МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: «**РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ВІДВІДУВАНОСТІ ЗАНЯТЬ: МОДУЛЬ ВИКЛАДАЧ»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Пантюшко Ілля Русланович

Керівник:
Красюк Богдан Віталійович

Рецензент:
Гаврильчик Леонід Сергійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № ____ від « ___ » _____ 2022 р.

Острог, 2022

Міністерство освіти і науки України

Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА

«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу/проєкт студента

Пантюшка Іллі Руслановича

1. Тема роботи Розробка системи контролю відвідуваності занять: модуль викладач. *керівник роботи/проєкту* Красюк Богдан Віталійович.

Затверджено наказом ректора НаУОА від 29 жовтня 2021 року №110.

2. Термін здачі студентом закінченої роботи/проєкту: 03 червня 2022 року.

3. Вихідні дані до роботи/проєкту: система контролю відвідуваності занять під час навчального процесу.

4. Перелік завдань, які належить виконати:
Авторизація та аутентифікація,_сторінка викладача, студента та адміністратора, формування звітів успішності, інтерактивна залікова книжка, виставлення балів та присутності студентів на занятті.

5. Перелік графічного матеріалу: таблиці, рисунки.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Красюк Б. В.	01.12.2021р.	01.12.2021р.
2	Красюк Б. В.	01.12.2021р.	01.12.2021р.
3	Красюк Б. В.	01.12.2021р.	01.12.2021р.

7. Дата видачі завдання: 01.12.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	До 01.11.21	
2	Написання технічного завдання	До 01.12.21	
3	Розробка структури проекту	До 03.12.21	
4	Авторизація та аутентифікація	До 04.12.21	
5	Розробка схеми бази даних	До 05.12.21	
6	Написання CRUD команд	До 09.12.21	
7	Узгодження патернів розробки на фронтенді	До 09.12.21	
8	Розробка дизайну	До 11.12.21	
9	Реалізація ролей, підключення Nlog, реалізація приватних роутів	До 16.12.21	
10	Функціонал для перевірки курсів на активність, сторінка авторизації з таблицями	До 23.12.21	
11	Верстка сторінки студента	До 03.01.22	
12	Реалізація функціоналу виставлення оцінок та присутності на сторінці викладача	До 14.01.22	
13	Написання кваліфікаційної роботи/проєкту	До 18.01.22	
14	Перша перевірка кваліфікаційної роботи/проєкту керівником	До 28.01.22	
15	Реалізація сторінки викладача	До 07.02.22	
16	Реалізація сторінки адміністратора	До 20.02.22	
17	Реалізація звітів та імпорту засобами Excel	До 02.03.22	
18	Друга перевірка кваліфікаційної роботи/проєкту керівником	До 06.03.22	
19	Наповнення бази даних початковими даними	До 12.03.22	
20	Тестування програми	До 18.03.22	
21	Третя перевірка кваліфікаційної роботи/проєкту керівником	До 07.04.22	
22	Попередній захист кваліфікаційної роботи/проєкту	До 01.06.22	
23	Здача кваліфікаційної роботи/проєкту на кафедру	03.06.22	

Студент: _____ Ілля ПАНТЮШКО

Керівник кваліфікаційної роботи/проєкту: _____ Богдан КРАСЮК

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *Розробка системи контролю відвідуваності занять у навчальних закладах: модуль викладач.*

Автор: *Пантюшко І.Р.*

Науковий керівник: Красюк Б.В.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи/проєкту: *155 с., 40 рис., 2 табл., 9 додатків, 24 джерел.*

Ключові слова: відвідування занять, успішність студентів, система контролю відвідуваності, виставлення балів.

Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту було створення програмного продукту для контролю відвідуваності занять під час навчального процесу. На сайті можна створювати групи за окремим предметом. Викладач може перевіряти відвідуваність студентів та виставляти бали за кожне заняття. Студент може переглянути свою статистику успішності по кожному предмету, у вигляді звітів. В кінці семестру формується електронна залікова книжка та рейтинговий бал студента в групі. Користувачами сайту є студенти та викладачі вищих навчальних закладів.

The purpose of the thesis was to create a software product to control attendance during the learning process. On the site you can create groups for a particular subject. The teacher can check the attendance of students and set scores for each lesson. The student can view their performance statistics for each subject in the form of reports. At the end of the semester, an electronic record book and a student's rating score in the group are formed. Users of the site are students and teachers of higher educational institutions.

_____

ЗМІСТ

ВСТУП

В умовах стрімкого розвитку технологій, з’являються проблеми, які дедалі частіше вирішують за допомогою програмних продуктів.

Якщо ще декілька років назад люди активно користувалися паперовими документами, то на 2022 рік, особливо в період пандемії стає гострою потреба переходу на електронний документообіг.

Однією з причин такого переходу є зручність та надійність. Людина у будь-який момент часу з будь-якої точки землі, де є інтернет зможе отримати доступ до документів.

Якщо мова йде про ведення журналу відвідуваності занять в навчальних закладах, то тут також можна спостерігати значний прогрес. Паперові журнали все частіше почали замінювати на більш сучасні та ефективні рішення.

На сьогоднішній день вже існує чимало додатків для навчальних закладів, які здатні вирішувати ті чи інші проблеми. Але кожен навчальний заклад або навіть окремий викладач використовує власні методи та підходи до ведення журналу. Якщо навчальним закладом не передбачено дотримання єдиного способу ведення журналу, викладачі мають можливість вибирати серед різних існуючих варіантів. Найчастіше вони віддають перевагу зручності, доступності, а також спираються на свій попередній досвід. Чимало викладачів звикли користуватися паперовим варіантом ведення журналу, і їм інколи тяжко буває перейти на нову платформу. Нерідко, щоб почати використовувати ту чи іншу програму, потрібен час для ознайомлення з її інтерфейсом та дослідження її можливостей.

Тому, основним завданням, яке ми ставили перед собою, це зробити додаток доступним, з інтуїтивно зрозумілим інтерфейсом, щоб він не був нагромадженим зайвими функціями та модулями.

Як працює веб-застосунок.

На сторінці викладача зібрана вся необхідна інформація про поточні курси та студентів на цих курсах.

Групи на курс формуються адміністратором заздалегідь. У такому випадку викладачу не потрібно буде слідкувати за тим, хто приєднався до курсу, а хто ні.

Для того, щоб перемикатися між предметами, викладачу достатньо зайти на головну сторінку, вибрати необхідний предмет та перейти на нього.

Разом з тим, кожен студент має свою сторінку, де зібрано всю актуальну інформацію про поточні предмети та є доступ до успішності з конкретного предмету. В кінці семестру у студента формується залікова книжка.

Головною особливістю додатку є можливість формувати звіти успішності та відвідуваності на курсах.

ОСНОВНА ЧАСТИНА ТЕХНІЧНЕ ЗАВДАННЯ

Розробка системи контролю відвідуваності занять у навчальних закладах

Вимоги:

1. Авторизація через корпоративну пошту(студентів, викладачів).

2. Різні ролі користувача(студент, викладач, адміністратор).

3. Сторінка для студента – організована у вигляді залікової книжки, розрахунок поточного балу за кредитами.

При переході на предмет виводиться таблиця з n-ками та к-стю балів.

4. Сторінка для викладача – виводиться список дисциплін та груп на даний семестр.

При переході на конкретну групу – у викладача є вже готовий попередньо сформований адміністратором(деканатом) список груп студентів.

Викладач має змогу виставляти бали та кількість пропусків за заняття.

5. Сторінка для адміністратора. Адміністратор має змогу додавати, редагувати, видаляти студентів та викладачів, формувати групи з них.

6. Система повинна вміти будувати звіти, за предметом; за викладачем; за студентом.

Стек технологій:

Back-end:

- ASP.NET Core
- Entity Framework / MSSQL
- NLog
- Hangfire
- Axios

Front-end:

- ReactJS, React Hooks
- Redux
- Antd

РОЗДІЛ 1

ОПИС ІСНУЮЧИХ РІШЕНЬ

1.1. Паперовий варіант

Паперовий варіант ведення журналу відвідуваності в навчальних закладах все ще існує. Він не потребує доступу до інтернету та наявності комп’ютера чи певного виду гаджета.

Під час паперового ведення журналу відвідуваності та виставлення оцінок, зазвичай на початку курсу викладач просить в старости подати список групи і заповнити його. Далі протягом курсу він виставляє відмітки про присутність студента на парі, і бали. В кінці закінчення курсу викладач формує звіт про успішність кожного студента та передає його в деканат. Можна помітити значний недолік такого методу. Найперше, це затрачений час та зусилля для того, щоб порахувати підсумковий бал для кожного студента. Нерідко студенти хочуть слідкувати за своєю успішністю на курсі, а також знати про пропуски, для того, щоб мати змогу перездати їх. У випадку такого підходу дану інформацію студенти можуть дізнатись безпосередньо з уст викладача, або ж написати викладачу на пошту і попросити надати відповідні дані. В умовах прогресивного технічного розвитку це можна розцінювати, як проблему.

1.2. Excel

Microsoft Excel (повна назва Microsoft Office Excel) – табличний процесор, програма для роботи з електронними таблицями, створена корпорацією Microsoft для Microsoft Windows, Windows NT і Mac OS. Програма входить до складу офісного пакета Microsoft Office [1].

Проблеми, які з’являються при паперовому ведені журналу частково вирішуються, використовуючи програму Excel.

Такий метод ведення журналу є найпопулярнішим серед вищих навчальних закладів, але і він має ряд недоліків, які вирішує наш веб додаток.

Перший недолік, це записування груп вручну.

Сюди ж можна віднести ризик втрати даних, у випадку, коли програма використовується локально на комп’ютері, але така проблема вирішується з використанням хмарної технології Google Sheets.

Серед очевидних переваг над паперовим веденням журналу – це автоматичне сумування всіх балів та підбиття підсумків засобами самого Excel.

РОЗДІЛ 2

ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА

Розробку програмного забезпечення було розділено на декілька етапів – дизайн, бекенд та фронтенд.

Кожен з цих процесів вимагав використання певного, специфічного для конкретних задач програмного середовища.

На проекті було використано Microsoft Visual Studio 2019 для розробки серверної частини, Visual Studio Code для розробки інтерфейсу, Figma для проектування та малювання дизайну інтерфейсу.

2.1. Microsoft Visual Studio

Microsoft Visual Studio – це інтегроване середовище розробки від Microsoft, з безліччю можливостей, воно дозволяє розробляти як консольні програми, так і повноцінні веб-сайти та веб-додатки. Має найкращу підтримку в написанні коду на мовах C++, C#, та платформах Asp.Net, .Net Core [2]

Переваги Microsoft Visual Studio:

- Велика кількість розширень, які постійно оновлюються;
- IntelliSense – потужна технологія автодоповнення, має можливість дописувати назву методів при введенні початкових букв;
- Можливість налаштування робочої панелі під свої потреби;
- Має безкоштовну версію – Community.

2.2. Visual Studio Code

Visual Studio Code – це легкий, але потужний редактор коду з вбудованою підтримкою JavaScript, TypeScript і Node.js, має багату екосистему розширень для інших мов програмування.[3]

Редактори коду дозволяють зручно та легко розробляти інтерфейси програм, чи веб-сайтів, мають настроюваний зовнішній вигляд, підсвічування синтаксису, можливість додавання сторонніх плагінів для покращення взаємодії з користувачем, що дає високі переваги над типовими текстовими процесорами.

Переваги Visual Studio Code:

- Безкоштовний;
- Велика кількість плагінів;
- Динамічний набір тексту;
- Підтримка Git;
- Легка вага – редактор не потребує значних ресурсів операційної системи.

2.3. Figma

Figma – векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач. [4]

Figma є потужним інструментом, який має значні переваги над іншими програмами, що донедавна використовувались у проектуванні та малюванні дизайну сайтів. До прикладу, Adobe Photoshop на 2022 рік для створення web-інтерфейсів майже не використовується. Figma надає хороші можливості командної роботи над проектом, адже дозволяє зберігати всі макети не локально у себе на комп’ютері, а на віддаленому сервері. За рахунок цього кращою стає комунікація дизайнера та

програміста, а для внесення будь-яких правок у проект витрачається набагато менше часу. Зменшується ризик втрати будь-яких важливих файлів.

2.4. Git, GitHub

Git – розподілена система контролю версій, яка дає можливість розробникам відстежувати зміни у файлах та працювати над одним проектом разом із колегами.

Підхід Git до зберігання даних подібний до набору знімків мініатюрної файлової системи. Щоразу, коли ми зберігаємо стан свого проекту в Git, система запам'ятовує, як виглядає кожен файл у цей момент, та зберігає посилання на цей знімок.

Переваги Git:

- Безкоштовний та open-source.
- Виконує всі операції локально, що збільшує його швидкість. Крім того, Git локально зберігає весь репозиторій у невеликий файл без втрати якості даних;
- Резервне копіювання. Git ефективний у зберіганні бекапів, тому відомо мало випадків, коли хтось втрачав дані під час використання Git;
- Просте розгалуження. В інших системах контролю версій створення гілок – стомлююча і трудомістка задача, тому що весь код копіюється в нову гілку. У Git управління гілками реалізовано набагато простіше та ефективніше.

Так як розробка проекту виконувалась у команді необхідною складовою було використання онлайн-хостингу репозиторіїв GitHub, який володіє всіма функціями розподіленого контролю версій та функціональністю управління вихідним кодом – все, що підтримує Git і навіть більше. Git-репозиторій, завантажений на GitHub, доступний за допомогою інтерфейсу командного рядка Git та Git-команд. Також є й інші функції: документація, запити на прийняття змін (pull requests), історія коммітів, інтеграція з безліччю популярних сервісів, email-повідомлення, емодзі, графіки, вкладені списки завдань, система @згадувань і т.д.

2.5. Hangfire

Hangfire – багатопотоковий і масштабований планувальник завдань, побудований за клієнт-серверною архітектурою на стеку технологій .NET (насамперед Task Parallel Library та Reflection), з проміжним зберіганням завдань у БД. Цілком функціональний у безкоштовній (LGPL v3) версії з відкритим вихідним кодом. Зазвичай Hangfire використовують для створення фонових задач які будуть повторюватися з певним проміжком часу, однією з таких задач в нашому застосунку є перевірка і зміна статусу предмету в залежності від дати. Якщо термін активності предмету минув він переходить в статус не активний що свідчить, що курс завершився.

2.6. Тестування Api

В якості програмного забезпечення для тестування серверної частини та окремих функцій було вирішено використати Postman та Swagger.

