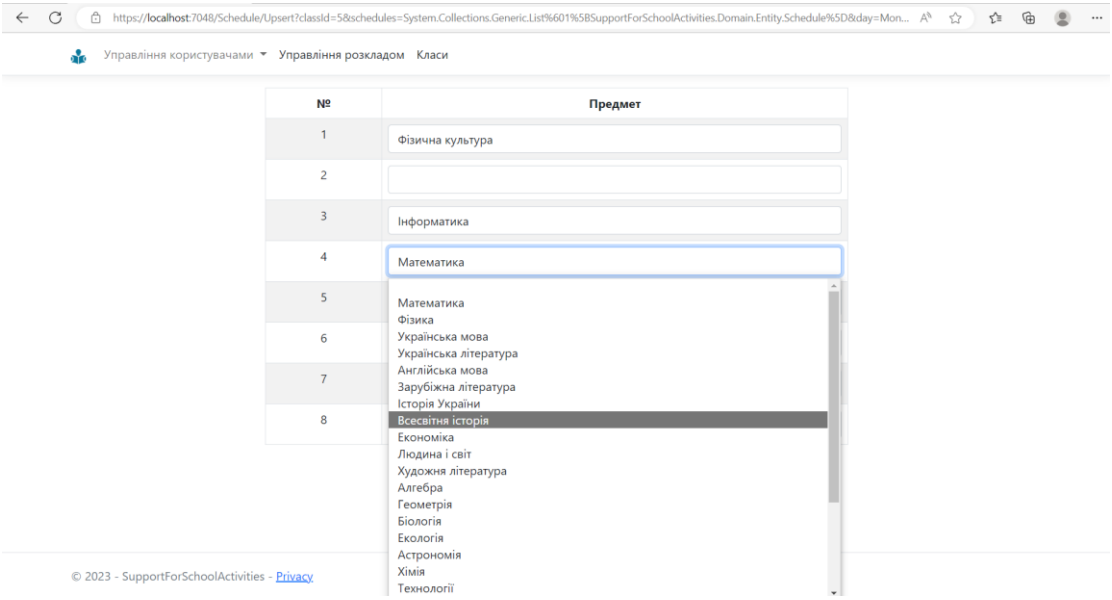

Рисунок 3.12 – Редагування розкладу

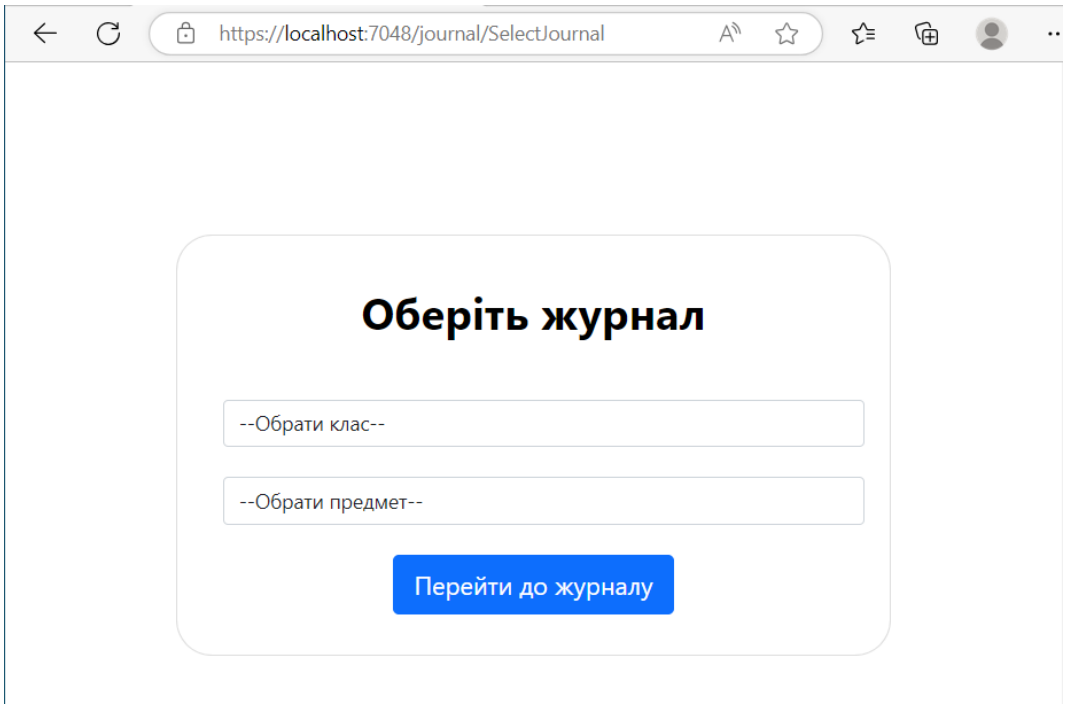

Рисунок 3.13 – Вибір журналу

На сторінці журналу вчитель може управляти оцінками, задавати домашні завдання, а також додавати відомості до учня, так як поведінка або присутність. Цей функціонал можна обрати, натиснувши на кнопку, яка знаходиться ліворуч від імені учня. Знизу під кожним уроком є змога додати домашнє завдання. Вигляд журналу зображено на Рисунок 3.

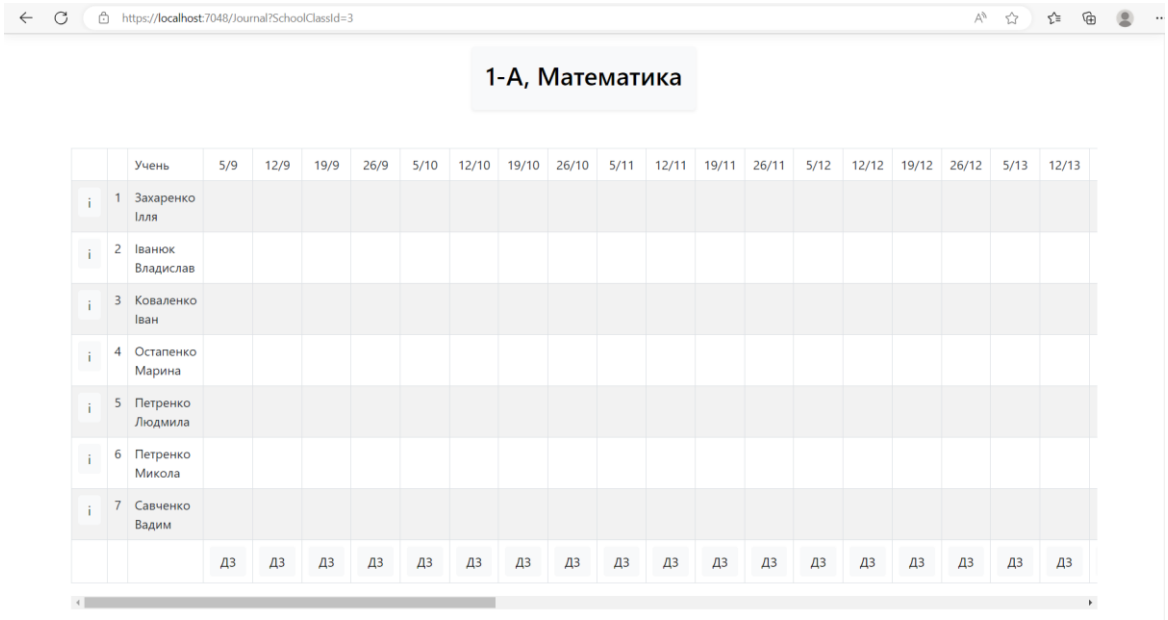

Рисунок 3.14 – Журнал

Якщо у систему увійшов учень, у нього буде можливість переглянути щоденник, який зображено на Рисунок 3..

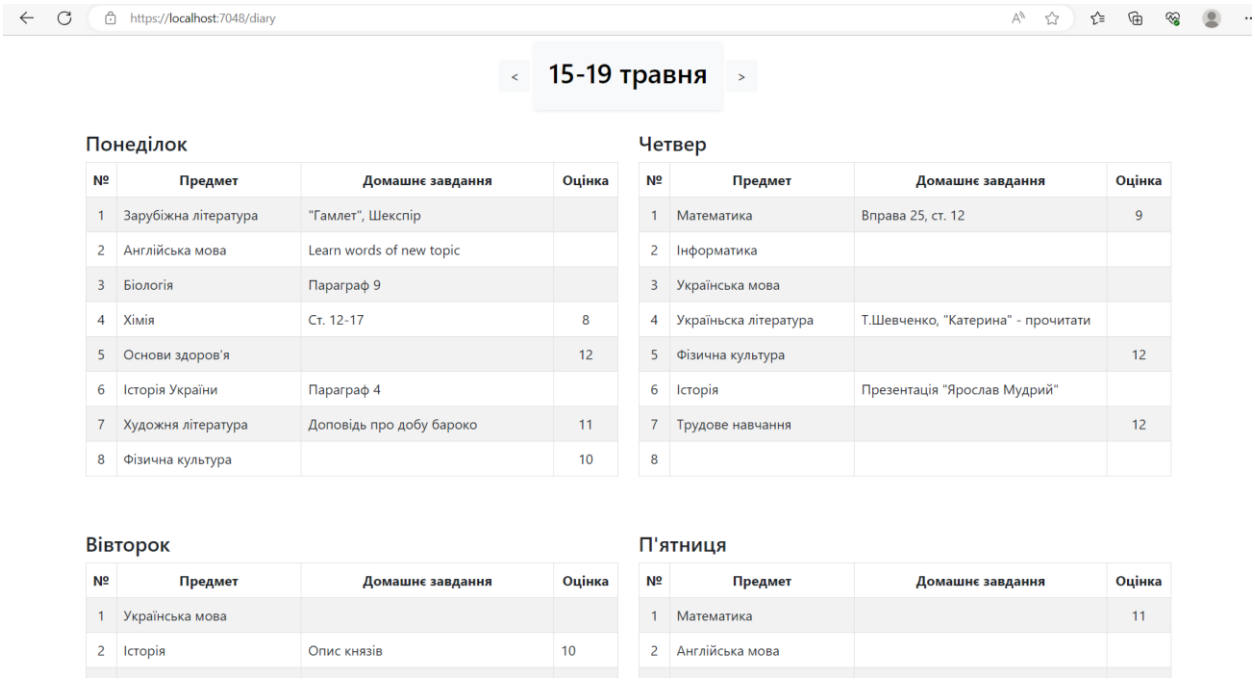

Рисунок 3.15 – Щоденник

ВИСНОВКИ

Розроблено web-додаток підтримки діяльності навчального процесу в ЗОШ. Для досягнення поставленої мети було проведено аналіз предметної області, оглянуто останні дослідження та публікації, що підтвердило актуальність даної теми та великий попит на продукти підтримки діяльності навчального процесу в ЗОШ. Також було розглянути аналоги, після чого було виявлено їх переваги та недоліки. Скориставшись цим, було сформовано вимоги, задачі та мету. Також, обрано технології для розробки.

Було проведено моделювання та проектування:

- Створено діаграму в нотації IDEF0 та її декомпозицію для розуміння процесів та взаємозв’язків між ними
- Розроблено діаграму варіантів використання для уявлення як різні користувачі можуть взаємодіяти з web-додатком
- Створено логічну модель даних для зображення сутностей у базі даних та їх взаємозв’язків

Також, показано архітектуру web-додатку, програмну реалізацію основних модулів та використання програмного продукту.

Результатом роботи є готовий web-додаток підтримки діяльності навчального процесу в ЗОШ, що допоможе спростити процес роботи усім учасникам навчального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Обраменко Д.С., Нагорний В.В. Web-додаток підтримки діяльності навчального процесу в ЗОШ. // «Інформатика, математика, автоматика»: матеріали та програма науково-технічної конференції, м. Суми, 27-28 квітня 2023 р. – Суми: Сумський державний університет (дата звернення: 16.05.2023).

2. Пасічна А. Не на папері: досвід використання електронних щоденників у світі та в Україні // Шкільне життя. URL: https://www.schoollife.org.ua/ne-na-paperi-dosvid-vykorystannya-elektronnyh-shhodennykiv-u-sviti-ta-v-ukrayini/ (дата звернення: 15.04.2023).

3. Ємець Т., Можарівська О. Довідка про результати вивчення питання щодо організованого початку 2022/2023 навчального року у закладах дошкільної, загальної середньої, позашкільної освіти. URL: https://sqe.gov.ua/wp-content/uploads/2022/10/Dovidka_pochatok_2022-2023_ZDO_ZZSO_ZPO_SQE-2022.pdf (дата звернення 15.04.2023)

4. Майже половина шкіл перейшла на електронні журнали // Освіта.ua. URL: https://osvita.ua/school/87599/ (дата звернення: 15.04.2023)

5. Юрченко О. Електронний журнал в школі, як перейти і з чого розпочати? // Освіторія. URL: https://osvitoria.media/experience/elektronnyj-zhurnal-u-shkoli-yak-perejty-i-z-chogo-rozpochaty/ (дата звернення: 15.04.2023).

6. Нові знання: вебсайт. URL: https: https://nz.ua (дата звернення: 20.04.2023)

7. E-school: вебсайт. URL: https://e-schools.info/ (дата звернення: 21.04.2023)

8. Smart-School: вебсайт. URL: https://smart-school.com.ua/ (дата звернення: 21.04.2023)

9. Методологія IDEF0 // Stud. URL: https://stud.com.ua/87184/ekonomika/metodologiya_idef0 (дата звернення: 15.05.2023)

10. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти // DOU. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 21.05.2023)

11. Логічна модель даних // Wikiwand. URL: https://www.wikiwand.com/uk/%D0%9B%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85 (дата звернення: 21.05.2023)

12. How to build and deploy a three-layer architecture application with C# // Enlab. URL: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-layer-architecture-application-with-c-sharp-net-in-practice.html (дата звернення: 22.05.2023)

13. Overview of ASP.NET Core MVC // Microsoft. URL: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/overview?view=aspnetcore-7.0 (дата звернення: 22.05.2023)

14. What is Code-First? // EntityFrameworkTutorial. URL: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx (дата звернення: 29.05.2023)

15. Spasojevic M. Introducing Identity to the ASP.NET Core Project // CodeMaze. URL: https://code-maze.com/identity-asp-net-core-project/ (дата звернення: 29.05.2023)

16. AutoMapper in C# // Dot Net Tutorials. URL: https://dotnettutorials.net/lesson/automapper-in-c-sharp/ (дата звернення: 29.05.2023)

17. Spasojevic M. User Registration ASP.NET Core Identity // CodeMaze. URL: https://code-maze.com/user-registration-aspnet-core-identity/ (дата звернення: 29.05.2023)

18. Login Form // MDBootstrap. URL: https://mdbootstrap.com/docs/standard/extended/login/? (дата звернення: 01.06.2023)

19. Registration Form // MDBootstrap. URL: https://mdbootstrap.com/docs/standard/extended/registration/ (дата звернення: 01.06.2023)

20. Boogaard K. How to write SMART goals // Atlassian. URL: https://www.atlassian.com/blog/productivity/how-to-write-smart-goals (дата звернення: 01.04.2023)

21. Freeman J. What is Work Breakdown Structure (WBS) Diagram? // Edraw. URL: https://www.edrawsoft.com/what-is-work-breakdown-structure-diagram.html (дата звернення: 02.04.2023)

22. What Is an Organizational Breakdown Structure (OBS)? // Chron. URL: https://smallbusiness.chron.com/implications-organizational-culture-project-structure-73039.html (дата звернення: 02.04.2023)

23. Gantt Chart: How to make a Gant Chart // ProjectManager. URL: https://www.projectmanager.com/guides/gantt-chart (дата звернення: 03.04.2023)

24. Hrabyna, K., & Shendryk, V. (2020). ОГЛЯД ПРОЦЕСІВ УПРАВЛІННЯ РИЗИКАМИ В ІТ-ПРОЄКТАХ У КОНТЕКСТІ СТАНДАРТІВ ПРОЄКТНОГО МЕНЕДЖМЕНТУ. Управління розвитком складних систем, (43), 26–32. https://doi.org/10.32347/2412-9933.2020.43.26-32 (дата звернення: 03.04.2023)

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку web-додатку
«Підтримки діяльності навчального процесу в ЗОШ»

ПОГОДЖЕНО:

К.т.н., доцент кафедри інформаційних технологій

_____ Нагорний В.В.

Студент групи ІТ-91

_____ Обраменко Д.С.

Суми 2023

1 Призначення й мета створення web-додатку

1.1 Призначення web-додатку

Web-додаток призначений для підтримки діяльності навчального процесу в загальноосвітній школі

1.2 Мета створення web-додатку

Головна мета проекту – це спрощення ведення журналу та щоденнику за допомогою інформаційних технологій в умовах навчання в онлайн.

1.3 Цільова аудиторія

Цільовою аудиторією даного проекту є школа, а саме адміністрація, вчителі, учні та їх батьки.

2 Вимоги до web-додатку

2.1 Вимоги до web-додатку в цілому

2.1.1 Вимоги до структури й функціонування web-додатку

Web-додаток повинен бути реалізований за допомогою web-інструментів та забезпечувати визначений набір функціональних можливостей. Кінцевий продукт даного проекту має бути представлений web-додатком, який містить якісне інформаційне наповнення.

2.1.2 Вимоги до персоналу

Основою вимогою до користувачів web-додатку є наявність ПК з web-браузером та доступ до мережі Інтернет.

2.1.3 Вимоги до збереження інформації

Уся інформація про адміністрацію, вчителів, учнів, а також навчальні відомості (розклад, оцінки, успішність, домашні завдання) повинні зберігатися у базі даних реалізованій засобами системи управління базами даних MS SQL.

2.1.4 Вимоги до розмежування доступу

Розроблюваний web-додаток повинен бути доступним тільки тим користувачам, хто має акаунт (логін і пароль надає адміністратор). Права доступу розподіляються наступним чином: адміністратор, вчитель, учень, батьки. Користувач додатку, залежно від своїх прав, матиме змогу взаємодіяти з даним web-додатком наступним чином:

- Адміністратор – перегляд усієї можливої інформації, створення класів, додавання учнів та вчителів, складання розкладу.
- Вчитель – створення уроків, додавання, видалення та редагування оцінок, додавання домашніх завдань, ведення інформації про учнів: дисципліна та присутність.
- Учень – перегляд щоденника та таблиці успішності.
- Батьки учня – перегляд оцінок та домашніх завдань їхньої дитини, а також їх дисципліни та присутності.

2.2 Структура web-додатку

2.2.1 Загальна інформація про структуру web-додатку

Web-додаток складається з 5 наступних web-сторінок:

- Форма авторизації. Єдина спільна сторінка для всіх користувачів.

- Сторінка адміністратора. Тут адміністратор зможе додавати класи, створювати нових користувачів, складати розклад, а також переглядати усю інформацію, яка зберігається в базі даних.
- Сторінка вчителя. У вчителя буде змога переглядати інформацію про учнів, створювати уроки, додавати/редагувати/видаляти оцінки та домашні завдання.
- Сторінка учня. Тут учень зможе переглядати свої оцінки, в форматі щоденника (там же будуть домашні завдання), а також матиме змогу переглянути таблицю успішності по всім предметам.
- Сторінка батьків. Той же самий функціонал, як і в учня. А також відомості про їх дитину: присутність та поведінка.

Також слід зауважити, що сторінки реєстрації не буде. При впровадженні web-додатку буде створено один аккаунт адміністратора, який матиме змогу створювати нових адміністратор, а також усіх інших типів користувачів.

2.2.2 Навігація

У шапці web-додатку для навігації буде створено меню, аби користувач міг швидко переходити по сторінкам, до яких має доступ. Для кожного типу користувачів меню буде різним, так як кожен матиме різні можливості користування web-додатку.

2.2.3 Наповнення web-додатку (контент)

Управління контентом буде здійснювати адміністратор. Він буде заносити інформацію про всіх типів користувачів, розклад, класи до бази

даних. Також, операції з базою даних буде виконувати вчитель: додавання/редагування/видалення оцінок та відомостей про учня.

2.2.4 Дизайн та структура додатку

Дизайн web-додатку повинен бути виконаний в сучасному стилі, використовуючи мінімум контенту на сторінках, при тому змістовно.

Сайт має бути «легким», приємний у користуванні для всіх видів користувачів. Усі елементи на web-сторінках повинні бути зроблені так, щоб користувач міг відразу їх замітити та виконати відповідну дію. Для кожного із видів користувачів меню різне.

Розташування елементів на сторінці авторизації схематично показано на рисунку А.1.

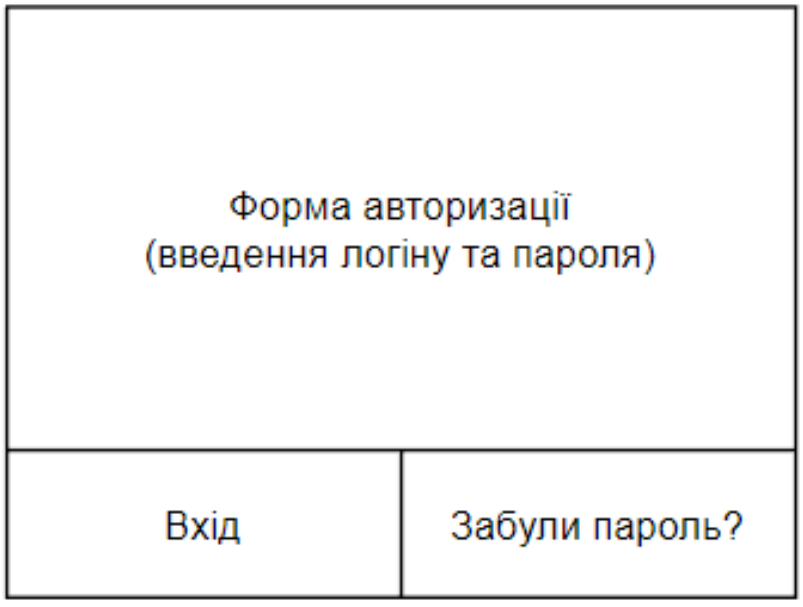

Рисунок А.1 – Схема сторінки авторизації

Розташування елементів на сторінці адміністратора зображено на рисунку А.2.