2.6.1. Swagger

Swagger – це мова опису інтерфейсу для опису RESTful API, виражених за допомогою JSON. Swagger використовується разом із набором програмних засобів з відкритим кодом для проектування, створення, документування та використання веб-служб RESTful. Swagger включає автоматизовану документацію, генерацію коду та тестування. Додаток А

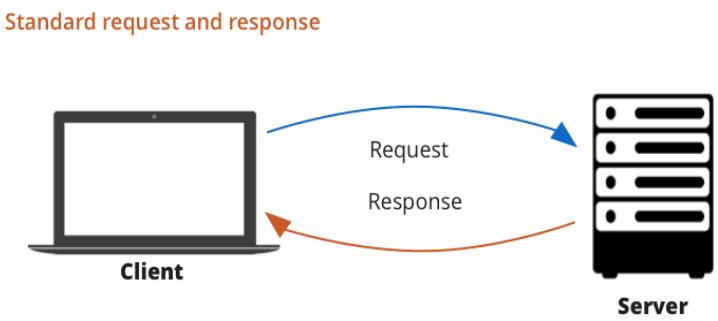


Рис. 2.1 «Схема взаємодії клієнтської та серверної частини»

2.6.2. Postman

Postman було обрано, оскільки він є найпопулярнішим із всіх можливих рішень.

Переваги Postman:

- інтуїтивно-зрозумілий і простий у використанні, не вимагає якогось складного налаштування або знання мов програмування, безкоштовний;
- підтримує різні API (REST, SOAP, GraphQL);
- розширюється під будь-які потреби за допомогою Postman API;
- легко інтегрується в CI/CD за допомогою Newman – консольної утиліти для запуску тестів;
- запускається на будь-яких ОС;
- підтримує ручне та автоматизоване тестування;
- зібрав навколо себе велике ком'юніті, де можна знайти відповіді на будь-які питання.

Тестувальнику цей інструмент дозволяє:

- надсилати запити та отримувати відповіді;
- зберігати запити до папок та колекцій;
- змінювати параметри запитів;
- змінювати оточення (dev, test, production);
- виконувати автотести, використовуючи Collections runner, зокрема за розкладом;
- імпортувати та експортувати колекції запитів та набори тестів, щоб обмінюватися даними з колегами.

2.7. Звіти

Однією з найголовніших задач на проекті було створення звітів за окремими сутностями.
В процесі розробки було вирішено реалізувати цю задачу в вигляді Excel таблиць. В якості інструмента було вибрано бібліотеку EPPlus.

EPPlus – це бібліотека .NET, яка читає та записує файли Excel у форматі Office Open XML (xlsx). EPPlus не має інших залежностей, крім .NET.

EPPlus підтримує:

- Діапазони клітинки;
- Оформлення клітинки (рамка, колір, заливка, шрифт, число, вирівнювання);
- Перевірка даних;
- Умовне форматування;
- Діаграми;
- Картинки;
- Фігури;
- Коментарі;
- Таблиці;
- Зведені таблиці;
- Охорона;
- Шифрування;
- VBA;
- Розрахунок формули.

РОЗДІЛ 3

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На першому етапі розробки програми необхідно було визначитись з майбутньою архітектурою. У виборі між монолітною та багатошаровою архітектурою ми зупинились на другому варіанті.

3.1. Що таке багатошарова архітектура

Багатошарова архітектура зосереджена на групуванні пов’язаних функцій у програмі в окремі шари, які накладаються вертикально один на одного. Кожен шар має унікальні простори імен і класи. Компоненти кожного рівня взаємодіють з компонентами іншого рівня за допомогою чітко визначених інтерфейсів. Багатошарова архітектура має три рівні:

- Рівень презентації – це перший і найвищий рівень, на якому користувачі можуть взаємодіяти з програмою.
- Рівень бізнес логіки – це середній рівень, який містить всю бізнес-логіку програми, описує, як бізнес-об’єкти взаємодіють один з одним. Цей рівень виступає посередником між інтерфейсом програми і базою даних.
- Рівень доступу до даних – забезпечує спрощений доступ до даних, що зберігаються в постійному сховищі.

Тришарова архітектура працює наступним чином:

Рівень презентації – це єдиний клас, який безпосередньо взаємодіє з користувачем. В основному він використовується для представлення/збору даних від користувачів, а потім передає їх на рівень бізнес-логіки для подальшої обробки.

Після отримання інформації від рівня презентації рівень бізнес-логіки виконує певну

бізнес-логіку над даними. Під час цього процесу цей рівень може отримати або оновити деякі дані з бази даних програми. Однак цей рівень не несе відповідальності за доступ до бази даних, він надсилає запити до наступного рівня, рівня доступу до даних.

Рівень доступу до даних отримує запити від рівня бізнес-логіки і створює деякі запити до бази даних для обробки цих запитів. Після завершення виконання він надсилає результат назад на рівень бізнес-логіки.

Рівень бізнес-логіки отримує відповіді від рівня доступу до даних, потім завершує процес і надсилає результат на рівень презентації.

Рівень презентації отримує відповіді та представляє їх користувачам за допомогою компонентів інтерфейсу користувача. [5]

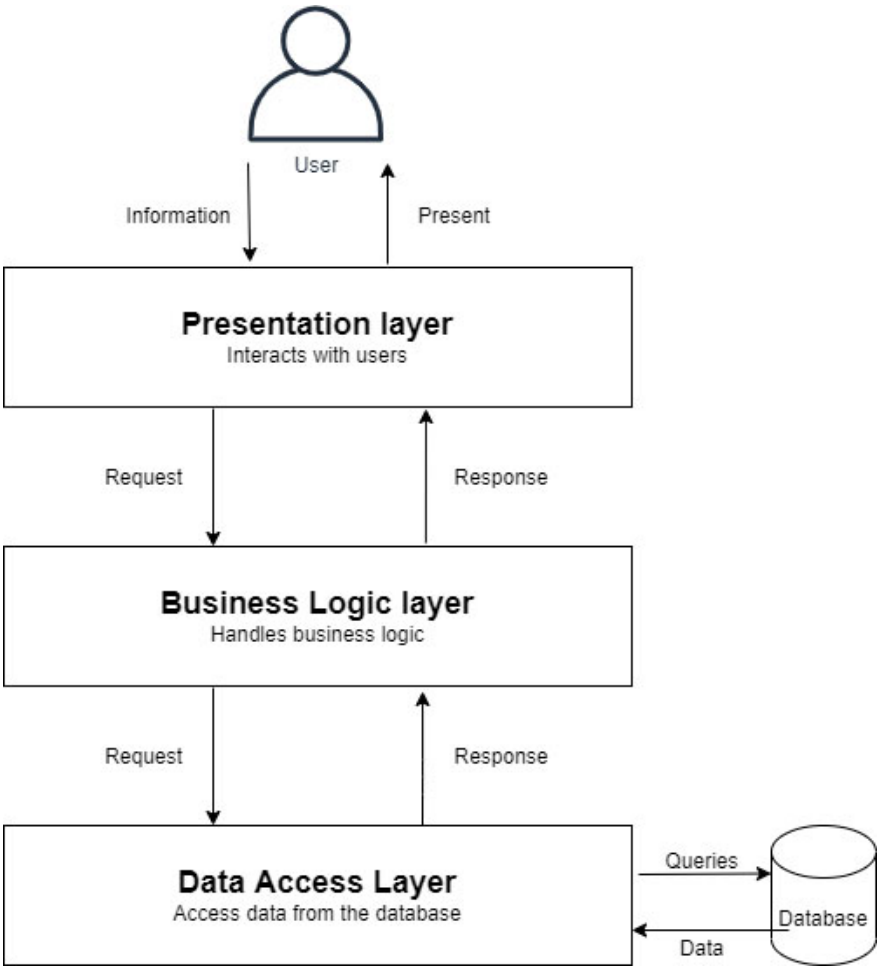


Рис. 3.1. Схема багатошарової архітектури

3.2. Переваги багатошарової архітектури

- Чистота та підтримка коду

  Кожен шар відповідає лише за свій процес і є ізольованим від інших. Такий код легко читати та підтримувати.
- Висока масштабованість

  Зміни в коді, або якогось з рівнів не впливатимуть на інші шари програми.
- Легкий розподіл робочого навантаження

  Кожен член команди може писати свою частину коду на кожному рівні незалежно, що, у свою чергу, допомагає розробникам контролювати своє робоче навантаження.

3.3. Реалізація багатошарової архітектури на проекті

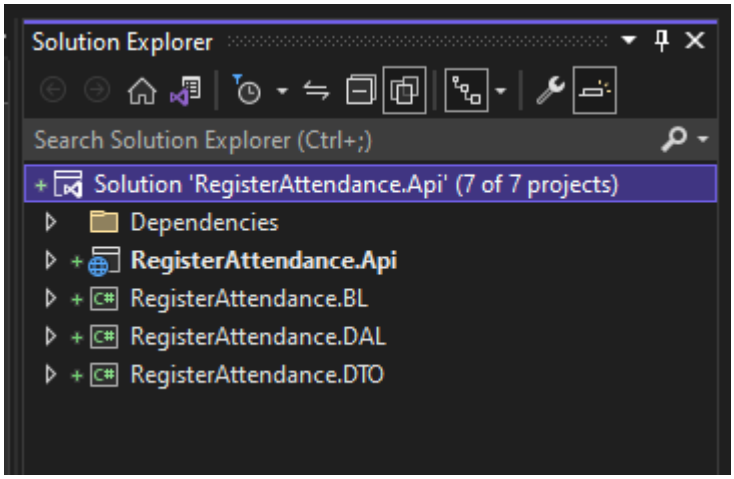


Рис. 3.2 Багатошарова архітектура

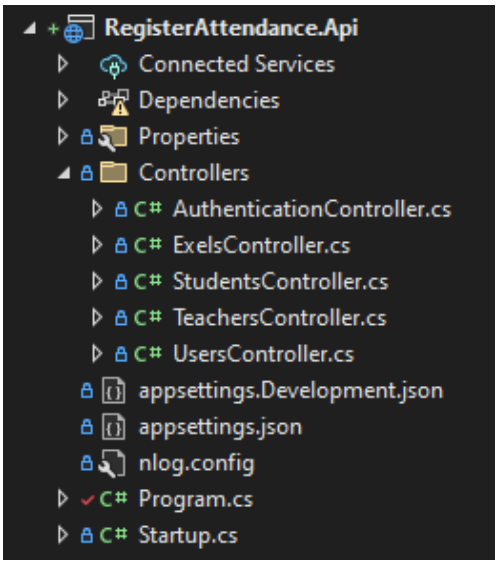


Рис. 3.3. Рівень презентації

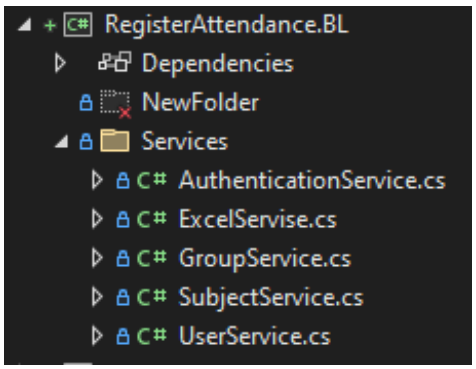


Рис. 3.4. Рівень бізнес логіки

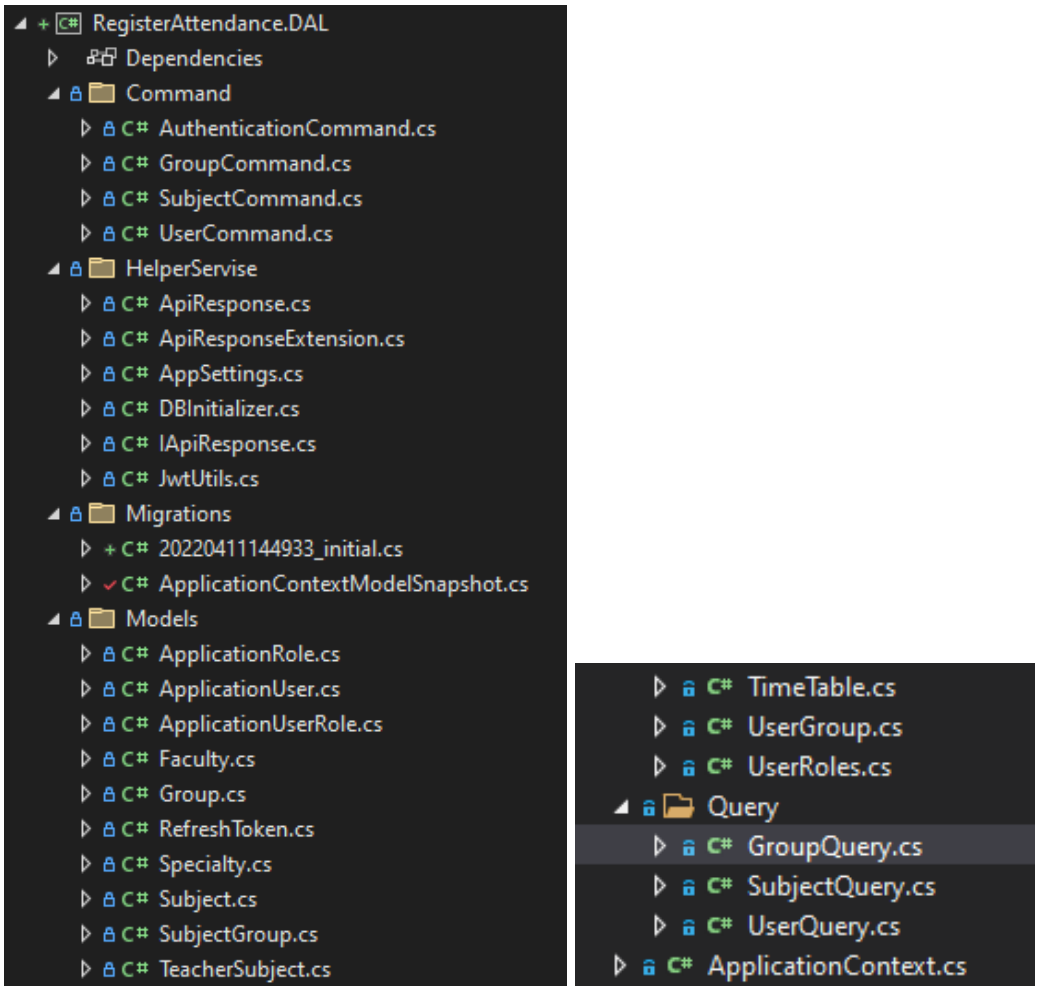


Рис. 3.5. Рівень доступу до даних DAL

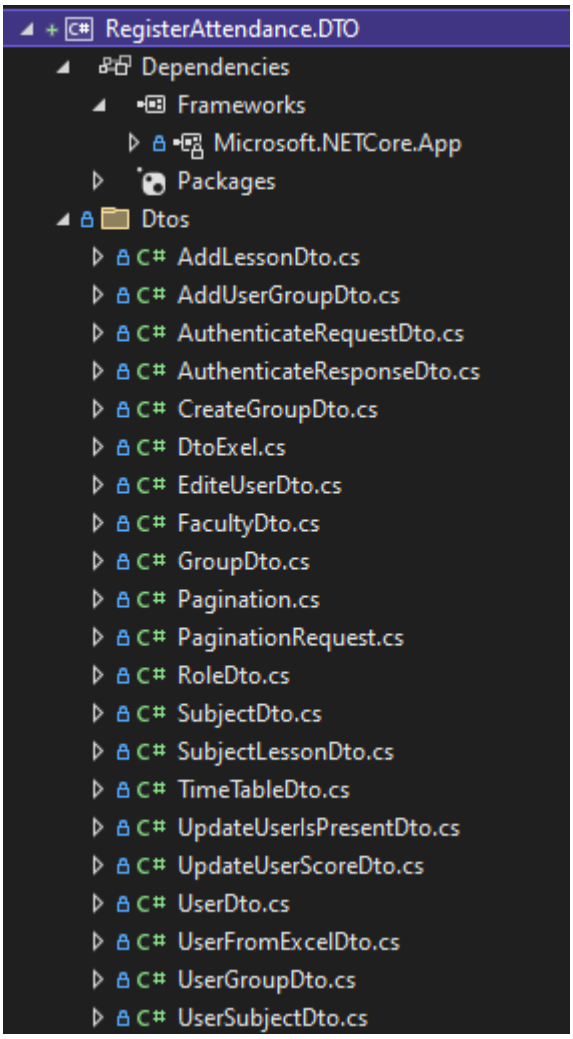


Рис. 3.6. Рівень доступу до даних DTO

Табл. 3.1. Рівень презентації, RegisterAttendance.API

Controllers	Функціонал	Методи
AuthenticationController	Аутентифікація за JWT токенами	– [HttpPost("authenticate")] – [HttpPost("refresh-token")] – [HttpGet("getCurrentUser")]
ExcelsController	Заповнення таблиць бази даних за допомогою Excel: – додавання користувачів; – додавання предметів; – додавання предметів студенту; – додавання факультетів; – отримання предметів; – отримання користувачів .	– [HttpPost("AddUsersFromExel")] – [HttpPost("AddSubjectsFromExel")] – [HttpPost("addStudentSubjectFromExel")] – [HttpPost("AddFacultyFromExel")] – [HttpGet("exportToExcelBySubject")] – [HttpGet("exportToExcelByUser")]
StudentsController	Операції для студентів: – додавання факультету; – додавання груп до факультету; – додавання спеціальності; – додавання користувача до групи; – отримання факультетів; – отримання груп; – отримання дати занять певного студента; – отримання предметів певного студента;	– [HttpPost("createFaculty")] – [HttpPost("addFacultyGroup")] – [HttpPost("addGroupSpecialty")] – [HttpPost("addUserGroup")] – [HttpGet("getFaculty")] – [HttpGet("getGroups")] – [HttpGet("getTimeTableByUser")] – [HttpGet("GetAllSubject")]

Продовження табл. 3.1.

TeachersController	Операції для викладачів: – отримання викладачів за факультетом; – отримання викладачів за групою; – отримання всіх предметів; – отримання дати занять певного викладача; – отримання груп за предметом; – отримання всіх предметів певного викладача; – отримання успішності по предмету; – додавання предмету; – додавання заняття; – оновлення предмету; – видалення предмету; – видалення заняття; – оновлення балу студента за заняття; – оновлення присутності студента на занятті.	– [HttpGet("getAllTeacherFaculty")] – [HttpGet("getAllTeacherGroups")] – [HttpGet("getAllSubjests")] – [HttpGet("getUsersTimeTable")] – [HttpGet("getAllGroupsBySubject")] – [HttpGet("getAllTeacherSubject")] – [HttpGet("GetSubjectTopic")] – [HttpPost("createSubject")] – [HttpPost("addLesson")] – [HttpPut("updateSubject")] – [HttpDelete("deleteSubject")] – [HttpDelete("deleteLesson")] – [HttpPut("updateUserScore")] – [HttpPut("updateUserIsPresent")]

Продовження табл. 3.1.

UsersController	Операції для користувачів: – отримання користувачів; – отримання всіх груп; – отримання користувачів з предметами; – отримання даних по заняттю; – додавання ролі; – видалення ролі; – створення користувача; – створення груп; – додавання користувача до групи; – додавання заняття; – додавання групи; – оновлення користувачів; – видалення предмету; – додавання предмету користувачу; – отримання предметів; – видалення користувачів; – видалення груп	– [HttpGet("getUsers")] – [HttpGet("getAllGroups")] – [HttpGet("getAllUsersWithSubjests")] – [HttpGet("getTimeTable")] – [HttpPost("addRole")] – [HttpPost("deleteRole")] – [HttpPost("createUser")] – [HttpPost("addGroup")] – [HttpPost("addUserGroup")] – [HttpPost("createTimeTable")] – [HttpPost("createGroup")] – [HttpPut("updateUser")] – [HttpDelete("removeSubject")] – [HttpPost("addUserSubject")] – [HttpGet("getSubjects")] – [HttpDelete("deleteUser")] – [HttpDelete("deleteGroup")]

Додаток Б

Табл. 3.2. Рівень бізнес логіки, RegisterAttendance.BL

Services	Функціонал	Методи
AuthenticationService	Аутентифікація за JWT токенами	– Authenticate, – RefreshToken
ExcelService	– Записування користувачів; – записування факультетів та груп; – записування студентів та його предметів; – записування факультетів; – записування предметів; – завантаження звіту успішності за предметом; – завантаження звіту успішності для певного студента	– Import – ImportFacultyGroups – ImportStudentSubject – ImportFaculty – ImportSubject – ExportToExcelBySubject – ExportToExcelByUser
GroupService	– Додавання групи до факультету; – додавання факультету; – додавання спеціальності; – отримання груп; – отримання факультетів; – отримання всіх груп; – додавання користувача до групи; – видалення групи; – додавання групи за предметом	– AddFacultyGroupAsync – CreateFacultyAsync – AddGroupSpecialtyAsync – GetGroups – GetFaculty – GetAllGroups – AddUserGroup – DeleteGroup – AddGroupSubject

Продовження табл. 3.2.

SubjectService	– Додавання передмету; – оновлення передмету; – видалення предмету; – отримання передмету; – отримання всіх предметів; – отримання всіх користувачів та предметів; – отримання всіх викладачів певного факультету; – отримання всіх викладачів певної групи; – отримання всіх груп за предметом; – отримання всіх предметів; – додавання заняття; – видалення заняття	– CreateSubjectAsync – UpdateSubjectAsync – DeleteSubject – GetSubjects – GetAllSubjects – GetAllUsersWithSubjests – GetAllTeacherFaculty – GetAllTeacherGroups – GetAllGroupsBySubject – GetAllSubject – AddLessonAsync – DeleteLessonAsync

Продовження табл. 3.2.

UserService	– Отримання користувачів; – отримання успішності за заняття; – додавання ролі користувачеві; – видалення ролі користувача; – додавання користувача; – оновлення користувача; – додавання групи; – видалення предмету; – отримання користувача; – додавання заняття; – видалення користувача; – додавання користувача; – отримання викладачів та їх предметів; – отримання заняття; – отримання успішності за заняття; – оновлення балу студента за заняття – оновлення присутності студента; – отримання заняття з предмету та студентів; – створення груп	– GetUsers – GetTimeTable – AddRoleAsync – DeleteRolesAsync – CreateUserAsync – UpdateUserAsync – AddGroup – RemoveSubject – GetUser – CreateTimeTableAsync – DeleteUser – AddRangeUsers – GetAllTeacherSubject – GetUsersTimeTable – GetSubjectTopic – UpdateUserScore – UpdateUserIsPresent – GetTimeTableByUser – CreateGroup

Додаток В

На рівні доступу до даних RegisterAttendance.DAL вся логіка на отримання запитів від рівня бізнес-логіки та створення запитів до бази даних зберігається в папці Command.

В ній розміщено відповідні класи:

- AuthenticationCommand.cs
- GroupCommand.cs
- SubjectCommand.cs
- UserCommand.cs

Додаток Г

В папці HelperService розміщено допоміжні класи:

- ApiResponse.cs
- ApiResponseExtention.cs
- ApiSettings.cs
- DBInitializer.cs
- IApiResponse.cs
- JwtUtils.cs

Додаток Д

3.4. CQRS

Для реалізації DAL було використано архітектурний патерн проектування CQRS. CQRS – це стиль архітектури, в якому операції читання відокремлені від операцій запису. Підхід сформулював Грег Янг з урахуванням принципу CQS, запропонованого Бертраном Мейєром. Найчастіше CQRS реалізується в обмежених контекстах (bounded context) додатків, проектованих з урахуванням DDD. Одна з природних причин розвитку CQRS – не симетричний розподіл навантаження та складності бізнес-логіки на read та write-підсистеми. Більшість бізнес-правил та складних перевірок знаходиться у write-підсистемі. При цьому читають дані частіше, ніж змінюють. В класичній реалізації патерн передбачає використання додаткового компонента медіатора. Для нашого випадку було використано реалізацію без

використання медіатора, це було зроблено з метою покращення простоти і зрозумілості структури.

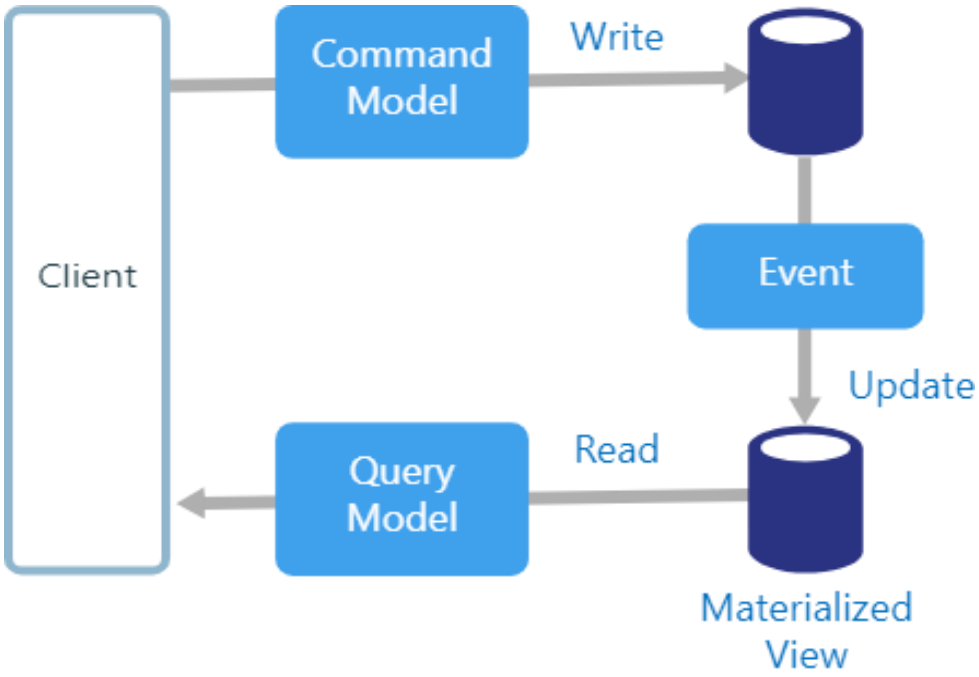


Рис. 3.7. Структура роботи патерна CQRS

РОЗДІЛ 4

БАЗА ДАНИХ

База даних є головною частиною будь-якого програмного забезпечення, а якість проектування бази даних безпосередньо впливає на всі майбутні процеси розробки. До якісного проектування бази даних можна віднести такі етапи:

1 Виділення сутностей та його атрибутів, які зберігатимуться у базі даних, і формування з них таблиць.
2 Визначення унікальних ідентифікаторів (первинних ключів) об'єктів, що зберігаються у рядках таблиці.
3 Визначення відносин між таблицями за допомогою зовнішніх ключів.
4 Нормалізація бази даних.

Для того, щоб спростити роботу взаємодії з базою даних було використано Entity Framework Core.

4.1. Entity Framework Core

Entity Framework Core (EF Core) – це об'єктно-орієнтована, легка і розширювана технологія від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping – відображення даних як реальних об'єктів).[6]

Особливістю EF Core є можливість працювати з базами даних, використовуючи більш високий рівень абстракції, що дозволяє абстрагуватися від самої бази даних, її таблиць та працювати з даними незалежно від типу сховища.

Відмінністю між використанням звичайної бази даних є те, що якщо фізично ми оперуємо таблицями, індексами, первинними та зовнішніми ключами, то на

концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує багато різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-яким СУБД, якщо для неї є потрібний провайдер. А ключовою перевагою використання EF Core є наявність універсального API для роботи з даними. У будь-який момент ми можем змінити цільову СУБД, і всі зміни, які потрібно буде зробити – це зміна конфігурації та налаштування підключення до відповідних провайдерів.

Поведінка Entity Framework за замовчуванням полягає у завантаженні тільки тих сутностей, які безпосередньо необхідні вашому додатку. Якщо Entity Framework прямо завантажує всі сутності, пов'язані через один чи кілька асоціацій, швидше за все, буде завантажено більше сутностей, ніж було би потрібно. Це збільшить обсяг пам'яті, а також вплине на продуктивність.

В Entity Framework можна контролювати, коли відбувається завантаження пов'язаних сутностей, та оптимізувати кількість виконуваних запитів до бази даних. Ретельне керування завантаженням пов'язаних об'єктів може підвищити продуктивність програми і дати більше контролю над даними.

4.1.1. Підхід Code first, міграції

В Entity Framework було представлено підхід Code-First у версії 4.1. У підході Code-First ви зосереджуєтеся на предметній області програми та починаєте створювати класи для свого доменного об'єкта, а не спочатку проектуєте свою базу даних, а потім створюєте класи, що відповідають вашій структурі бази даних. Такий підхід застосовується, коли у вас немає готової бази даних, і ви хочете створити та підтримувати її безпосередньо зі свого коду.[7]

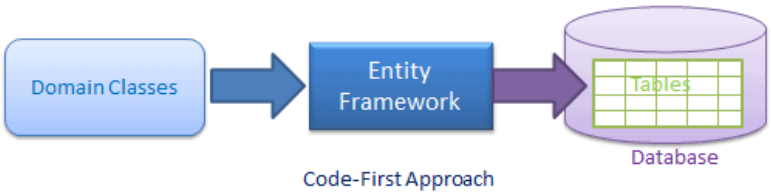


Рис. 4.1. Code first підхід для створення бази даних

Процес створення бази даних на основі Code first підходу відбувається за такою схемою: спочатку створюються моделі бази даних, в коді це виглядатиме, як звичайні класи, далі ці класи описуються на основі Fluent-API або атрибутів анотації даних, і на останньому етапі створюється схема бази даних за допомогою автоматичних міграцій або міграцій на основі коду.

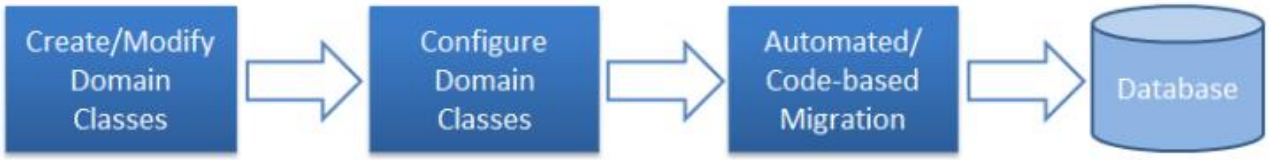


Рис. 4.2. Структура Code first підходу

Підхід Code first має ряд переваг та недоліків.