Рисунок А.2 – Схема сторінки адміністратора

Кабінет вчителя схематично зображено на рисунку А.3.

Рисунок А.3 – Схема кабінету вчителя

Розташування елементів у кабінеті учня показано на рисунку А.4.

Рисунок А.4 – Схема кабінету учня

Розташування елементів на сторінці кабінету батьків зображено на рисунку А.5.

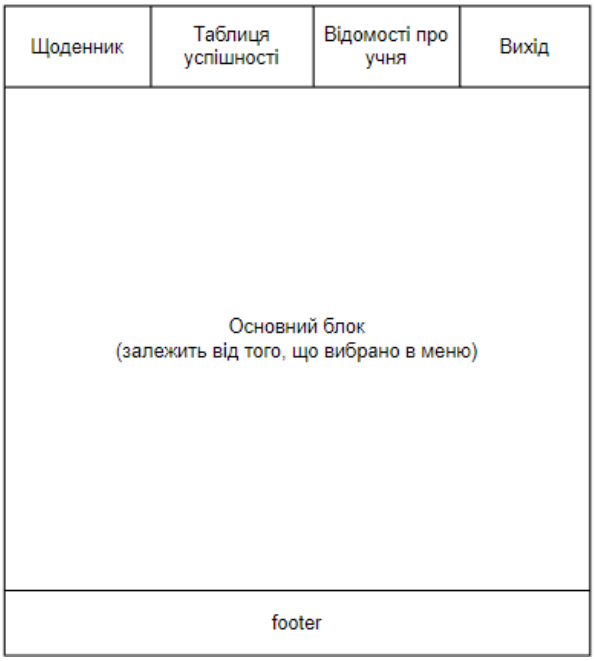

Рисунок А.5 – Схема кабінету батьків

2.2.5 Система навігації (карта web-додатку)

Карта web-додатку зображена на рисунку А.6.

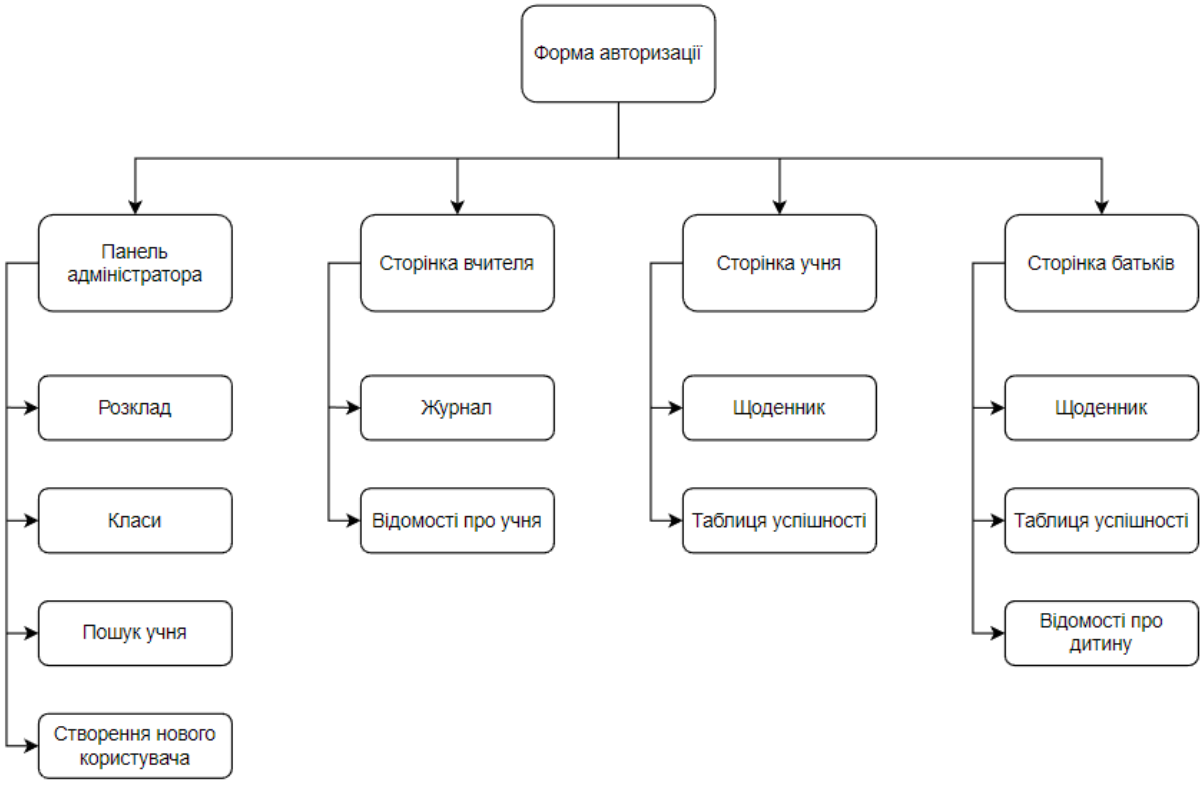

Рисунок А.6 – Карта web-додатку

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Потреби користувача, визначені на основі рішення замовника, представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN-01	Електронний журнал	Адміністратор, вчитель
UN-02	Електронний щоденник	Учень, батьки
UN-03	Таблиця успішності	Учень, батьки

Продовження таблиці А.1

ID	Потреби користувача	Джерело
UN-04	Панель адміністратора	Адміністратор
UN-05	Відомості про учня	Вчитель, батьки
UN-06	Розклад	Адміністратор, вчителі, учні, батьки

2.3.2 Функціональні вимоги

На основі потреб користувача, було визначено наступні функціональні вимоги:

- Авторизація
- Можливість у адміністратора створювати нових користувачів
- Управління усією інформацією на панелі адміністратора
- Ведення електронного журналу
- Перегляд електронного щоденника
- Перегляд таблиці успішності
- Ведення вчителями відомостей про учнів
- Перегляд батьками відомостей про їх дитину

2.3.3 Системні вимоги

Щоб задовільнити потреби користувачів та виконати функціональні вимоги було визначено системні вимоги. Вони наведені в таблиці А.2.

Таблиця А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR-01	Використання ASP.NET Core 6 MVC	M	Створення web-додатку: розробка front-end та back-end частин паралельно.

Продовження таблиці А.2

ID	Системні вимоги	Пріоритет	Опис
SR-02	Використання бази даних MS SQL	M	База даних необхідна для зберігання даних про користувачів, а також інформації навчального процесу.
SR-03	Використання фреймворку Bootstrap	S	Даний фреймворк буде використовуватися для розробки сучасного та зручного інтерфейсу.
SR-04	Використання HTTPS протоколу	S	Забезпечення безпеки, валідації введених даних, обмеження доступу для різних виді користувачів.

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Web-додаток буде реалізований за допомогою наступних інструментів:

- ASP.NET Core MVC 6
- MS SQL

2.4.2 Вимоги до лінгвістичного забезпечення

Інтерфейс web-додатку повинен бути виконаний українською мовою.

2.4.3 Вимоги до програмного забезпечення

Для забезпечення стабільної роботи web-додатку, необхідно мати ПК з стабільним інтернет з’єднанням та наступними системними вимогами:

- Операційна система: Windows, macOS або Linux
- Двоядерний процесор з частотою 1,8 ГГц або вище
- Оперативна пам’ять: 2 Гб або більше
- Будь-який сучасний веб-браузер, такий як Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari тощо. Бажано, браузер повинен бути оновлений до останньої версії.

3 Склад і зміст робіт зі створення web-додатку

Докладний опис етапів створення web-додатку наведено в таблиці А.3.

Таблиця А.3 – Етапи створення web-додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Дослідження предметної області	3 дні
2	Дослідження аналогів	1 день
3	Створення ТЗ	2 дні
4	Розробка шаблону web-додатку	4 дні
5	Задання верстки сторінок web-додатку	5 днів
6	Створення бази даних	3 дні
7	Розробка панелі адміністратора	5 днів
8	Розробка модулю для вчителів	5 днів
9	Розробка модулю для учнів	5 днів
10	Розробка модулю для батьків	2 дні
11	Тестування	3 дні

Продовження таблиці А.3

№	Склад і зміст робіт	Строк розробки (у робочих днях)
12	Обирання хостингу	1 день
13	Реліз web-додатку	1 день
	Загальна тривалість робіт	40 днів

4 Вимоги до складу й змісту робіт із введення web-додатку в експлуатацію

Першим етапом введення web-додатку в експлуатацію є тестування, тобто необхідно переконатися в правильності роботи web-додатку, а також в перевірити на відповідність вимогам користувача та функціональним вимогам. Наступним кроком є обрання хостингу, на якому буде розміщено web-додаток. Після попередніх кроків, web-додаток можна вводити в експлуатацію.

ДОДАТОК Б

Планування робіт

Інформаційні технології в наш час розвиваються швидкими темпами. Не можна навіть уявити сучасний світ без інноваційних систем, так як вони наявні в усіх галузях: від освіти до науки, від медицини до спорту. Через мережу Інтернет люди спілкуються, здійснюють онлайн купівлі, записуються до лікарів, бронюють квитки, а що найголовніше – можуть знайти будь-яку необхідну інформацію на будь-яку тему.

Не виключення з цього, це область освіти. За останні 10 років, за допомогою інформаційних технологій галузь освіти дуже змінилася. Вчителям зараз набагато легше подавати матеріал, робити його більш цікавим, швидко перевіряти знання**.** У свою чергу, учням доступно безліч навчального матеріалу, який подається різними способами: уроки на різних майданчиках, наприклад на YouTube, інтерактивні завдання з миттєвою перевіркою, і, мабуть найголовніше, відкритий доступ літератури.

Останнім часом, багато шкіл по всьому світу перейшли на онлайн навчання через всесвітню пандемію. А тепер, через війну, більшість українських школярів та студентів також продовжують навчання в онлайн форматі. Через ці причини виникає великий попит на додатки, які будуть забезпечувати навчальний процес в закладах освіти.

Деталізація мети проекту методом SMART. Метод SMART [20] дозволяє чітко сформувати мету та зробити її більш конкретною, вимірювальною, досяжною, реалістичною та обмеженою в часі. за допомогою 5 принципів. Результати деталізації методом SMART розміщені у Таблиця Б..