Переваги:

- Швидкість розробки. Не потрібно турбуватися про створення бази даних, можна відразу приступити до кодування та створення класів сутностей. Підходить для розробників, які не мають багато досвіду роботи з базами даних.
- Автоматичні оновлення бази даних за допомогою міграцій. При внесенні будь-яких змін в модель, схема бази даних автоматично оновлюється.
- POCOs – код набагато чистіший, менше автоматично згенерованого коду.
- Є повний контроль над кожним з класів сутностей без особливих знань SQL.
- Не потрібно підтримувати модель EDMX для оновлення бази даних.
- Контроль версією бази даних.

Недоліки:

- Створення вручну моделей предметної області є не зручним, коли на проекті є декілька сотень таблиць.

- Потрібні хороші знання мови C# та її угоди з Entity Framework.
- Не рекомендується в програмах з великим обсягом даних, складна логіка для створення або обслуговування цих даних.[8]
- Якщо до бази даних буде внесено будь-які зміни, вони не відобразяться на об'єктах у додатку.

В Entity Framework з'явився інструмент міграції, який автоматично оновлює схему бази даних при зміні моделі без втрати існуючих даних або інших об'єктів бази даних. Він використовує новий ініціалізатор бази даних під назвою MigrateDatabaseToLatestVersion.

Загалом міграції працюють в такий спосіб:

При зміні моделі даних розробник використовує засоби EF Core, щоб додати відповідну міграцію з описом оновлень, необхідних для синхронізації схеми бази даних. EF Core порівнює поточну модель із моментальним знімком старої моделі для визначення відмінностей та створює вихідні файли міграції. Файли можна відстежувати в системі керування версіями проекту, як будь-які інші вихідні файли.

Створену міграцію можна застосовувати до бази даних у різний спосіб. EF Core записує всі застосовані міграції до спеціальної таблиці журналу, з якої буде ясно, які міграції були застосовані, а які ні.[9]

Існує два види міграцій: автоматична міграція та міграція на основі коду.

На проекті було використано міграцію на основі коду.

Міграція на основі коду забезпечує більший контроль над міграцією і дозволяє налаштовувати додаткові параметри, такі як встановлення значення стовпця за замовчуванням, настроювання стовпця, що обчислюється, і т.д.

Щоб використовувати міграцію на основі коду, необхідно виконати наступні команди в консолі диспетчера пакетів Visual Studio:

Enable-Migrations: включає міграцію у проекті шляхом створення Configuration класу.

Add-Migration: створює новий клас міграції відповідно до вказаного імені за допомогою методів Up() та Down().

Update-Database: виконує останній файл міграції, створений Add-Migration командою, та застосовує зміни до схеми бази даних.

4.1.2. Завантаження пов'язаних даних

В Entity Framework існує три підходи для завантаження пов’язаних даних: “відкладене завантаження”(lazy loading), “пряме завантаження”(eager loading) і явне завантаження(explicit loading). За допомогою цих підходів забезпечується однакове завантаження даних, але вони впливають на продуктивність програми.[10]

Відкладене завантаження (lazy loading)

Відкладене завантаження (lazy loading) полягає у тому, що Entity Framework автоматично завантажує дані. Коли потрібні зв'язувані дані, Entity Framework створить ще один запит до бази даних.

Entity Framework застосовує відкладене завантаження за допомогою динамічних проксі-об'єктів. Ось як це працює. Коли Entity Framework повертає результати запиту, він створює екземпляри класів та заповнює їх даними, які були повернуті з бази даних. Entity Framework має можливість динамічно створювати новий тип під час виконання, похідний від класу моделі. Цей новий клас виступає як проксі-об'єкт для класу і називається динамічним проксі-об'єктом. Він буде перевизначати навігаційні властивості класу і включати деяку додаткову логіку для вилучення даних з бази даних, коли викликається навігаційна властивість.

Для того, щоб використовувати динамічні проксі-об'єкти є кілька умов, яким повинен відповідати клас моделі. Якщо ці умови не виконуються, Entity Framework, не буде створювати динамічні проксі-об'єкти для класу і просто повертатиме екземпляри класу, які не можуть виконувати відкладене завантаження, а отже, взаємодіяти з базою даних:

- Клас моделі повинен мати модифікатор доступу public і не повинен забороняти спадкування (ключове слово sealed у C#).
- Навігаційні властивості, які мають забезпечувати відкладене завантаження, мають бути віртуальними (модифікатор virtual).

Відкладене завантаження є дуже простим у використанні, тому що виконується автоматично. При цьому воно є досить небезпечним в плані продуктивності.[10]

Явне завантаження(explicit loading)

Явне завантаження, як і відкладене завантаження, не призводить до завантаження всіх пов'язаних даних у першому запиті. Але при цьому, на відміну від відкладеного завантаження, при виклику навігаційної властивості зв'язаного класу, це завантаження не призводить до автоматичного вилучення пов'язаних даних, для цього потрібно викликати метод Load(), щоб завантажити пов'язані дані. Такий тип завантаження може використовуватись у таких випадках:

- Цей тип завантаження усуває необхідність відзначати навігаційні характеристики класу моделі як віртуальні. Це може здатися не значною зміною, але той факт, що технологія доступу до пов'язаних даних при відкладеному завантаженні вимагає змінювати класи POCO моделі, далека від ідеальної.
- Є можливість працювати з існуючою бібліотекою класів, де навігаційні властивості не позначені як віртуальні, і не можливо змінити бібліотеку.
- Зрештою, явне завантаження дозволяє бути впевненим, у тому, що ви точно знаєте, коли запити надсилаються до бази даних.[11]

Пряме завантаження(eager loading)

Пряме завантаження даних (eager loading) дозволяє вказати у запиті, які пов'язані дані потрібно завантажити під час виконання запиту. Завдяки цьому, коли у коді потрібно

буде посилатися на пов'язану таблицю через навігаційну властивість, SQL-запит не буде направлятися до бази даних, так як пов'язані дані вже будуть завантажені при першому запиті. Entity Framework для цих цілей використовується метод Include(), якому передається делегат, в якому можна вказати навігаційну властивість, за якою дані повинні завантажуватися при першому запиті. Цей метод є розширюючим для IQueryable.

Метод Include() можна використовувати для завантаження кількох зв'язаних таблиць. Цей метод повертає тип IQueryable<TEntity>, де TEntity це базовий тип, в колекції якого він був викликаний, тому можна використовувати ланцюжок викликів Include.[10]

На проекті було використано пряме завантаження.

На рівні доступу до даних в папці Query реалізовано три класи:

- GroupQuery.cs
- SubjectQuery.cs
- UserQuery.cs

Додаток Е

4.1.3. Складові Бази даних

Сутність (entity) представляє тип об'єктів, які повинні зберігатися в базі даних. Кожна таблиця у базі даних має представляти одну сутність. Як правило, сутності відповідають об'єктам із реального світу. Кожна сутність визначає набір атрибутів. Атрибут є властивістю, яка описує деяку характеристику об'єкта.[12]

Ключі є способом ідентифікації рядків у таблиці. За допомогою ключів ми можемо встановлювати зв’язки з іншими таблицями.

Первинний ключ (primary key) безпосередньо застосовується для ідентифікації рядків таблиці. Він повинен відповідати наступним обмеженням:

- унікальний;
- обов’язковий для кожної окремої сутності;

- незмінний;

Бази даних можуть містити таблиці, пов'язані між собою різними зв'язками. Зв'язок (relationship) є асоціацією між сутностями різних типів.
Для організації зв'язку застосовуються зовнішні ключі.
Зовнішній ключ представляє один або кілька стовпців однієї таблиці, який одночасно є потенційним ключем з іншої таблиці. Зовнішній ключ необов'язково має відповідати первинному ключу головної таблиці. Хоча, як правило, зовнішній ключ із залежної таблиці вказує на первинний ключ із головної таблиці.[13]

Зв'язки між таблицями бувають наступних типів:

- Один до одного (One to one).
- Один до багатьох (One to many).
- Багато хто до багатьох (Many to many).

4.2. Схема бази даних

На етапі проектування бази даних за основу брались завдання, які має вирішувати програмне забезпечення.

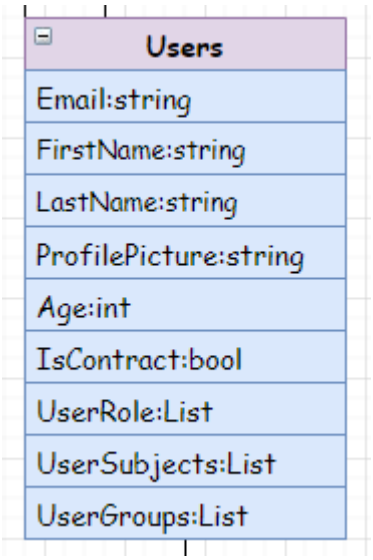
Основні сутності бази даних: Faculty, Groups, Users, Subject, TimeTable.

Проміжні сутності: GroupSubjects, UserGroups, UserSubjects.

Опис всіх сутностей знаходиться в папці Models, яка у свою чергу належить рівню DAL(рівень доступу до даних).

Додаток Є

Сутність **Users**



Має такі поля:

Email – електронна адреса користувача, тип string.

FirstName – ім’я користувача, тип string.

LastName – прізвище користувача, тип string.

ProfilePicture – фото користувача, тип string.

Age – вік користувача, тип int.

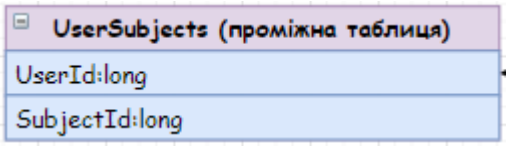
Сутність Users, посилається на інші сутності: Subjects, Groups, Role. Щоб реалізувати дану логіку, було створено три моделі UserSubject, UserGroups, UserRoles. Ці моделі описують зв’язок користувача з іншими сутностями.

Сутність **UserRoles** містить поля Amin, Student, Teacher, типу string.

Вона реалізована для того, щоб надавати користувачеві різний доступ до програми, відповідно до його ролей.

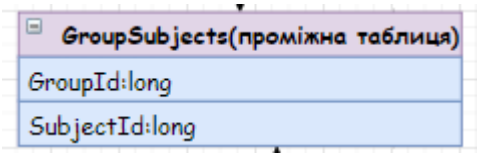
При логіні на сайті буде перевірятись роль юзера і надаватись доступ до певних сторінок. До прикладу, студент матиме доступ лише до сторінки для студента, відповідним чином реалізована логіка і для викладача та адміністратора.

Сутність **UserSubjects**



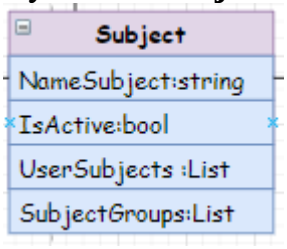
UserSubject зв'язує користувача з предметом, це реалізовано за допомогою зовнішніх ключів UserId та SubjectId. Ця сутність є проміжною сутністю між головними сутностями User і Subject.

Сутність **UserGroups**



UserGroup зв'язує користувача з певною групою, це реалізовано за допомогою зовнішніх ключів UserId та GroupId. Ця сутність є проміжною сутністю між головними сутностями User і Group.

Сутність **Subject**



Має такі поля:

NameSubject – назва предмету, тип string.

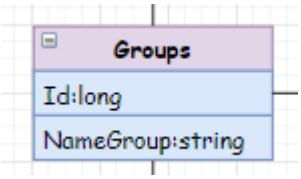
IsActive – показує чи предмет активний, тип bool.

Модель Subject містить проміжні моделі UserSubjects та SubjectGroup.

UserSubject зв’язує користувача і предмет(див. вище).

SubjectGroup зв’язує сутність Subject та Group.

Сутність **Groups**



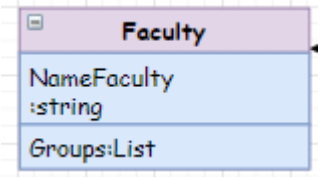
Має такі поля:

Id – первинний ключ, тип long.

NameGroup – назва групи, тип string.

За допомогою сутності Groups реалізовано зв’язок багато до багатьох з Users та Subject. Це означає, що один користувач може належати до різних груп, а одна група може мати безліч користувачів. Один предмет може вивчатись на різних групах, а одна група може мати багато предметів.

Сутність **Faculty**

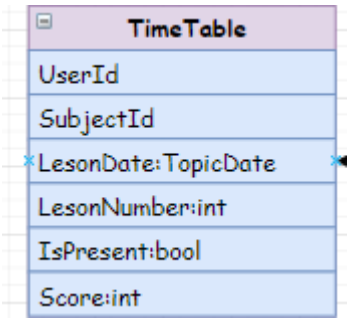


Має такі поля:

NameFaculty – назва факультету.

Ця сутність має зв’язок один до багатьох з сутністю Group. Це означає, що один факультет має різні групи, а одна група може належати лише одному факультету.

Сутність **TimeTable**



Має такі поля:

UserId – зовнішний ключ для зв'язку з сутністю User.

SubjectId – зовнішний ключ для зв'язку з сутністю Subject.

LesonDate – дата заняття, тип TopicDate.

LesonNumber – номер заняття, тип int.

IsPresent – поле, яке показує присутність студента на занятті, тип bool.

Score – бал, виставлений за заняття студенту, тип int.

Реалізована для контролю відвідуваності пар студентами.