Таблиця Б.1 – деталізація мети проекту методом SMART

Критерій	**Опис**
Specific	Спрощення ведення журналу для вчителів, а також щоденника для учнів, щоб вони могли швидко переглядати свої оцінки та домашні завдання.
Measurable	Використання електронного журналу та щоденника збільшиться на 3% в Україні.
Achievable	Мета досяжна, так як є ТЗ, розробник має досвід розробки web-додатків необхідною технологією, наявне програмне середовища для розробки web-додатку. Також є потенційні клієнти, існує попит, так як в Україні в більшості шкіл онлайн-навчання.
Relevant	Для спрощення ведення журналу для вчителів. Для можливості учням переглядати свої оцінки та домашні завдання на одній сторінці web-додатку.
Time-framed	Існує конкретний термін – до кінця весни 2023 року (31.05.2023)

Планування змісту робіт. WBS [21] (Work Breakdown Structure) діаграма, яку зображено на Рисунок Б. – це необхідний елемент для успішного проекту. Кожен менеджер проекту використовує дану діаграму для планування, виконання, моніторингу та контролю процесів. Усі елементи згруповані та ієрархічно вибудовані. На верхівці знаходиться сам продукт – назва проекту. На другому рівні ієрархії знаходяться глобальні задачі. Далі йде розбивка даних задач на елементарні роботи, які повинні мати один конкретний результат.

Планування структури виконавців. Наступним інструментом управління проектами є діаграма OBS [22] (Organization Breakdown Structure), Рисунок Б.. Вона використовується для візуального представлення робіт та їх

розбиття на виконавців. Даний інструмент дозволяє визначити ролі та відповідальність кожного учасника проекту.

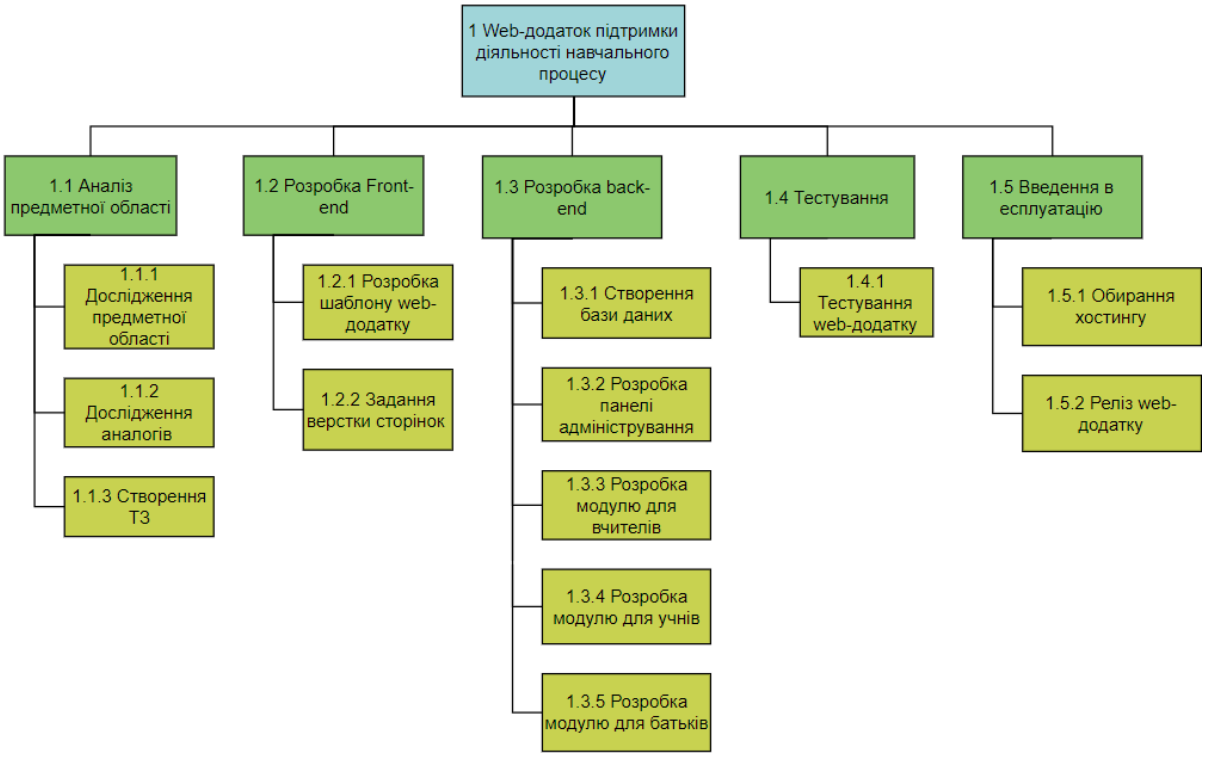

Рисунок Б.1 – WBS-структура робіт проекту

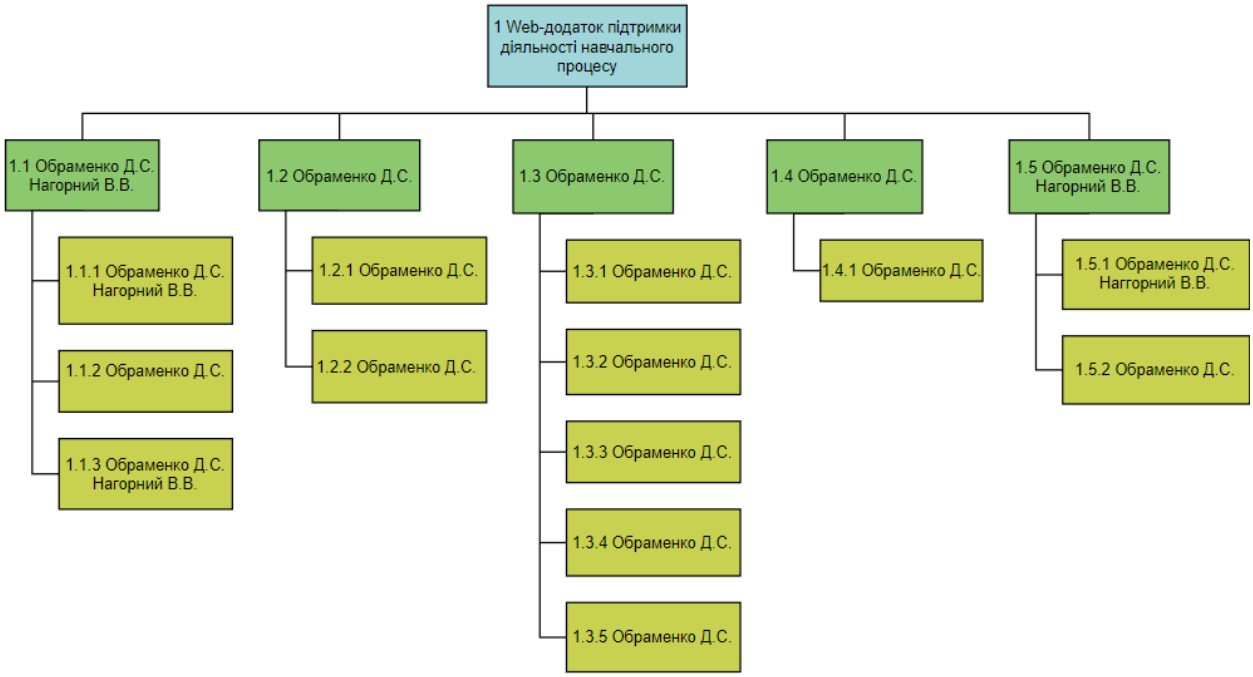

Рисунок Б.2 – OBS-структура робіт проекту

Список виконавців проекту знаходиться в Таблиця Б.2.

Таблиця Б.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Обраменко Д.С.	Виконує розробку front-end та back-end частини web-додатку
Проектувальник	Обраменко Д.С.	Розробляє структуру web-додатку та виконує проектування бази даних
Тестувальник	Обраменко Д.С.	Виконує тестування web-додатку
Керівник проекту	Нагорний В.В.	Формує завдання на розробку проекту
Менеджер проекту	Обраменко Д.С.	Виконує збір та аналіз даних. Відповідає за терміни виконання, розподіл завдань та ресурсів.

Діаграма Ганта [23]. Наступним етапом при плануванні робіт є побудова діаграми Ганта, яка зображена на Рисунок Б.. Даний інструмент дозволяє чітко розробити календарний план та візуалізувати роботи проекту по критерії часу. Дана діаграма дозволяє учасникам проекту зрозуміти хронологію виконання завдань, їх залежності та загальний графік проекту. Кожна смуга на діаграмі Ганта представляє окреме завдання, а її довжина відповідає його тривалості.

Аналіз ризиків [24]. Важливою складовою успішного проекту є попередня ідентифікація потенційних ризиків. Необхідно оцінити їх вплив та розробити стратегії для їх управління. Аналіз ризиків допомагає передбачити ймовірні проблеми, які можуть вплинути на хід виконання проекту. Даний процес включає систематичне виявлення, оцінку та управління ризиками з метою зниження негативного впливу на проект.