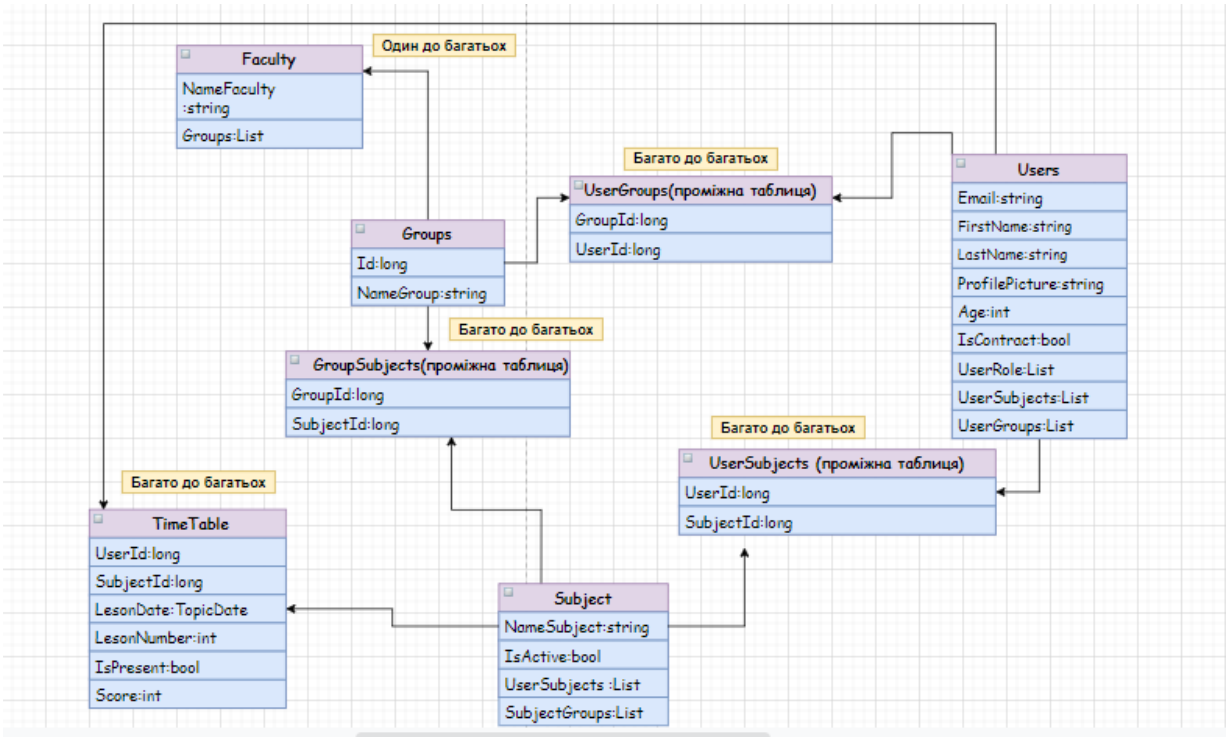


Рис. 4.3. Схема бази даних

РОЗДІЛ 5

АВТОРИЗАЦІЯ ТА АУТЕНТИФІКАЦІЯ

5.1. Аутентифікація

Аутентифікація (від англ. authentic – істинний, справжній) – це надання доказів, що ви ідентифіковані, або інакше кажучи попередньо зареєстровані в системі.

Аутентифікація може здійснюватися різними способами, найпростіший це аутентифікація за паролем.

Цей метод ґрунтується на тому, що користувач повинен надати username та password для успішної ідентифікації та аутентифікації в системі. Пара username/password задається користувачем при його реєстрації в системі, при цьому username – це може бути адреса електронної пошти користувача.

Існують й інші методи аутентифікації:

- Аутентифікація за сертифікатами;
- Аутентифікація за одноразовими паролями;
- Аутентифікація за ключами доступу.

На проекті було вибрано аутентифікацію за токенами. Це ще один вид аутентифікації, який добре підходить для запобігання взломів та забезпечення надійності сайту.

Типовий приклад цього способу – вхід до програми через обліковий запис у соціальних мережах. Тут соціальні мережі є сервісами автентифікації, а програма довіряє функцію автентифікації користувачів соціальним мережам.

Отже, ще однією причиною використання саме цього методу аутентифікації є зручність, адже було використано аутентифікацію за допомогою акаунту користувача в Google.

Є декілька видів токенів : Simple Web Token(SWT), JSON Web Token (JWT), Security Assertion Markup Language (SAML).

Було використано JWT токен.

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який дозволяє безпечно передавати інформацію між сторонами за допомогою JSON об’єкта.[14]

Аутентифікація – це найпоширеніший спосіб використання JWT токенів. Як тільки користувач входить в систему, він отримує JWT токен, який дозволить користувачеві отримати доступ до маршрутів, служб і ресурсів, які дозволені цим токеном.

5.1.1. Структура JWT токена

JWT токен складається з трьох частин, розділених крапками.

Зазвичай це виглядає наступним чином:

xxxxx.yyyyy.zzzzz

- Header
- Payload
- Signature[14]

Header

Header(заголовок) зазвичай складається з двох частин: типу маркера, яким є JWT, і використовуваного алгоритму підписання, наприклад HMAC SHA256 або RSA.

Наприклад:

{

“alg”:”HS256”,

“typ”:”JWT”

}

Потім цей JSON кодується в Base64Url, щоб утворити першу частину JWT.

Payload

Payload (корисне навантаження) є другою частиною токена.

Прикладом корисного навантаження може бути:

{

“sub”:”123456”,

“name”:”John Doe”,

“admin”:”true”,

}

Signature

Signature(підпис) використовується для перевірки того, щоб повідомлення не було змінено на шляху, а у випадку токенів, підписаних за допомогою закритого ключа, він також може перевірити, що відправник JWT є тим, за кого себе видає. Щоб створити частину підпису, потрібно взяти закодований заголовок, закодоване корисне навантаження, секрет, алгоритм, зазначений у заголовку, і підписати все.

Наприклад:

Підпис за допомогою алгоритму HMAC SHA256:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

5.1.2. Як працюють web токени

Під час аутентифікації, коли користувач успішно входить до системи, використовуючи свої облікові дані, буде повернуто веб-токен JSON. Оскільки токени є обліковими даними, потрібно бути обережними, щоб запобігти проблемам безпеки. Як правило, не потрібно зберігати токени довше, ніж зазначено. Також не варто зберігати конфіденційні дані сеансу в сховищі браузера.

Щоразу, коли користувач хоче отримати доступ до захищеного маршруту або ресурсу, користувач повинен відправити JWT, зазвичай в заголовку авторизації, використовуючи схему Bearer.

Вміст заголовка має виглядати так:
Authorization: Bearer <token>

5.2. Авторизація

Авторизація – це перевірка ролі користувача в системі та рішення про надання доступу до запитаної сторінки або ресурсу.
Ключовим інструментом для авторизації є атрибут AuthorizeAttribute із простору імен Microsoft.AspNetCore.Authorization.
При цьому атрибут Authorize не вказує, як користувач повинен автентифікуватись, він тільки виконує перевірку.
На сайті є три сторінки, які потребують авторизуватись, це сторінка Студента, Вчителя та Адміністратора, і залежно від заданої ролі(Student, Teacher, Admin), виконується перевірка доступу до цих сторінок.

На проекті було реалізовано авторизацію та аутентифікацію, Додаток Ж.

РОЗДІЛ 6

ІНТЕРФЕЙС ПРОГРАМИ

Для написання Frontend частини було використано бібліотеку React.
React – це бібліотека JavaScript, яка використовується для створення користувацьких інтерфейсів. React було створено компанією Facebook, а перший реліз випущено у березні 2013 року.
Популярність цієї біблотеки стрімко зростає.
Переваги React:

- Віртуальний DOM.
- Розбиття інтерфейсу на окремі компоненти.
- Має відкритий код.
- Має хороше ком’юніті.
- Використання Jsx.

6.1. Віртуальний DOM

Вся структура веб-сторінки може бути представлена за допомогою DOM (Document Object Model) – організація елементів html, якими ми можемо маніпулювати, змінювати, видаляти чи додавати нові. Для взаємодії з DOM використовується мова JavaScript. Однак, коли ми намагаємося маніпулювати html-елементами за допомогою JavaScript, ми можемо зіткнутися зі зниженням продуктивності, особливо при зміні великої кількості елементів. Але якби ми працювали з JavaScript об'єктами, то операції робилися б швидше.

Для вирішення проблеми продуктивності якраз і виникла концепція віртуального DOM.

Віртуальний DOM є легковажною копією звичайного DOM. І відмінністю React є те, що дана бібліотека працює саме з віртуальним DOM, а не звичайним.

Якщо програмі потрібно дізнатися інформацію про стан елементів, відбувається звернення до віртуального DOM.

Якщо потрібно змінити елементи веб-сторінки, то зміни спочатку вносяться до віртуального DOM, а вже потім новий стан віртуального DOM порівнюється із поточним станом. І якщо ці стани відрізняються, то React знаходить мінімальну кількість маніпуляцій, які потрібні до оновлення реального DOM до нового стану і робить їх.

У результаті така схема взаємодії з елементами веб-сторінки працює набагато швидше і ефективніше, ніж якщо ми працювали б з JavaScript з DOM безпосередньо.[15]

6.2. JSX

JSX(JavaScript Syntax Extension) представляє спосіб опису візуального коду в React за допомогою комбінації коду JavaScript і розмітки XML. JSX не є обов'язковим, тобто весь код можна писати і на чистому Java Script, але використання його рекомендовано.

JSX виступає своєрідним синтаксичним цукром.

Приклад використання JSX:

```
const element = <h1>Hello, world!</h1>;
```

Тут можна побачити, як код HTML присвоєно змінній Java Script.

При роботі з JSX слід враховувати ряд моментів, зокрема в JSX для установки класу застосовується атрибут className, а не class. Другий момент: атрибут style як значення приймає об'єкт Java Script. І третій момент: в JSX використовується camel-case, тобто якщо ми хочемо визначити стильову властивість для шрифту, наприклад,

властивість font-family, то відповідна властивість в об'єкті стилю називатиметься fontFamily , тобто дефіс відкидається, а наступна частина слова починається з великої літери. [16]

JSX створює елементи React.

Весь інтерфейс React дозволяє розбивати на окремі компоненти, їх можна повторно використовувати та легко оновлювати. Компоненти є аналогічними функціями JavaScript. Вони зберігають стан за допомогою властивостей і повертають елементи React, які потім повертаються на веб-сторінці.

Компоненти є двох типів: функціональні та класові. На проекті використовувались функціональні компоненти, що є більш сучасним і рекомендованим способом.

6.3. React Hooks

Хуки з'явились у версіїї React 16.8. Вони дозволяють використовувати стан та інші можливості React без написання класу.

Однією із переваг використання хуків над класовими компонентами є можливість написання функціонального коду та зберігання і зміну стану окремого функціонального компонента.

Стан – це об’єкт, який описує внутрішній стан компонента, він схожий на props за винятком, що стан визначається всередині компонента і доступний тільки із компонента. Стан визначає, як відбувається рендеринг компонентів і як вони поводяться. Одним з основних застосувань стану є те, що він допомагає веб-сайту бути більш динамічним і спрощує взаємодію зі змінними всередині програми.

За допомогою хуків можливо виокремити логіку стану з компонента, щоб протестувати її або повторно використати. Перевикористання логіки відбувається з відслідкуванням стану без зміни ієрархії компонентів. Через це стає легко ділитися хуками поміж іншими компонентами.

Як і більшість програм React до появи React 16.8, інтерфейс користувача був розділений на дві категорії: контейнери і презентаційні компоненти, як запропонував

Ден Абрамов . Контейнер зазвичай складався з безлічі компонентів вищого порядку (HOC) для повторного використання логіки та введення даних. Презентаційні компоненти отримували дані та зворотні виклики через властивості, передані з контейнерів.

Дві поширені проблеми з використанням HOC включають використання властивостей рендерингу та можливість мати компоненти, оточені шарами постачальників, споживачів, компонентів вищого порядку, властивостей рендерингу та інших абстракцій, яких ще називають пеклом обгортки.

У порівнянні з рішенням HOC, підхід Hooks забезпечує більш чітку логіку та кращий спосіб зрозуміти взаємозв'язок між даними та компонентом.[17]

Отже, використання React Hooks дає три переваги: можливість повторного використання, зручність читання і тестування.

6.4. Redux

Redux – відкрита JS бібліотека призначена для управління станом програм JavaScript. Найчастіше використовується разом з React або Angular для побудови інтерфейсів користувача. Вперше він був представлений Деном Абрамовим та Ендрю Кларком у 2015 році.

6.4.1. Основні концепції Redux:

Незмінне дерево станів.

У Redux загальний стан програми представлено одним об'єктом JavaScript – state (стан) або state tree (дерево станів). Незмінне дерево станів доступне лише для читання, змінити нічого безпосередньо не можна. Всі зміни можливі лише при надсиланні action (дії).

Дія (action) – це JavaScript-об'єкт, який лаконічно описує суть зміни. Єдина вимога до об'єкта дії – це наявність властивості type, значенням якого є рядок.

Генератори дій (actions creators) – це функції, що створюють дії. Зазвичай ініціюються разом із функцією відправлення дії(dispatch).

Редуктор (reducer) – це чиста функція, яка обчислює наступний стан дерева на підставі його попереднього стану та дії, що застосовується.

(currentState, action) => newState

Редуктор працює незалежно від стану програми, та повертає новий об'єкт дерева станів, яким замінюється попередній.

Сховище (store) – це об'єкт, який:

- містить стан програми;
- відображає стан через getState();
- може оновлювати стан через dispatch();
- дозволяє реєструватися (або видалятися) як слухач зміни стану через subscribe().

Сховище у додатку завжди унікальне.[18]

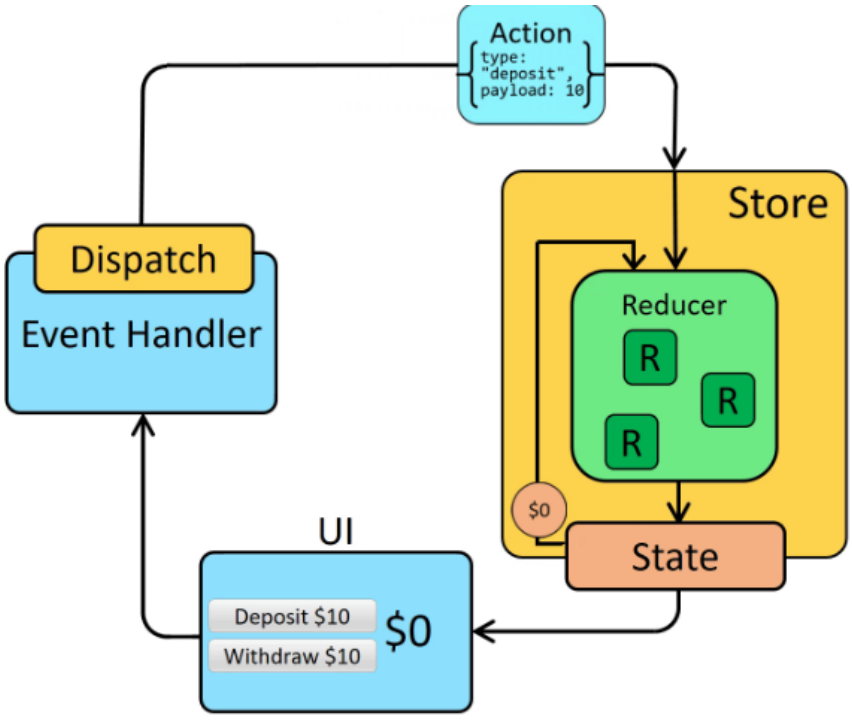


Рис. 6.1. Принципи роботи redux

На проекті реалізовано чотири редуктори:

– Admin-reducer

- Auth-reducer
- Profile-reducer
- Teacher-reducer

Всі вони зберігаються в єдиному сховищі (store), це реалізовано за допомогою функції combineReducers, яку надає сам Redux. Ця функція приймає об’єкт, де ключі це назви всіх редукторів, а значення, це стани редукторів.

Додаток З

6.4.2. Переваги Redux

Redux допомагає легко управляти станами компонента, і доступатися до стану будь-якого компонента з будь-якого місця, навідмінну від того, як це відбувається в React.

В React ми працюєм зі станом через props, і можемо передати дані лише від батьківського компонента до дочірнього.

Props (скорочення від властивості) – це конфігурація компонента, його параметри. Вони отримані згори і незмінні , що стосується компонента, який їх отримує.