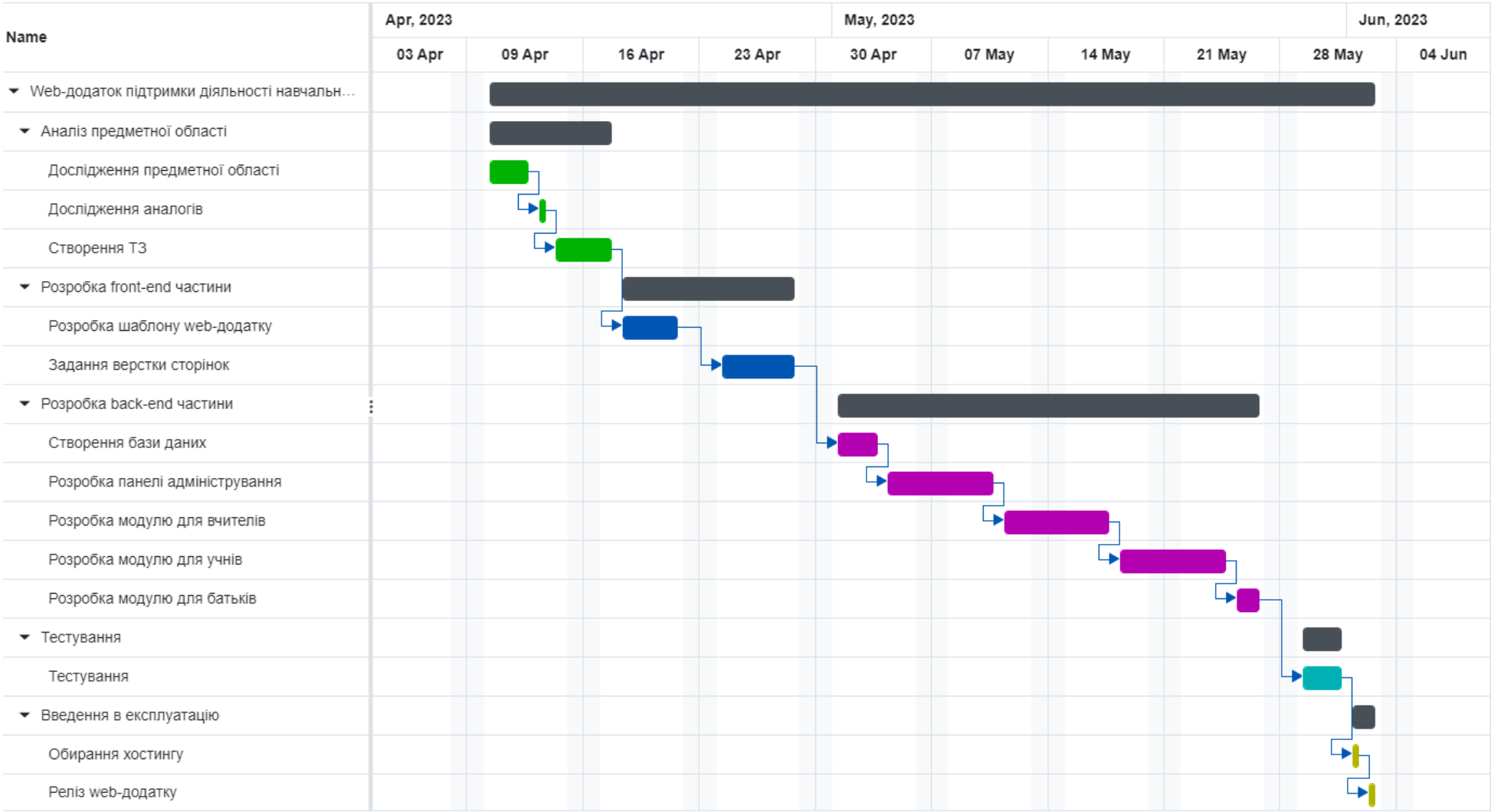

Рисунок Б.3 – Діаграма Ганта

У Таблиця Б.3 представлено шкалу для класифікації ризиків за величиною впливу на проект та ймовірністю виникнення.

Таблиця Б.3 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	**Ймовірність виникнення**	**Вплив ризику**	**Тип ризику**
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

У Таблиця Б.4 наведено ймовірність виникнення ризиків та їх вплив.

Таблиця Б.4 – Матриця ймовірності

Ймовірність ризику	Вплив загрози				
	Дуже малий	Малий	Середній	Великий	Дуже великий
	0,05	0,1	0,2	0,4	0,8
0,9	0,045	0,09	0,18 **R1**	0,36	0,72
0,7	0,035	0,07	0,14 **R8**	0,28	0,56
0,5	0,025	0,05	0,1 **R3**	0,2	0,4
0,3	0,015	0,03 **R4**	0,06 **R2, R6**	0,12	0,24 **R9**
0,1	0,005	0,01	0,02	0,04	0,08 **R5, R7**

Нижче наведено класифікацію ризиків за рівнем, відповідно до отриманого значення індексу.

Зелений ($0{,}005 \leq R \leq 0{,}05$) – прийнятні

Жовтий ($0{,}05 < R \leq 0{,}14$) – виправдані

Червоний ($0{,}14 < R \leq 0{,}72$) – недопустимий

Таблиця Б.5 – Ризики та стратегії реагування

ID ризику	Статус	Опис	Ймовірність виникнення	Вплив	Ранг	План А	Тип стратегії реагування	План Б
R1	Відкритий	Загроза бойових дій	Низька	Високий	7	Переміщення розробника подалі від лінії фронту чи кордону з країною-агресором	Ухилення	Збільшити дедлайни
R2	Відкритий	Періодичне відключення електроенергії	Низька	Середній	4	Ходити розробляти програмний продукт в СумДУ, так як там завжди наявна електроенергія	Ухилення	Знайти місце для коворкінгу
R3	Відкритий	Відсутність інтернету	Низька	Середній	6	Забезпечити розробника мобільним інтернетом	Ухилення	Ходити в навчальні центри, у яких є інтернетом, наприклад - СумДУ

Продовження таблиці Б.5

ID ризику	Статус	Опис	Ймовірність виникнення	Вплив	Ранг	План А	Тип стратегії реагування	План Б
R4	Відкритий	Поломка ПК	Низька	Низький	2	Скористатися публічним ПК в університеті, бібліотеці або інших спеціальних місцях	Ухилення	Віддати ПК в сервісний центр з можливістю якнайшвидшого ремонту
R5	Відкритий	Недосвідченість розробника (в роботі з даною технологією)	Низька	Середній	4	Підвищити кваліфікацію розробника: перегляд онлайн-курсів, читання документації	Зменшення	Купити інтенсивний курс у професіонала, найняти ментора
R6	Відкритий	Хвороба розробника	Низька	Низький	3	Змістити сроки виконання задач	Ухилення	Збільшити дедлайни
R7	Відкритий	Часте внесення змін у ТЗ	Низька	Середній	4	Проаналізувати вимоги та виділити усі необхідні задачі	Зменшення	Розібратися в причині зміни ТЗ, вирішити чи необхідно переробляти продукт

Продовження таблиці Б.5

ID ризику	Статус	Опис	Ймовірність виникнення	Вплив	Ранг	План А	Тип стратегії реагування	План Б
R8	Відкритий	Нестача часу	Низька	Середній	6	Дотримуватися термінів робіт. Визначити найактуальніші потреби, які слід реалізувати, а також звернути увагу на новий розподіл часу	Зменшення	Розробиити новий календарний план, усунуваши деякий функціонал продукту після повторного аналізу
R9	Відкритий	Вибір неефективної технології розробки	Низька	Високий	8	Проаналізувати інші технології. Обрати той інструмент, який підходить для розробки даного продукту, а також той який гарно знайомий розробнику	Зменшення	Виділити час та ресурси для ознайомлення з новою технологією

ДОДАТОК В

Лістинг програмного коду основних модулів web-додатку

User.cs:

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class User : IdentityUser
    {
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
    }
}
```

Admin.cs:

```
namespace SupportForSchoolActivities.Domain.Entity
{
    public class Admin : User
    {
    }
}
```

Student.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class Student : User
    {
        public virtual ICollection<Grade> Grades { get; set; }
        public virtual ICollection<Remark> Remarks { get; set; }
        [Required]
        public virtual Parent Parent { get; set; }
        public virtual SchoolClass SchoolClass { get; set; }
    }
}
```

Teacher.cs:

```
namespace SupportForSchoolActivities.Domain.Entity
{
    public class Teacher : User
    {
        public ICollection<Subject> Subjects { get; set; } = new List<Subject>();
```

```
    }
}
```

Parent.cs:

```
namespace SupportForSchoolActivities.Domain.Entity
{
    public class Parent : User
    {
        public virtual ICollection<Student> Students { get; set; } = new List<Student>();
    }
}
```

SchoolClass.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class SchoolClass
    {
        public int Id { get; set; }
        public string Name { get; set; }
        [Required]
        [Range(1, 11)]
        public int ClassNumber { get; set; }
        public virtual ICollection<Student> Students { get; set; } = new List<Student>();
        public virtual ICollection<Homework> Homeworks { get; set; }
        public virtual ICollection<Schedule> Schedules { get; set; }
    }
}
```

Subject.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class Subject
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        public virtual ICollection<Teacher> Teachers { get; set; }
        public virtual ICollection<Grade> Grades { get; set; }
        public virtual ICollection<Homework> Homeworks { get; set; }
        public virtual ICollection<Schedule> Schedules { get; set; }
    }
}
```

Schedule.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class Schedule
    {
        public int Id { get; set; }
```

```
        public DayOfWeek DayOfWeek { get; set; }
        [Range(1, 8)]
        public int LessonNumber { get; set; }
        public virtual Subject Subject { get; set; }
        public virtual SchoolClass SchoolClass { get; set; }
    }
}
```

Remark.cs:

```
namespace SupportForSchoolActivities.Domain.Entity
{
    public class Remark
    {
        public int Id { get; set; }
        public string Description { get; set; }
        public DateTime Date { get; set; }
        public virtual Student Student { get; set; }
    }
}
```

Homework.cs:

```
namespace SupportForSchoolActivities.Domain.Entity
{
    public class Homework
    {
        public int Id { get; set; }
        public string Description { get; set; }
        public DateTime Deadline { get; set; }
        public virtual SchoolClass SchoolClass { get; set; }
        public virtual Subject Subject { get; set; }
    }
}
```

Grade.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Domain.Entity
{
    public class Grade
    {
        public int Id { get; set; }
        [Range(1, 12)]
        public int Mark { get; set; }
        public DateTime Date { get; set; }
        public virtual Student Student { get; set; }
        public virtual Subject Subject { get; set; }
    }
}
```

RoleConfiguration.cs:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore;

namespace SupportForSchoolActivities.DAL
{
```

```
    public class RoleConfiguration : IEntityTypeConfiguration<IdentityRole>
    {
        public void Configure(EntityTypeBuilder<IdentityRole> builder)
        {
            builder.HasData(
                new IdentityRole
                {
                    Name = "Admin",
                    NormalizedName = "ADMIN",
                },
                new IdentityRole
                {
                    Name = "Teacher",
                    NormalizedName = "TEACHER",
                },
                new IdentityRole
                {
                    Name = "Student",
                    NormalizedName = "STUDENT",
                },
                new IdentityRole
                {
                    Name = "Parent",
                    NormalizedName = "PARENT",
                });
        }
    }
}
```

ApplicationDbContext.cs:

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL
{
    public class ApplicationDbContext : IdentityDbContext<User>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options)
        {
        }
        public DbSet<Admin> Admin { get; set; }
        public DbSet<Student> Student { get; set; }
        public DbSet<Teacher> Teacher { get; set; }
        public DbSet<Parent> Parent { get; set; }
        public DbSet<SchoolClass> SchoolClass { get; set; }
        public DbSet<Subject> Subject { get; set; }
        public DbSet<Grade> Grade { get; set; }
        public DbSet<Homework> Homework { get; set; }
        public DbSet<Schedule> Schedule { get; set; }
        public DbSet<Remark> Remark { get; set; }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Student>(entity =>
            {
                entity.HasOne<Parent>(s => s.Parent)
                .WithMany(p => p.Students)
                .OnDelete(DeleteBehavior.NoAction);
                entity.Navigation(e => e.Parent).IsRequired();
            });
```

```
            modelBuilder.Entity<Student>(entity =>
            {
                entity.HasOne<SchoolClass>(s => s.SchoolClass)
                .WithMany(c => c.Students)
                .OnDelete(DeleteBehavior.NoAction);
            });
            modelBuilder.ApplyConfiguration(new RoleConfiguration());
        }
    }
}
```

IBaseRepository.cs:

```
namespace SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces
{
    public interface IBaseRepository<T>
    {
        Task<bool> CreateAsync(T entity);
        Task<List<T>> SelectAsync();
        Task<T> UpdateAsync(T entity);
        Task<bool> DeleteAsync(T entity);
    }
}
```

IBaseEntitiesRepository.cs:

```
namespace SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces
{
    public interface IBaseEntitiesRepository<T> : IBaseRepository<T>
    {
        Task<T> GetAsync(int id);
        Task<bool> DeleteByIdAsync(int id);
    }
}
```

IBaseUserRepository.cs:

```
namespace SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces
{
    public interface IBaseUsersRepository<T> : IBaseRepository<T>
    {
        Task<T> GetAsync(string id);
        Task<bool> DeleteByIdAsync(string id);
    }
}
```

IAdminRepository.cs:

```
using SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Interfaces
{
    public interface IAdminRepository : IBaseUsersRepository<Admin>
    {
    }
}
```

IStudentRepository.cs:

```
using SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Interfaces
{
    public interface IStudentRepository : IBaseUsersRepository<Student>
    {
        Task<bool> CreateAsync(Student student, Parent parent);
    }
}
```

ITeacherRepository.cs:

```
using SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Interfaces
{
    public interface ITeacherRepository : IBaseUsersRepository<Teacher>
    {
    }
}
```

BaseRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces;

namespace SupportForSchoolActivities.DAL.Repository.BaseRepositories
{
    public abstract class BaseRepository<TEntity> : IBaseRepository<TEntity>
        where TEntity : class
    {
        protected readonly ApplicationDbContext _db;

        protected BaseRepository(ApplicationDbContext db)
        {
            _db = db;
        }

        public async Task<bool> CreateAsync(TEntity entity)
        {
            await _db.Set<TEntity>().AddAsync(entity);
            await _db.SaveChangesAsync();
            return true;
        }

        public async Task<bool> DeleteAsync(TEntity entity)
        {
            if (entity == null)
                return false;
            _db.Set<TEntity>().Remove(entity);
            await _db.SaveChangesAsync();
            return true;
        }

        public async Task<List<TEntity>> SelectAsync()
        {
            return await _db.Set<TEntity>().ToListAsync();
        }

        public async Task<TEntity> UpdateAsync(TEntity entity)
        {
```

```
            _db.Set<TEntity>().Update(entity);
            await _db.SaveChangesAsync();
            return entity;
        }
    }
}
```

BaseEntitiesRepository.cs:

```
using SupportForSchoolActivities.DAL.Interfaces.BaseInterfaces;

namespace SupportForSchoolActivities.DAL.Repository.BaseRepositories
{
    public abstract class BaseEntitiesRepository<TEntity> : BaseRepository<TEntity>,
IBaseEntitiesRepository<TEntity>
        where TEntity : class
    {
        protected BaseEntitiesRepository(ApplicationDbContext db) : base(db)
        {
        }

        public async Task<bool> DeleteByIdAsync(int id)
        {
            var entity = await _db.Set<TEntity>().FindAsync(id);
            if (entity != null)
            {
                _db.Set<TEntity>().Remove(entity);
                await _db.SaveChangesAsync();
                return true;
            }
            return false;
        }

        public async Task<TEntity> GetAsync(int id)
        {
            return await _db.Set<TEntity>().FindAsync(id);
        }
    }
}
```

BaseUserRepository.cs:

```
using Microsoft.EntityFrameworkCore;

namespace SupportForSchoolActivities.DAL.Repository.BaseRepositories
{
    public abstract class BaseUsersRepository<TEntity> : BaseRepository<TEntity>,
IBaseUsersRepository<TEntity>
        where TEntity : class
    {
        protected BaseUsersRepository(ApplicationDbContext db) : base(db)
        {
        }
        public async Task<TEntity> GetAsync(string id)
        {
            return await _db.Set<TEntity>().FindAsync(id);

        }
        public async Task<bool> DeleteByIdAsync(string id)
        {
            var entity = await _db.Set<TEntity>().FindAsync(id);
            if (entity != null)
```

```
            {
                _db.Set<TEntity>().Remove(entity);
                await _db.SaveChangesAsync();
                return true;
            }
            return false;
        }
    }
}
```

AdminRepository.cs:

```
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.DAL.Repository.BaseRepositories;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Repository
{
    public class AdminRepository : BaseUsersRepository<Admin>, IAdminRepository
    {
        public AdminRepository(ApplicationDbContext db) : base(db)
        {
        }
    }
}
```

StudentRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.DAL.Repository.BaseRepositories;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Repository
{
    public class StudentRepository : BaseUsersRepository<Student>, IStudentRepository
    {
        public StudentRepository(ApplicationDbContext db) : base(db)
        {
        }

        public async Task<bool> CreateAsync(Student student, Parent parent)
        {
            try
            {
                student.Parent = parent;
                await _db.Student.AddAsync(student);
                return true;
            }
            catch
            {
                return false;
            }

        }

        public async Task<List<Student>> SelectAsync()
        {
            return await _db.Student
                .AsNoTracking()
                .Include(s => s.Parent)
                .Include(s => s.SchoolClass)
```

```
                .ToListAsync();
        }
    }
}
```

TeacherRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.DAL.Repository.BaseRepositories;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.DAL.Repository
{
    public class TeacherRepository : BaseUsersRepository<Teacher>, ITeacherRepository
    {
        public TeacherRepository(ApplicationDbContext db) : base(db)
        {
        }

        public async Task<List<Teacher>> SelectAsync()
        {
            return await _db.Teacher
                .AsNoTracking()
                .Include(t => t.Subjects)
                .ToListAsync();
        }
    }
}
```

IAdminService.cs:

```
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Service.Interfaces
{
    public interface IAdminService
    {
        Task<List<Admin>> GetListAdmins();
        Task<Admin> GetAdmin(string id);
        Task<bool> CreateAdmin(Admin admin);
        Task<bool> UpdateAdmin(string id, Admin admin);
        Task<bool> DeleteAdmin(string id);
    }
}
```

IStudentService.cs:

```
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Service.Interfaces
{
    public interface IStudentService
    {
        Task<List<Student>> GetAllStudents();
        Task<Student> GetStudent(string id);
        Task<bool> CreateStudent(Student student, Parent parent);
        Task<bool> CreateStudent(Student student);
        Task<bool> UpdateStudent(string id, Student student);
```

```
        Task<bool> DeleteStudent(string id);
        Task<List<Student>> GetStudentsFromOneClass(int id);
    }
}
```

ITeacherService.cs:

```
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Service.Interfaces
{
    public interface ITeacherService
    {
        Task<List<Teacher>> GetAllTeachers();
        Task<Teacher> GetTeacher(string id);
        Task<bool> CreateTeacher(Teacher teacher);
        Task<bool> UpdateTeacher(string id, Teacher teacher);
        Task<bool> DeleteTeacher(string id);
        Task<List<Subject>> GetAllSubjectOfThisTeacher(string id);
    }
}
```

AdminService.cs:

```
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Service.Interfaces;

namespace SupportForSchoolActivities.Service
{
    public class AdminService : IAdminService
    {
        private readonly IAdminRepository _adminRepository;

        public AdminService(IAdminRepository adminRepository)
        {
            _adminRepository = adminRepository;
        }

        public async Task<bool> CreateAdmin(Admin admin)
        {
            try
            {
                await _adminRepository.CreateAsync(admin);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<bool> DeleteAdmin(string id)
        {
            var admin = await _adminRepository.GetAsync(id);
            if(admin == null)
            {
                return false;
            }
            try
            {
                await _adminRepository.DeleteAsync(admin);
```

```
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<Admin> GetAdmin(string id)
        {
            var admin = await _adminRepository.GetAsync(id);
            return admin;
        }

        public async Task<List<Admin>> GetListAdmins()
        {
            var admins = await _adminRepository.SelectAsync();
            return admins;
        }

        public async Task<bool> UpdateAdmin(string id, Admin admin)
        {
            if(admin == null || id == null)
            {
                return false;
            }
            try
            {
                var oldAdmin = await _adminRepository.GetAsync(id);
                oldAdmin.FirstName = admin.FirstName;
                oldAdmin.LastName = admin.LastName;
                oldAdmin.Email = admin.Email;
                oldAdmin.PhoneNumber= admin.PhoneNumber;
                oldAdmin.UserName = admin.UserName;
                await _adminRepository.UpdateAsync(oldAdmin);
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

StudentService.cs:

```
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Service.Interfaces;

namespace SupportForSchoolActivities.Service
{
    public class StudentService : IStudentService
    {
        private readonly IStudentRepository _studentRepository;

        public StudentService(IStudentRepository studentRepository)
        {
            _studentRepository = studentRepository;
        }

        public async Task<bool> CreateStudent(Student student, Parent parent)
```

```
        {
            try
            {
                await _studentRepository.CreateAsync(student, parent);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<bool> CreateStudent(Student student)
        {
            try
            {
                await _studentRepository.CreateAsync(student);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<bool> DeleteStudent(string id)
        {
            var student = await _studentRepository.GetAsync(id);
            if(student == null)
            {
                return false;
            }
            try
            {
                await _studentRepository.DeleteAsync(student);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<List<Student>> GetAllStudents()
        {
            var students = await _studentRepository.SelectAsync();
            return students;
        }

        public async Task<Student> GetStudent(string id)
        {
            var student = await _studentRepository.GetAsync(id);
            return student;
        }

        public async Task<List<Student>> GetStudentsFromOneClass(int id)
        {
            var students = await _studentRepository.SelectAsync();
            var studentsFromOneClass = students.Where(s => s.SchoolClass.Id ==
id).ToList();
            return studentsFromOneClass;
        }

        public async Task<bool> UpdateStudent(string id, Student student)
```

```
        {
            if (student == null || id == null)
            {
                return false;
            }
            try
            {
                var oldStudent = await _studentRepository.GetAsync(id);
                oldStudent.FirstName = student.FirstName;
                oldStudent.LastName = student.LastName;
                oldStudent.Email = student.Email;
                oldStudent.PhoneNumber = student.PhoneNumber;
                oldStudent.UserName = student.UserName;
                await _studentRepository.UpdateAsync(oldStudent);
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

TeacherService.cs:

```
using SupportForSchoolActivities.DAL.Interfaces;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Service.Interfaces;

namespace SupportForSchoolActivities.Service
{
    public class TeacherService : ITeacherService
    {
        private readonly ITeacherRepository _teacherRepository;

        public TeacherService(ITeacherRepository teacherRepository)
        {
            _teacherRepository = teacherRepository;
        }

        public async Task<bool> CreateTeacher(Teacher teacher)
        {
            try
            {
                await _teacherRepository.CreateAsync(teacher);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<bool> DeleteTeacher(string id)
        {
            var teacher = await _teacherRepository.GetAsync(id);
            if (teacher == null)
            {
                return false;
            }
            try
            {
```

```
                await _teacherRepository.DeleteAsync(teacher);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public async Task<List<Subject>> GetAllSubjectOfThisTeacher(string id)
        {
            var teacher = await _teacherRepository.GetAsync(id);
            return teacher.Subjects.ToList();
        }

        public async Task<List<Teacher>> GetAllTeachers()
        {
            return await _teacherRepository.SelectAsync();