Компонент не може змінювати свої властивості, але він відповідає за об’єднання властивостей своїх дочірніх компонентів.[19]

Такий підхід ще називають prop drilling.

Проп дрилінг це такий анти-паттерн при якому передача пропсів відбувається через проміжні компоненти які не використовують пропси, що отримуються, а тільки передають їх в наступні компоненти.

Проблему проп дрилінгу вирішує Redux. Redux дозволяє обмінюватися даними між різними частинами дерева компонентів.

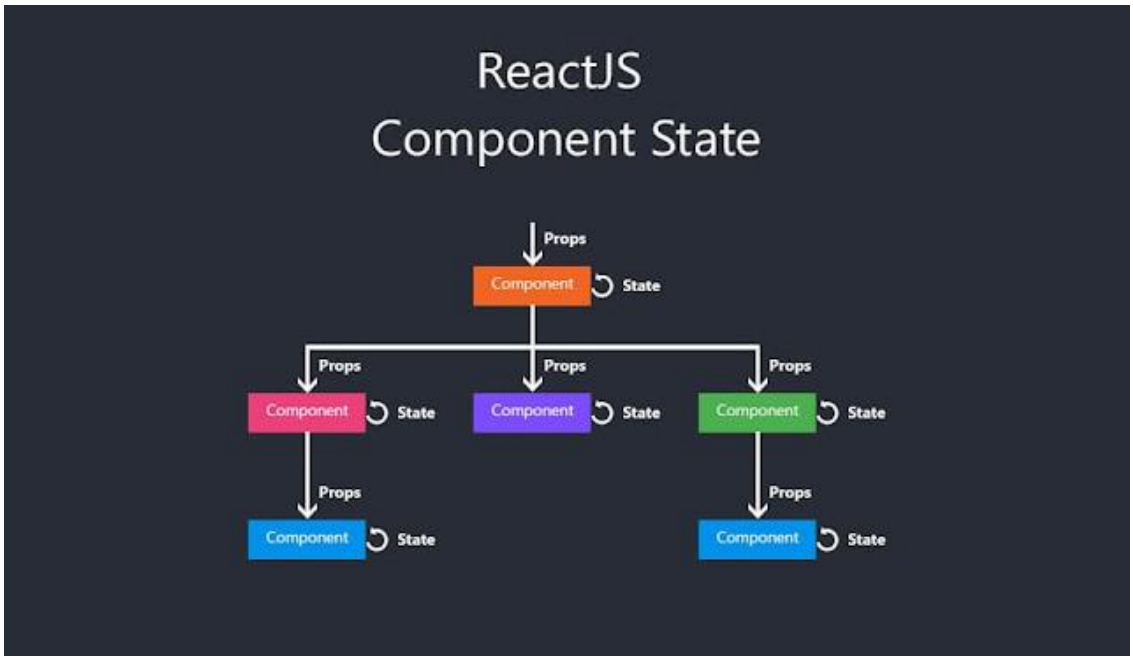


Рис. 6.2. Передача стану у React за допомогою props

Redux найбільш корисний у випадках, коли:

- У вас є велика кількість станів програми, які потрібні в багатьох місцях програми
- Стан програми часто оновлюється
- Логіка оновлення цього стану може бути складною
- Програма має кодову базу середнього або великого розміру, і над нею може працювати багато людей
- Вам потрібно побачити, як цей стан оновлюється з часом[20]

6.4.3. Коли не варто використовувати Redux

Програми, які складаються в основному з простих змін інтерфейсу користувача, найчастіше не вимагають складного шаблону, такого як Redux. Іноді спільне використання стану між різними компонентами також працює та підвищує зручність підтримки вашого коду.

Крім того, користувачі можуть не використовувати Redux, якщо їх дані надходять із одного джерела даних для кожного представлення. Іншими словами, якщо нам не

потрібні дані з кількох джерел, не потрібно використовувати Redux. В такому випадку не буде проблемами неузгодженості даних при доступі до одного джерела даних для кожного представлення.

Незважаючи на те, що Redux досить ефективний шаблон, що використовує чисті функції, він може бути накладним для простих програм, які вимагають лише пари змін інтерфейсу користувача. Крім того, ми не повинні забувати, що Redux це сховище станів у пам'яті. Іншими словами, якщо наша програма дає збій, ми втрачаємо весь стан нашої програми. Це означає, що ми повинні використовувати рішення для кешування, для створення резервної копії стану нашої програми, що знову створює додаткові витрати.[21]

6.5. Axios

Для взаємодії серверної частини з клієнтською потрібно було використати інструмент для створення HTTP-запитів, ми зупинилися на одному з найкращих рішень, бібліотеці Axios.

Axios – популярна бібліотека JavaScript, яка використовується для виконання HTTP-запитів, вона працює як у браузері, так і на інших платформах. Підтримує всі сучасні браузери, включаючи підтримку IE8 та вище. Заснована на обіцянках, і це дозволяє нам писати код async / await для виконання XHR запитів дуже легко.

Використання Axios має ряд переваг:

- підтримує старі браузери (для Fetch потрібен поліфіл);
- є спосіб перервати запит;
- є спосіб встановити час очікування відповідь;
- має вбудований захист від CSRF;
- підтримує прогрес завантаження;
- виконує автоматичне перетворення даних JSON;

6.6. Ant Design

Ant Design – це повноцінна дизайн система з готовими React UI компонентами. Дану систему було вибрано у зв'язку з очевидними її перевагами – не потрібно розробляти дизайн окремих UI компонентів, та верстати їх, адже існують вже готові рішення. Різноманіття компонентів є досить великим, що дозволяє побудувати майже будь-який інтерфейс без особливих затрат.

Це допомогло зекономити значну частину часу, виділеного на проект.

Ще однією перевагою, є те, що всі UI компоненти дуже легко використовувати, наприклад кнопку зі стилями можна задати всього однією стрічкою.

Варто також відмітити досить хорошу та зручну документацію та багатий список прикладів з готовим кодом.

Ще однією перевагою цієї бібліотеки є те, що приклади є як на Java Script, так і на Type Script. Мова програмування Type Script набирає значної популярності, так як дозволяє задавати типи змінним, що забезпечує запобіганню помилок ще на етапі розробки. Тому Ant Design є одним із найкращих рішень, коли на проекті використовується Type Script.

Приклад використання Ant Design:

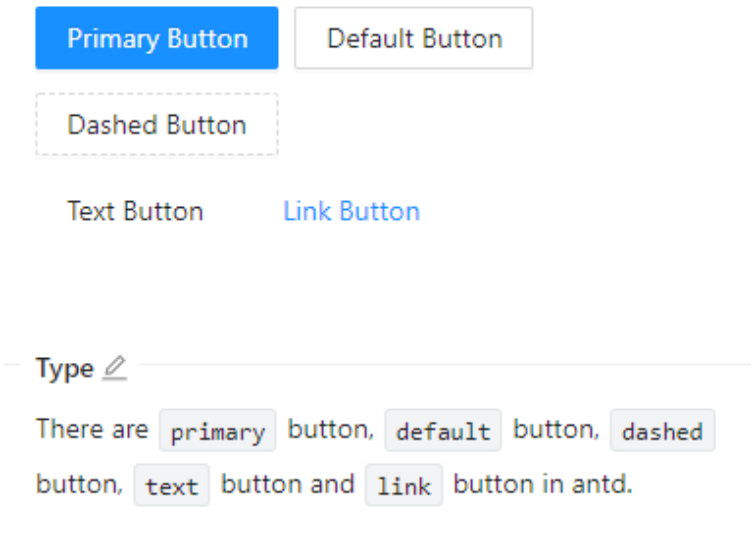


Рис. 6.4 Готові UI-компоненти дизайн системи Ant Design

Використаний код:

```
import { Button } from 'antd';

ReactDOM.render(
  <>
    <Button type="primary">Primary Button</Button>
    <Button>Default Button</Button>
    <Button type="dashed">Dashed Button</Button>
    <br />
    <Button type="text">Text Button</Button>
    <Button type="link">Link Button</Button>
  </>,
  mountNode,
);
```

Але основною превагою Ant Design над іншими дизайн системами є таблиці та форми. В Ant Design ці компоненти, реалізовано з максимально можливим та необхідним функціоналом, який не потрібно дописувати самостійно.

У таблиць налаштована пагінація, фільтрація та сортування. Ще однією корисною властивістю є можливість редагувати клітинки в таблиці та вносити свої дані. Такий функціонал було використано на проекті, зокрема на сторінці викладача, де він може виставляти бали студенту за заняття.

З недоліків даної дизайн системи, можна відмітити не дуже хорошу адаптацію під мобільні пристрої, для цього використовують Ant Design Mobile. Також варто зазначити, що Ant Design більше підходить для проектів, які не потребують унікальних дизайнерських рішень, але з іншої сторони ця система має майже усі необхідні готові UI компоненти, що значно дозволяє заощадити час при розробці.[22]

6.7. Приватні роути, React router dom

React Router – це повнофункціональна бібліотека маршрутизації на стороні клієнта та сервера для React, бібліотеки JavaScript для створення інтерфейсів користувача. React Router працює скрізь, де працює React, в інтернеті, на сервері з node.js і на React Native.[23]

Для роботи з маршрутизацією на проекті використано пакет react router dom v6.

Ця версія є новою, і має ряд відмінностей та переваг над старішими версіями цього ж пакету.

Основні відмінності react router dom v6 з react router dom v5:

Switch → Routes

Замість компонента Switch з'явився компонент Routes. Routes є більш функціональний. Основна відмінність полягає в тому, що Routes не вимагає жорсткого порядку роутів усередині.

Switch обходив роути в строгому порядку зверху вниз і при першому збігу рендерив заданий компонент. Тому було важливо зберігати та контролювати певний порядок роутів, без цього вони працювали некоректно.

Відносні маршрути

Раніше потрібно було задавати повний маршрут в пропс path .

Наприклад:

```
<Route path="app/login" component={Login } />
```

В версіє v6 буде достатньо вказати відносний маршрут:

```
<Route path="/login" element={<Login />} />
```

Element замість Component/render

Це дуже зручна функція, яка з’явилась у v6. Раніше в компоненті Route був вибір: або вказати компонент для рендера, або передати render-prop функцію. В першому випадку код виглядав акуратно:

```
<Route path={urls.courses} component={CoursesList} />
```

Але був один недолік. Не можна було задати додаткові пропси в компонент. Для цього використовувались render-функції:

```
<Route path={urls.courses} render={props => <CoursesList {...props} otherProp={myProp} />} />
```

В новій версії роутера ці пропси було замінено на один – element. В нього можна передати будь-який JSX-елемент. Наприклад:

```
<Route path={urls.courses} element={<CoursesList otherProp={myProp} />} />
```

Це набагато зручніше, та робить код чистішим.

Для роботи з історією використовується useNavigate замість useHistory.

На проекті реалізовано також приватні роути, і залежно від ролі користувача, буде відображатись контент, якщо користувач не має жодної з ролей (студент, викладач або адміністратор), його буде перенаправляти на сторінку з авторизацією. Для перенаправлення на іншу сторінку використовується компонент Navigate з react router dom v6: <Navigate to="/login" />.

```
import React from 'react';
import { Navigate } from 'react-router-dom';

const StudentPrivateRoute = ({ children, roles }) => {
  const checkRole = roles.includes('Student');
  return checkRole ? children : <Navigate to="/login" />;
};

export default StudentPrivateRoute;
```

```
<Route
  path="/"
  element={
    <StudentPrivateRoute roles={roles}>
    <Main />
    </StudentPrivateRoute>
  }
/>
```

Рис.6.5. Реалізація приватного роута для сторінки студента

6.8. Модульні CSS стилі в React.

На проекті було використано модульні css стилі.

Модуль CSS – це файл .css, у якому всі імена класів та анімацій мають локальну область видимості за умовчанням, класи діють подібно до локальних змінних у Java Script. Це інструмент, який робить кожен клас унікальним, включаючи хеш в його назву.

Такий підхід для задавання стилів виправданий самою концепцією React.

У зв’язку з тим, що весь інтерфейс розділяється на окремі незалежні компоненти, стилі для них також мають бути незалежними, тобто в якійсь мірі ізольованими, для запобігання побічних ефектів в іншій частині інтерфейсу.

Для цього були створені різні методології та правила, наприклад методологія БЕМ. Але і вона має ряд недоліків, один з яких це можливі синтаксичні помилки при заданні імен класам.

БЕМ (Блок, Елемент, Модифікатор) – компонентний підхід до веб-розробки. У його основі лежить принцип поділу інтерфейсу на незалежні блоки. Він дозволяє легко та швидко розробляти інтерфейси будь-якої складності та повторно використовувати існуючий код, уникаючи «Copy-Paste»[24]

Переваги використання модульних стилів CSS:

- модульні та повторно використовувані компоненти без побічних ефектів,
- більш чистий CSS код,
- уникнення монолітних CSS файлів, оскільки кожен компонент має свій файл зі стилями.

До недоліків можна віднести:

- складніше читати та розуміти DOM,
- потрібне деяке початкове налаштування оточення, щоб змусити все це працювати.

CSS Modules використовуються разом зі збіркою всього фронтенду, з підтримкою в браузері немає жодних проблем. Браузери отримують звичайний CSS від сервера, тому немає жодної можливості «зламати» сайт, використовуючи CSS Modules. Навпаки, при цьому лише підвищується його надійність, зменшується кількість потенційних багів. Webpack із завантажувачами, налаштованими для підтримки CSS Modules, не створює жодних проблем, так що, без сумніву, можна рекомендувати цей інструмент до використання.

Приклад використання модульних стилів на проекті для компонента Login:

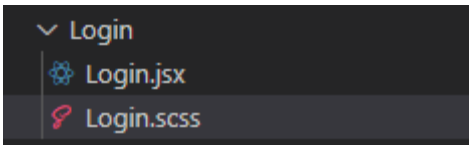


Рис. 6.6. Структура компонента Login

```jsx
const Login = (props) => {
  const navigate = useNavigate();

  const responseGoogle = (res) => {
    props.SetUserData(res.tokenId, navigate);
  };

  return (
    <div>
      <Row className="login-wraper" gutter={[16, 16]} justify="center" align="middle">
        <Col className="login-content" span={6}>
          <img className="logo" src="/images/logo.png" alt="Logo" />
          <GoogleLogin
            clientId="83091282364-d1ckabm35sr7p8lgl5marstr2r5sgia7.apps.googleusercontent.com"
            buttonText="Вхід через пошту @oa.edu.ua"
            onSuccess={responseGoogle}
            onFailure={responseGoogle}
            className="google"
          />
        </Col>
      </Row>
    </div>
  );
};
```

Рис. 6.7. Компонент Login.jsx

```scss
.login-wraper {
  height: 100vh;
}

.logo {
  margin-bottom: 30px;
  width: 300px;
  height: 300px;
}

.login-content {
  text-align: center;
}
```

Рис. 6.8. Модульні scss стилі для компонента Login

РОЗДІЛ 7

РЕЗУЛЬТАТ РОБОТИ НАД ІНТЕРФЕЙСОМ

7.1. Макети з Figma

Робота над інтерфейсом програмного продукту розпочиналася у Figma. Спочатку розроблялися чорно-білі прототипи сторінок. Завдяки цьому можна було досить зручно підбирати всі можливі варіанти для майбутнього макету сторінки, і не витрачати досить багато часу на деталізацію. Далі підбиралася основна кольорова палітра для сайту, та малювалися основні UI компоненти на основі бібліотеки Ant Design за її дизайн принципами. Готові макети використовувалися для верстки сторінок. У такому разі робота виконувалась швидше, і була більш гнучкою на кожному з етапів. У будь який момент можна було внести правки в дизайн, так як Figma дозволяє працювати над макетом віддалено.