        }

        public async Task<Teacher> GetTeacher(string id)
        {
            return await _teacherRepository.GetAsync(id);
        }

        public async Task<bool> UpdateTeacher(string id, Teacher teacher)
        {
            if (teacher == null || id == null)
            {
                return false;
            }
            try
            {
                var oldTeacher = await _teacherRepository.GetAsync(id);
                oldTeacher.FirstName = teacher.FirstName;
                oldTeacher.LastName = teacher.LastName;
                oldTeacher.Email = teacher.Email;
                oldTeacher.PhoneNumber = teacher.PhoneNumber;
                oldTeacher.UserName = teacher.UserName;
                oldTeacher.Subjects = teacher.Subjects;
                await _teacherRepository.UpdateAsync(oldTeacher);
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

LoginModel.cs:

```
using System.ComponentModel.DataAnnotations;
namespace SupportForSchoolActivities.Models.LoginModels
{
    public class LoginModel
    {
        [System.ComponentModel.DataAnnotations.Required(ErrorMessage = "Необхідно вказати
ім'я")]
        public string UserName { get; set; }
```

```
        [System.ComponentModel.DataAnnotations.Required(ErrorMessage = "Необхідно вказати
пароль")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

StudentVM.cs:

```
using Microsoft.AspNetCore.Mvc.Rendering;

namespace SupportForSchoolActivities.Models.RegisterModels
{
    public class StudentVM
    {
        public UserRegister Student { get; set; }
        public UserRegister Parent { get; set; }
        public IEnumerable<SelectListItem> SchoolClassSelectList { get; set; }
        public int SchoolClass { get; set; }
    }
}
```

TeacherVM.cs:

```
using Microsoft.AspNetCore.Mvc.Rendering;

namespace SupportForSchoolActivities.Models.RegisterModels
{
    public class TeacherVM
    {
        public UserRegister Teacher { get; set; }
        public IEnumerable<SelectListItem> SubjectsSelectList { get; set; }
        public List<int> SelectedSubjectsIds { get; set; }
    }
}
```

UserRegister.cs:

```
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Models.RegisterModels
{
    public class UserRegister
    {
        [Required(ErrorMessage = "Необхідно вказати ім'я")]
        public string FirstName { get; set; }
        [Required(ErrorMessage = "Необхідно вказати прізвище")]
        public string LastName { get; set; }
        [Required(ErrorMessage = "Необхідно вказати адресу електронної пошти")]
        [EmailAddress]
        public string Email { get; set; }
        [Required(ErrorMessage = "Необхідно вказати пароль")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Паролі не збігаються")]
        public string ConfirmPassword { get; set; }
    }
}
```

CreateSchoolClassVM.cs:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.ComponentModel.DataAnnotations;

namespace SupportForSchoolActivities.Models.ViewModels
{
    public class CreateSchoolClassVM
    {
        [Required(ErrorMessage = "Введіть назву класу")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Оберіть номер класу")]
        [Range(1, 11)]
        public int ClassNumber { get; set; }
        [Required(ErrorMessage = "Оберіть літеру класу")]
        public string ClassLetter { get; set; }
        public IEnumerable<SelectListItem> NumberClassSelectList { get; set; }
        public IEnumerable<SelectListItem> LettersClassSelectList { get; set; }
    }
}
```

OneSchoolClassVM.cs:

```
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Models.ViewModels
{
    public class OneSchoolClassVM
    {
        public IEnumerable<Student> Students { get; set; }
        public int SchoolClassId { get; set; }
        public string SchoolClassName { get; set; }
    }
}
```

ScheduleVM.cs:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Models.ViewModels
{
    public class ScheduleVM
    {
        public int ClassId { get; set; }
        public string ClassName { get; set; }
        public IEnumerable<SelectListItem> SubjectsSelectList { get; set; }
        public List<int> LessonNumbers { get; set; }
        public DayOfWeek DayOfWeek { get; set; }
        public List<Schedule> Schedules { get; set; }
    }
}
```

SchoolClassVM.cs:

```
using SupportForSchoolActivities.Domain.Entity;

namespace SupportForSchoolActivities.Models.ViewModels
{
    public class SchoolClassVM
```

```
    {
        public IEnumerable<SchoolClass> SchoolClasses { get; set; }
        public int MaxAmountOfClasses { get; set; }
    }
}
```

SubjectSchoolClassVM.cs:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.Build.Framework;

namespace SupportForSchoolActivities.Models.ViewModels
{
    public class SubjectSchoolClassVM
    {
        public IEnumerable<SelectListItem> SchoolClassSelectList { get; set; }
        [Required]
        public int SchoolClassId { get; set; }
        public IEnumerable<SelectListItem> SubjectSelectList { get; set; }
        [Required]
        public int SubjectId { get; set; }
    }
}
```

UserProfile.cs:

```
using AutoMapper;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Models.RegisterModels;

namespace SupportForSchoolActivities
{
    public class UserProfile : Profile
    {
        public UserProfile()
        {
            CreateMap<UserRegister, Admin>()
                .ForMember(u => u.UserName, opt => opt.MapFrom(x => x.Email));
            CreateMap<UserRegister, Student>()
                .ForMember(u => u.UserName, opt => opt.MapFrom(x => x.Email));
            CreateMap<UserRegister, Teacher>()
                .ForMember(u => u.UserName, opt => opt.MapFrom(x => x.Email));
            CreateMap<UserRegister, Parent>()
                .ForMember(u => u.UserName, opt => opt.MapFrom(x => x.Email));
        }
    }
}
```

WC.cs:

```
namespace SupportForSchoolActivities
{
    public static class WC
    {
        public const string AdminRole = "Admin";
        public const string StudentRole = "Student";
        public const string TeacherRole = "Teacher";
        public const string ParentRole = "Parent";
        public const int NumberOfLessons = 8;
    }
}
```

StudentController.cs:

```
using AutoMapper;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Models.RegisterModels;
using SupportForSchoolActivities.Service.Interfaces;
using SupportForSchoolActivities.Service.Interfaces.EntityInterfaces;

namespace SupportForSchoolActivities.Controllers.UsersControllers
{
    public class StudentController : Controller
    {
        private readonly IStudentService _studentService;
        private readonly ISchoolClassService _schoolClassService;
        private readonly IMapper _mapper;
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;

        public StudentController(IStudentService studentService, ISchoolClassService
schoolClassService,
            IMapper mapper, UserManager<User> userManager, RoleManager<IdentityRole>
roleManager)
        {
            _studentService = studentService;
            _schoolClassService= schoolClassService;
            _mapper = mapper;
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public async Task<IActionResult> Index()
        {
            IEnumerable<Student> students = await _studentService.GetAllStudents();
            students = students.OrderBy(s => s.SchoolClass.Name)
                .ThenBy(s => s.LastName)
                .ThenBy(s => s.FirstName)
                .ToList();
            return View(students);
        }

        [HttpGet]
        public async Task<IActionResult> Upsert()
        {
            var schoolClasses = await _schoolClassService.GetAllClasses();
            StudentVM studentVM = new StudentVM()
            {
                Student = new UserRegister(),
                Parent = new UserRegister(),
                SchoolClassSelectList = schoolClasses.Select(c => new SelectListItem
                {
                    Text = c.Name,
                    Value = c.Id.ToString()
                })
            };
            return View(studentVM);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Upsert(StudentVM studentVM)
```

```
        {
            var parent = _mapper.Map<Parent>(studentVM.Parent);
            var student = _mapper.Map<Student>(studentVM.Student);
            student.Parent = parent;
            student.SchoolClass = await
_schoolClassService.GetClass(studentVM.SchoolClass);
            var result = await _userManager.CreateAsync(student,
studentVM.Student.Password);
            if (result.Succeeded)
            {
                var role1 = await _roleManager.FindByNameAsync(WC.ParentRole);
                var role = await _roleManager.FindByNameAsync(WC.StudentRole);

                if (role != null && role1 != null)
                {
                    await _userManager.AddToRoleAsync(parent, role1.Name);
                    await _userManager.AddToRoleAsync(student, role.Name);
                    return RedirectToAction("Index");
                }
            }
            return View(studentVM);
        }

        public async Task<IActionResult> Delete(string id)
        {
            if(await _studentService.DeleteStudent(id))
            {
                return RedirectToAction("Index", "Student");
            }
            else
            {
                return RedirectToAction("Error", "Home");
            }
        }
    }
}
```

AdminController.cs:

```
using AutoMapper;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Models.RegisterModels;
using SupportForSchoolActivities.Service.Interfaces;

namespace SupportForSchoolActivities.Controllers.UsersControllers
{
    public class AdminController : Controller
    {
        private readonly IAdminService _adminService;
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly IMapper _mapper;

        public AdminController(IAdminService adminService, UserManager<User> userManager,
IMapper mapper, RoleManager<IdentityRole> roleManager)
        {
            _adminService = adminService;
            _userManager = userManager;
            _mapper = mapper;
            _roleManager = roleManager;
        }
```

```
        public async Task<IActionResult> Index()
        {
            IEnumerable<Admin> admins = await _adminService.GetListAdmins();
            return View(admins);
        }

        [HttpGet]
        public IActionResult Upsert()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Upsert(UserRegister userRegister)
        {
            var admin = _mapper.Map<Admin>(userRegister);
            var result = await _userManager.CreateAsync(admin, userRegister.Password);

            if (result.Succeeded)
            {
                var role = await _roleManager.FindByNameAsync(WC.AdminRole);
                if (role != null)
                {
                    await _userManager.AddToRoleAsync(admin, role.Name);
                    return RedirectToAction("Index", "Admin");
                }
            }
            return RedirectToAction("Error", "Home");
        }

        public async Task<IActionResult> Delete(string id)
        {
            if (await _adminService.DeleteAdmin(id))
            {
                return RedirectToAction("Index", "Admin");
            }
            else
            {
                return RedirectToAction("Error", "Home");
            }

        }
    }
}
```

ScheduleController.cs:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using SupportForSchoolActivities.Domain.Entity;
using SupportForSchoolActivities.Models.ViewModels;
using SupportForSchoolActivities.Service.Interfaces.EntityInterfaces;

namespace SupportForSchoolActivities.Controllers.EntitiesControllers
{
    public class ScheduleController : Controller
    {
        private readonly IScheduleService _scheduleService;
        private readonly ISchoolClassService _classService;
        private readonly ISubjectService _subjectService;
```

```
        public ScheduleController(IScheduleService scheduleService, ISchoolClassService
schoolClassService, ISubjectService subjectService)
        {
            _scheduleService = scheduleService;
            _classService = schoolClassService;
            _subjectService = subjectService;
        }

        public async Task<IActionResult> Index()
        {
            int max = 0;
            var schoolClasses = await _classService.GetAllClasses();
            for (int i = 1; i <= 11; i++)
            {
                var count = schoolClasses.Where(c => c.ClassNumber == i).Count();
                if (count > max)
                {
                    max = count;
                }
            }
            SchoolClassVM schoolClassVM = new SchoolClassVM()
            {
                SchoolClasses = schoolClasses,
                MaxAmountOfClasses = max
            };

            return View(schoolClassVM);
        }

        public async Task<IActionResult> ClassSchedule(int classId)
        {
            var schoolClass = await _classService.GetClass(classId);
            var subjects = await _subjectService.GetAllSubjects();
            var schedules = (await _scheduleService.GetAllSchedules())
                .Where(s => s.SchoolClass.Id == classId)
                .ToList();
            ScheduleVM scheduleVM = new ScheduleVM()
            {
                ClassId = schoolClass.Id,
                ClassName = schoolClass.Name,
                SubjectsSelectList = subjects.Select(s => new SelectListItem
                {
                    Text = s.Name,
                    Value = s.Id.ToString()
                }),
                Schedules = schedules.ToList()
            };

            TempData.Put("key", schedules);
            return View(scheduleVM);
        }

        [HttpGet]
        public async Task<IActionResult> Upsert(int classId, DayOfWeek day)
        {
            var allSchedules = TempData.Get<List<Schedule>>("key");
            var schedules = allSchedules.Where(s=>s.DayOfWeek== day).ToList();
            var subjects = await _subjectService.GetAllSubjects();

            ScheduleVM scheduleVM = new ScheduleVM()
            {
                ClassId = classId,
                SubjectsSelectList = subjects.Select(s => new SelectListItem
                {
```

```
                    Text = s.Name,
                    Value = s.Id.ToString()
                }),
                DayOfWeek = day,
                Schedules = schedules
            };

            return View(scheduleVM);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Upsert(ScheduleVM scheduleVM)
        {
            var schedulesBeforeEdit = (await _scheduleService.GetAllSchedules())
                .Where(s => s.SchoolClass.Id == scheduleVM.ClassId &&
                s.DayOfWeek == scheduleVM.DayOfWeek)
                .ToList();

            var schoolClass = await _classService.GetClass(scheduleVM.ClassId);

            var form = HttpContext.Request.Form;
            var scheduleSubjectList = form.ToList();
            Dictionary<int, int> editScheduleDic = new Dictionary<int, int>();
            foreach (var item in scheduleSubjectList)
            {
                if(item.Value != String.Empty)
                {
                    if(int.TryParse(item.Value.ToString(), out int value) &&
int.TryParse(item.Key.ToString(), out int key))
                    {
                       editScheduleDic.Add(key, value);
                    }
                }
            }

            var scheduleAfterEdit = new List<Schedule>();
            foreach(var schedule in editScheduleDic)
            {
                scheduleAfterEdit.Add(new Schedule()
                {
                    SchoolClass = schoolClass,
                    DayOfWeek = scheduleVM.DayOfWeek,
                    LessonNumber = schedule.Key,
                    Subject = await _subjectService.GetSubject(schedule.Value),
                });
            }
            int countWithoutChanges = 0;
            int countWithUpdates = 0;
            int countAdding = 0;
            int countDeleting = 0;

            foreach (var schedule in schedulesBeforeEdit)
            {
                if(scheduleAfterEdit.Any(s=>
                    s.SchoolClass.Id == schedule.SchoolClass.Id &&
                    s.DayOfWeek == schedule.DayOfWeek &&
                    s.LessonNumber == schedule.LessonNumber))
                {

                }
                else
                {
                    await _scheduleService.DeleteSchedule(schedule.Id);
```

```
                    countDeleting++;
                }
            }

            foreach(var schedule in scheduleAfterEdit)
            {
                if(schedulesBeforeEdit.Any(s =>
                    s.SchoolClass.Id == schedule.SchoolClass.Id &&
                    s.DayOfWeek == schedule.DayOfWeek &&
                    s.LessonNumber == schedule.LessonNumber))
                {
                    if (schedulesBeforeEdit.Any(s =>
                        s.SchoolClass.Id == schedule.SchoolClass.Id &&
                        s.DayOfWeek == schedule.DayOfWeek &&
                        s.Subject.Name == schedule.Subject.Name &&
                        s.LessonNumber == schedule.LessonNumber))
                    {
                        countWithoutChanges++;
                    }
                    else
                    {
                        var updateSchedule = schedulesBeforeEdit.FirstOrDefault(s =>
                            s.SchoolClass.Id == schedule.SchoolClass.Id &&
                            s.DayOfWeek == schedule.DayOfWeek &&
                            s.LessonNumber == schedule.LessonNumber);
                        await _scheduleService.UpdateSchedule(updateSchedule.Id,
schedule);
                        countWithUpdates++;
                    }
                }
                else
                {
                    await _scheduleService.CreateSchedule(schedule);
                    countAdding++;
                }
            }



            return RedirectToAction("Index");
        }


        }
    }
}
```

Index.cshtml (StudentController.cs):

```
@using SupportForSchoolActivities.Domain.Entity;
@model IEnumerable<Student>
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_LayoutAdmin.cshtml";
}

<div class="container p-3">
    <div class="row pt-4">
        <div class="col-6">
            <h2 class="text-primary">Учні</h2>
        </div>
        <div class="col-6 d-flex justify-content-between">
```

```
            <div></div>
            <div>
                <a asp-action="Upsert" class="btn btn-primary">Створити нового учня</a>
            </div>
        </div>
    </div>
    <br /><br />
    @if (Model.Count() > 0)
    {
        <table class="table table-bordered table-striped" style="width:100%">
            <thead>
                <tr>
                    <th>Прізвище</th>
                    <th>Ім'я</th>
                    <th>Клас</th>
                    <th>Username</th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td width="23%">@item.LastName</td>
                        <td width="23%">@item.FirstName</td>
                        <td width="10%">
                            @if(@item.SchoolClass != null)
                            {
                                @item.SchoolClass.Name
                            }
                        </td>
                        <td width="23%">@item.UserName</td>
                        <td width="21%" class="text-center">
                            <div class="w-75 btn-group text-center" role="group">
                                <a asp-action="Upsert" asp-route-id="@item.Id" class="btn
btn-primary mx-2">Редагувати</a>
                                <a asp-action="Delete" asp-route-Id="@item.Id" class="btn
btn-danger mx-2">Видалити</a>
                            </div>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    }
    else
    {
        <p>Учнів немає</p>
    }
</div>
```

Upsert.cshtml (StudentController.cs):

```
@using SupportForSchoolActivities.Models.RegisterModels;
@model StudentVM

@{
   ViewData["Title"] = "Upsert";
   Layout = "~/Views/Shared/_LayoutAdmin.cshtml";
}

<section class="vh-100">
   <div class="container h-100">
      <div class="row d-flex justify-content-center align-items-center h-100">
```

```
        <div class="col-lg-12 col-xl-11">
            <div class="card text-black" style="border-radius: 25px;">
                <div class="card-body p-md-5">
                    <form asp-action="Upsert" class="row justify-content-center">
                        <div class="col-md-10 col-lg-6 col-xl-5 order-2 order-lg-1">
                            <p class="text-center h1 fw-bold mb-5 mx-1 mx-md-4 mt-4">Додавання нового учня</p>
                            <div class="mx-1 mx-md-4" asp-action="Upsert">
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-user fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Student.FirstName" placeholder="Ім'я" class="form-control" />
                                        <span asp-validation-for="Student.FirstName" class="text-danger"></span>
                                    </div>
                                </div>
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-envelope fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Student.LastName" placeholder="Прізвище" class="form-control" />
                                        <span asp-validation-for="Student.LastName" class="text-danger"></span>
                                    </div>
                                </div>
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-lock fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <select asp-for="SchoolClass" asp-items="Model.SchoolClassSelectList" class="form-
control">
                                            <option disabled selected>--Обрати клас--</option>
                                        </select>
                                        <span asp-validation-for="SchoolClass" class="text-danger"></span>
                                    </div>
                                </div>
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-lock fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Student.Email" placeholder="Email" class="form-control" />
                                        <span asp-validation-for="Student.Email" class="text-danger"></span>
                                    </div>
                                </div>
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-key fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Student.Password" placeholder="Пароль" class="form-control" />
                                        <span asp-validation-for="Student.Password" class="text-danger"></span>
                                    </div>
                                </div>
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-key fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Student.ConfirmPassword" placeholder="Повторіть пароль" class="form-
control" />
                                        <span asp-validation-for="Student.ConfirmPassword" class="text-danger"></span>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-md-10 col-lg-6 col-xl-5 order-2 order-lg-1">
                            <p class="text-center h1 fw-bold mb-5 mx-1 mx-md-4 mt-4">Додавання одного з батьків</p>
                            <div class="mx-1 mx-md-4" asp-action="Upsert">
                                <div class="d-flex flex-row align-items-center mb-4">
                                    <i class="fas fa-user fa-lg me-3 fa-fw"></i>
                                    <div class="form-outline flex-fill mb-0">
                                        <input asp-for="Parent.FirstName" placeholder="Ім'я" class="form-control" />
                                        <span asp-validation-for="Parent.FirstName" class="text-danger"></span>
                                    </div>
                                </div>
```

```
                        <div class="d-flex flex-row align-items-center mb-4">
                            <i class="fas fa-envelope fa-lg me-3 fa-fw"></i>
                            <div class="form-outline flex-fill mb-0">
                                <input asp-for="Parent.LastName" placeholder="Прізвище" class="form-control" />
                                <span asp-validation-for="Parent.LastName" class="text-danger"></span>
                            </div>
                        </div>
                        <div class="d-flex flex-row align-items-center mb-4">
                            <i class="fas fa-lock fa-lg me-3 fa-fw"></i>
                            <div class="form-outline flex-fill mb-0">
                                <input asp-for="Parent.Email" placeholder="Email" class="form-control" />
                                <span asp-validation-for="Parent.Email" class="text-danger"></span>
                            </div>
                        </div>
                        <div class="d-flex flex-row align-items-center mb-4">
                            <i class="fas fa-key fa-lg me-3 fa-fw"></i>
                            <div class="form-outline flex-fill mb-0">
                                <input asp-for="Parent.Password" placeholder="Пароль" class="form-control" />
                                <span asp-validation-for="Parent.Password" class="text-danger"></span>
                            </div>
                        </div>
                        <div class="d-flex flex-row align-items-center mb-4">
                            <i class="fas fa-key fa-lg me-3 fa-fw"></i>
                            <div class="form-outline flex-fill mb-0">
                                <input asp-for="Parent.ConfirmPassword" placeholder="Повторіть пароль" class="form-
control" />
                                <span asp-validation-for="Parent.ConfirmPassword" class="text-danger"></span>
                            </div>
                        </div>
                    </div>
                    <div class="d-flex justify-content-center mx-4 mb-3 mb-lg-4">
                        <input type="submit" value="Створити студента" class="btn btn-primary btn-lg" />
                    </div>
                </div>
            </form>
        </div>
    </div>
</section>

@section Scripts{
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
        }
```