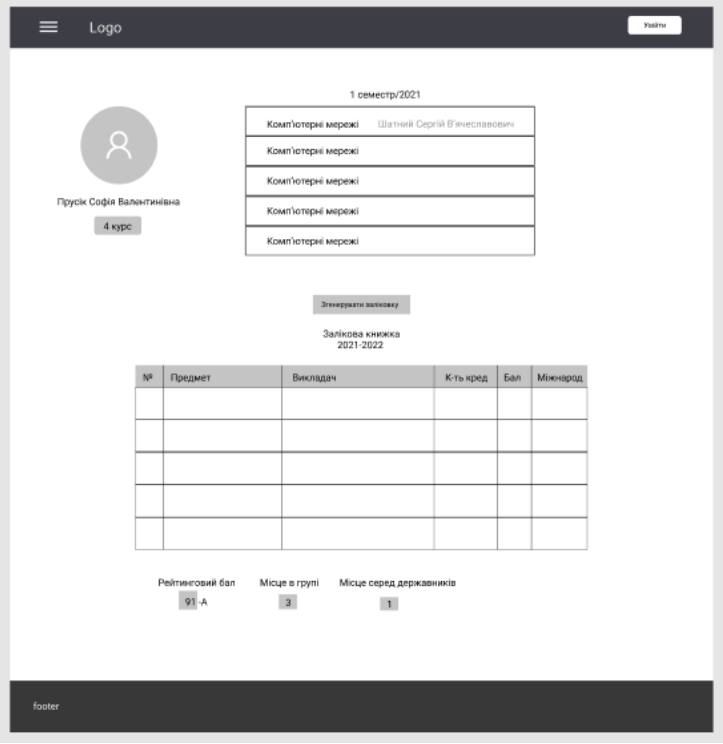


Рис. 7.1.1 Прототип макета для сторінки студента

Рис. 7.1.2 Основна кольорова палітра для сайту

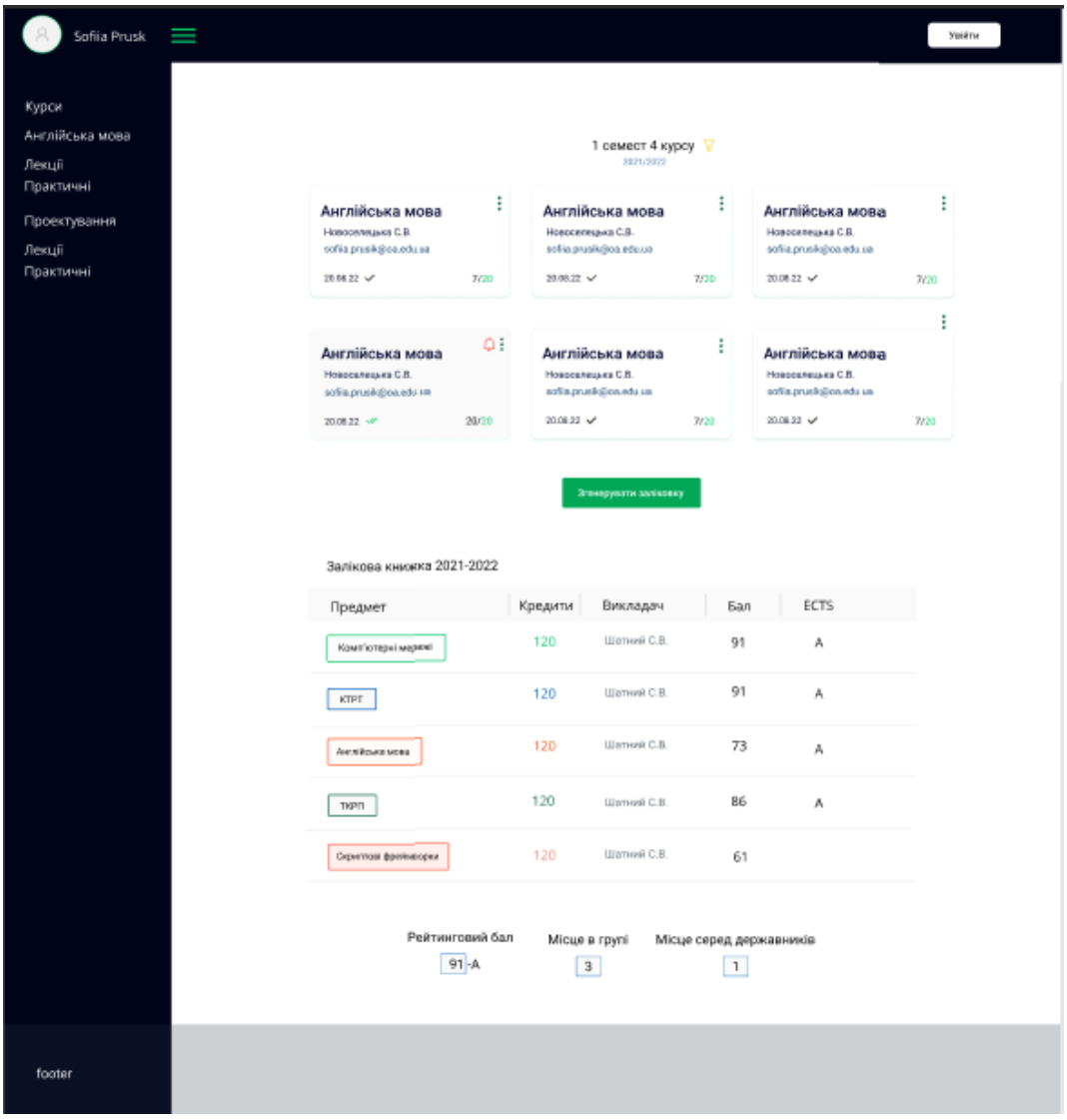


Рис. 7.1.3 Макет сторінки для студента

7.2. Авторизація

На сайті реалізовано авторизацію через корпоративну пошту oa.edu.ua. Така авторизація вже існує в університеті НаУОА, даний підхід є звичним та зрозумілим, що є хорошим плюсом у подальшому користуванні сайтом.

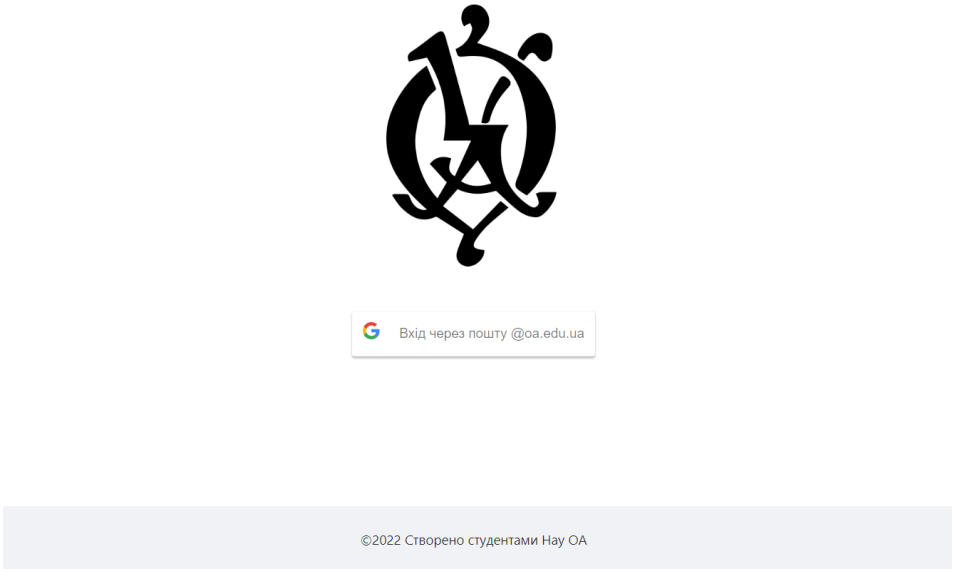


Рис. 7.2.1 Авторизація через корпоративну пошту

7.3. Сторінка студента

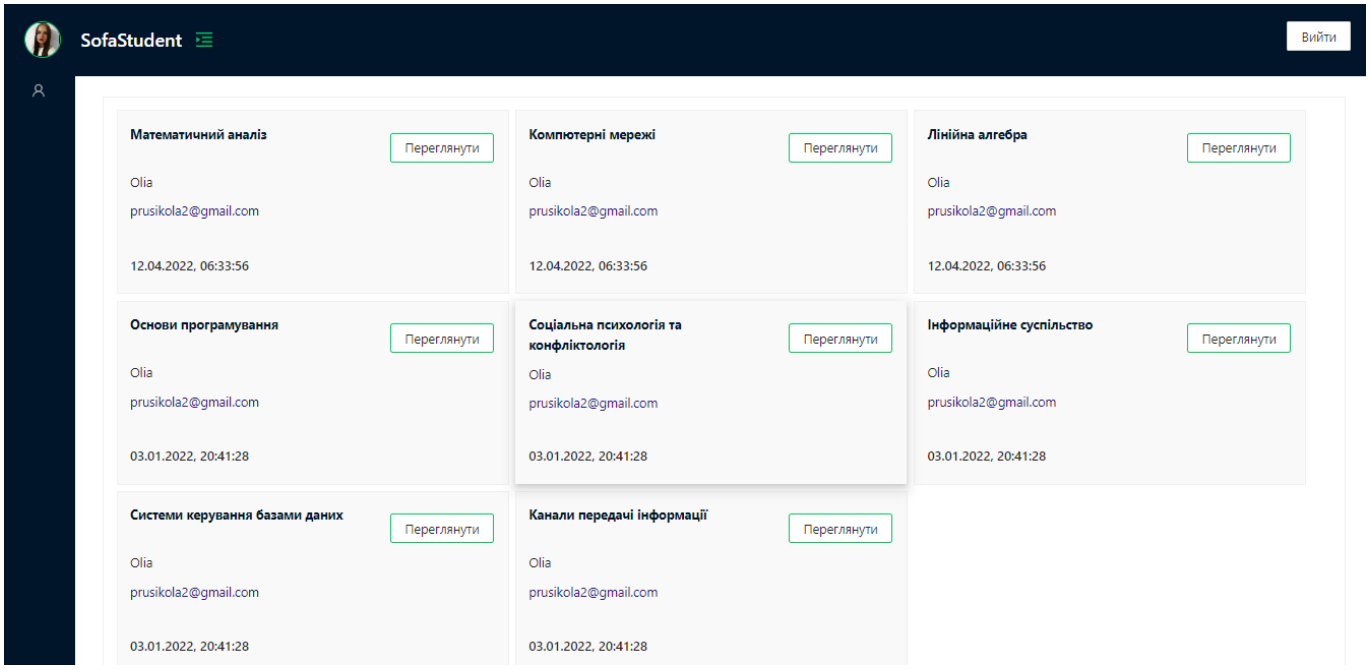


Рис. 7.3.1 Сторінка студента

На сторінці студент може бачити поточні курси. Всі курси представлені у вигляді карток, на яких студент може бачити дату закінчення курсу, скільки пар уже пройдено, ім’я та електронну пошту викладача курсу.

При переході на предмет в студента показується статистика за даним курсом.

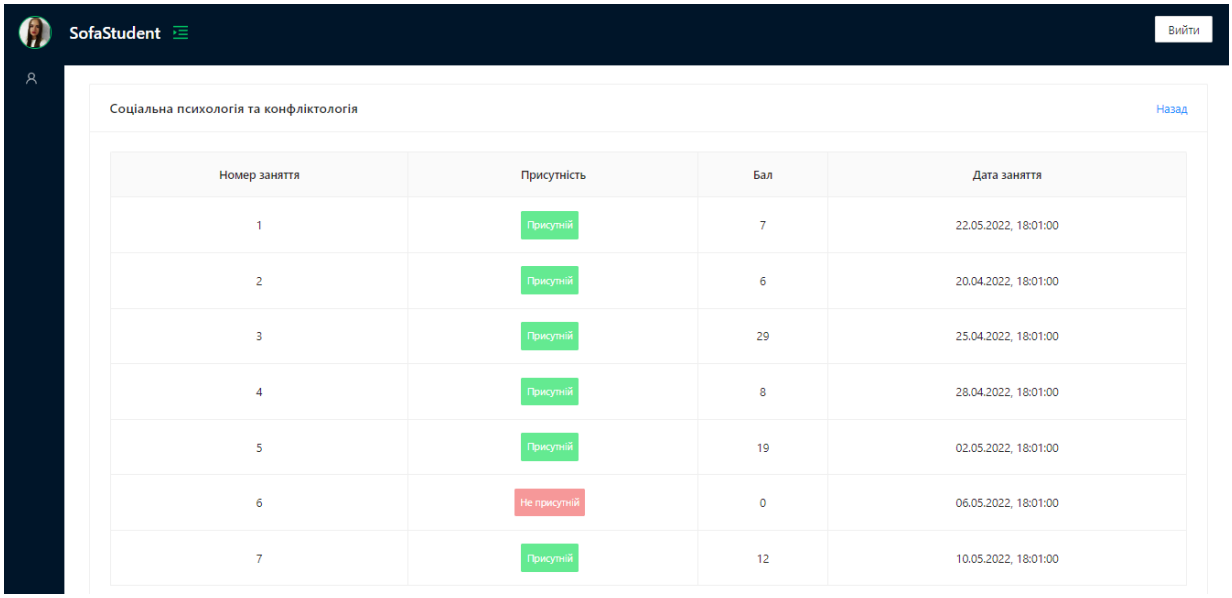


Рис. 7.3.2 Статистика успішності студента за курсом Математичний аналіз

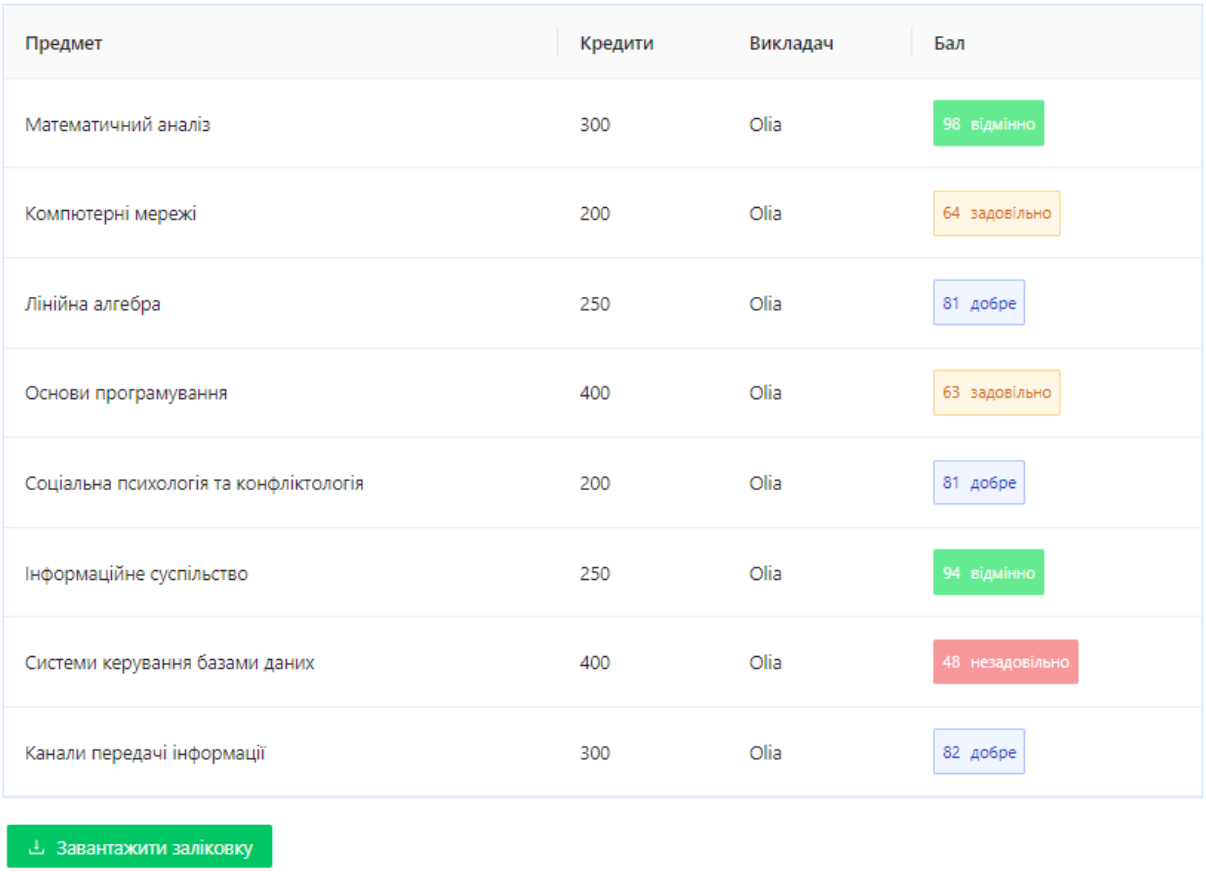


Рис. 7.3.3 Електронна залікова книжка

В кінці семестру для кожного студента формується електронна залікова книжка у вигляді таблиці.

Особливістю даної заліковки є візуальне представлення поточної оцінки у різних кольорах, залежно від набраного балу у міжнародній системі ECTS.

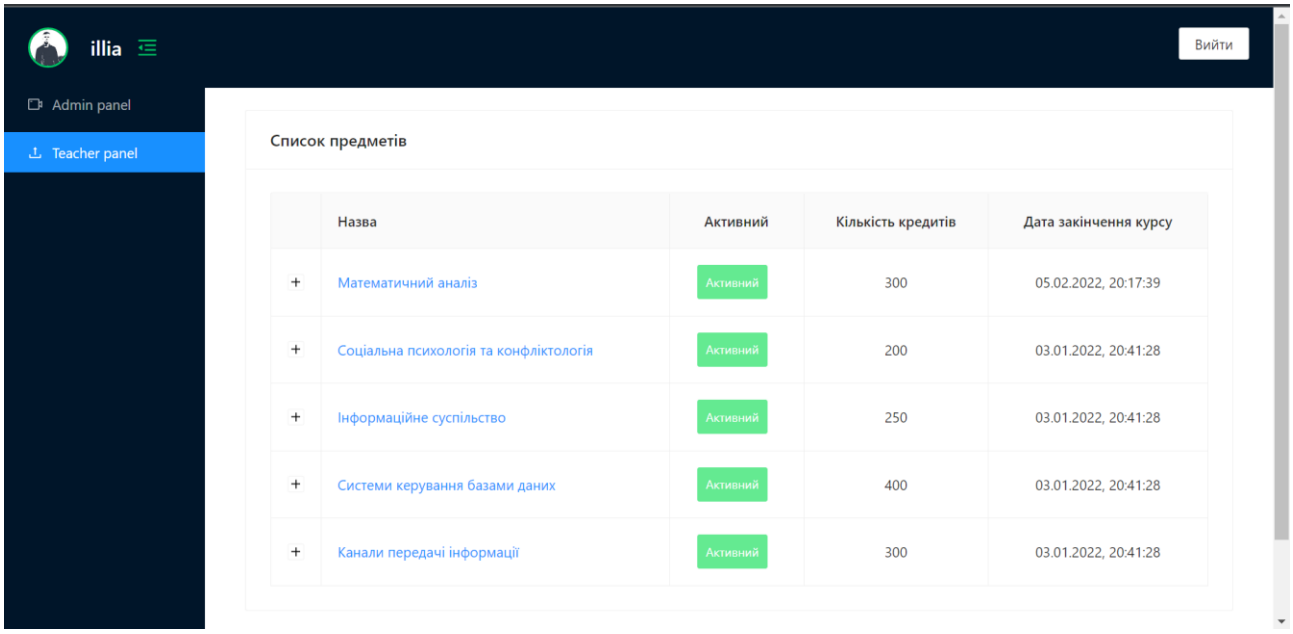
Таким чином, з точки зору психології було сформовано гіпотезу, що це більше мотивуватиме студентів вчитись, аби в заліковці не було червоного. Так як червоний колір асоціюється з незадовільною оцінкою, і в даному ключі викликатиме негативні емоції, зелений навпаки асоціюється з успіхом, з пройденим етапом.

Також студент може скачати заліковку у вигляді звіту Excel.

1	Предмет	Номер заняття	Присутність	Бал	Дата заняття
2	Математичний аналіз	1	Присутній	0	22.05.2022 18:01
3	Компютерні мережі	1	Присутній	2	22.05.2022 18:01
4	Лінійна алгебра	1	Присутній	5	22.05.2022 18:01
5	Основи програмування	1	Відсутній	0	22.05.2022 18:01
6	Соціальна психологія та конфліктологія	1	Присутній	7	22.05.2022 18:01
7	Інформаційне суспільство	1	Відсутній	0	22.05.2022 18:01
8	Системи керування базами даних	1	Присутній	1	22.05.2022 18:01
9	Канали передачі інформації	1	Відсутній	0	22.05.2022 18:01
10	Математичний аналіз	2	Присутній	11	03.04.2022 8:58
11	Математичний аналіз	3	Присутній	5	05.04.2022 8:58
12	Математичний аналіз	4	Присутній	1	08.04.2022 8:58
13	Математичний аналіз	5	Присутній	7	10.04.2022 8:59
14	Математичний аналіз	6	Присутній	11	14.04.2022 8:59
15	Математичний аналіз	7	Присутній	14	20.04.2022 8:59
16	Математичний аналіз	8	Присутній	9	02.05.2022 8:59
17	Математичний аналіз	9	Присутній	0	10.05.2022 9:00
18	Математичний аналіз	10	Присутній	40	15.05.2022 9:00

Рис.7.3.4 Звіт про успішність окремого студента

7.4. Сторінка викладача



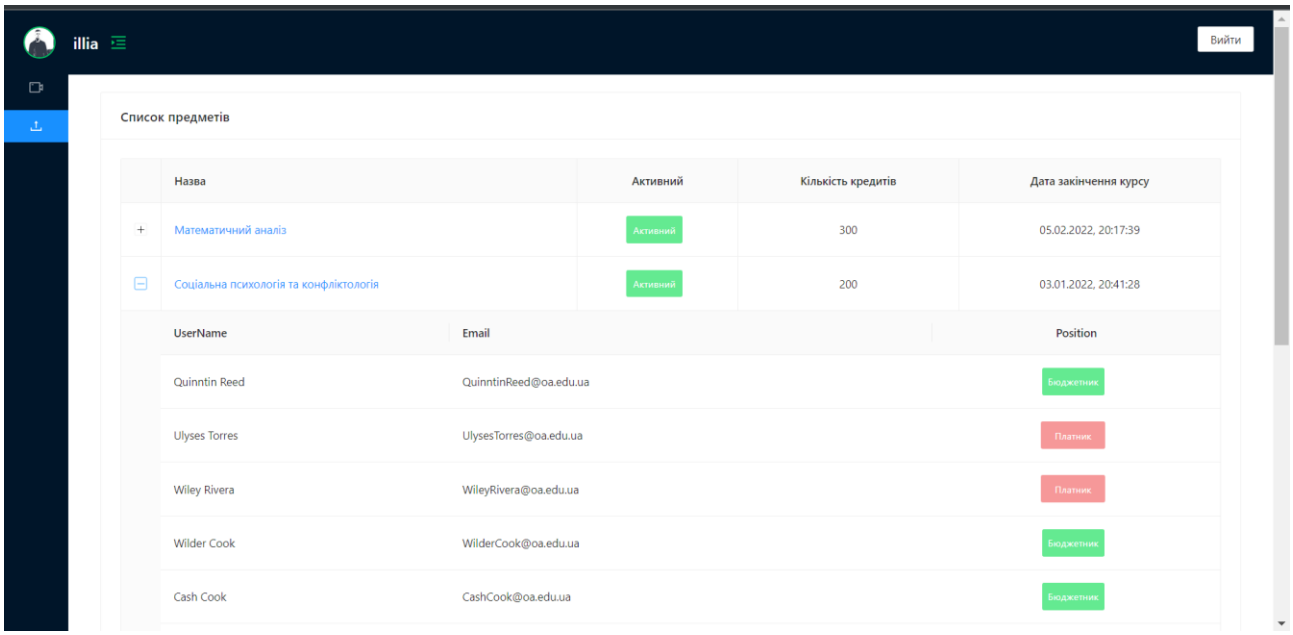


Рис. 7.4.1 Сторінка викладача

На сторінці викладач бачить всі свої поточні курси.

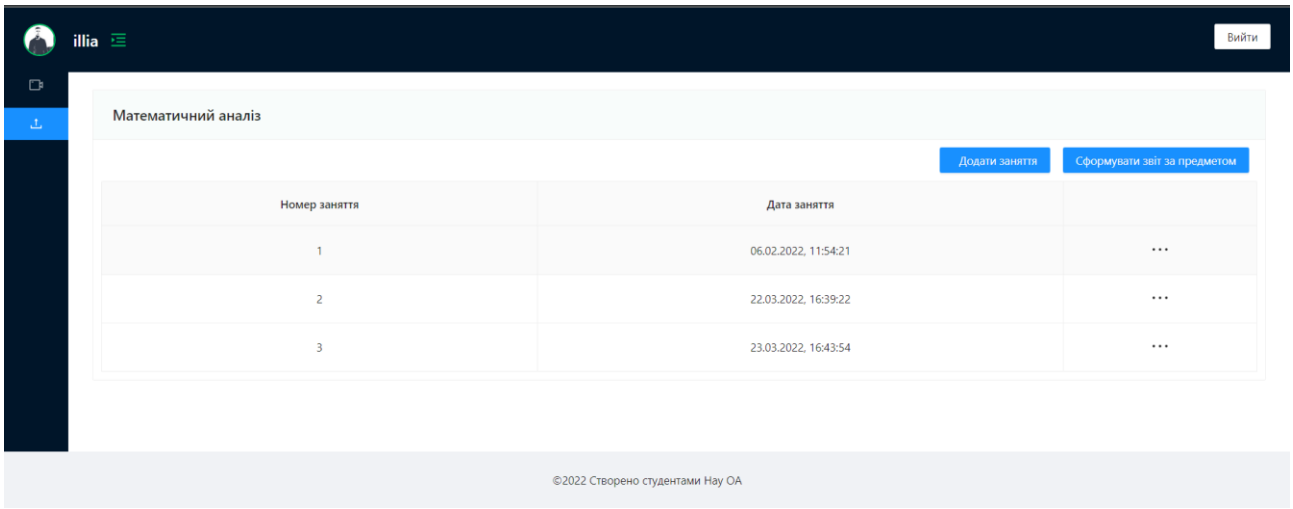


Рис. 7.4.2. Курс Математичний аналіз

Викладач переходити на сторінку з курсом, де може додати нове заняття, вказавши номер та вибравши дату заняття, для цього потрібно натиснути на кнопку «Додати заняття».

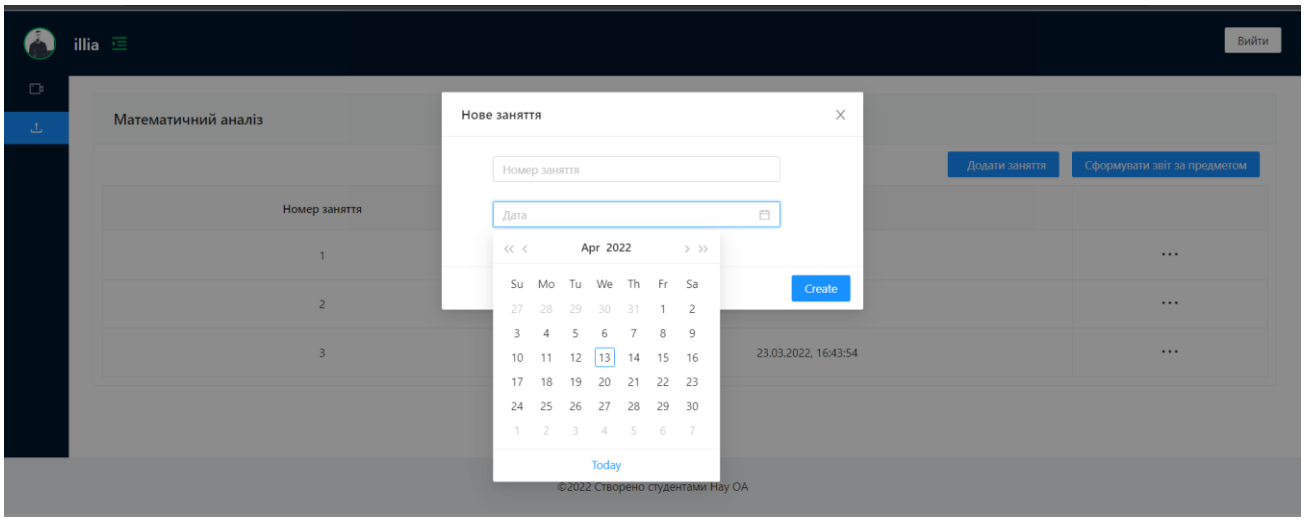


Рис. 7.4.3 Створення нового заняття

Далі викладач може перейти на це заняття, є також функція видалити заняття, у випадку якщо воно було створено неправильно.

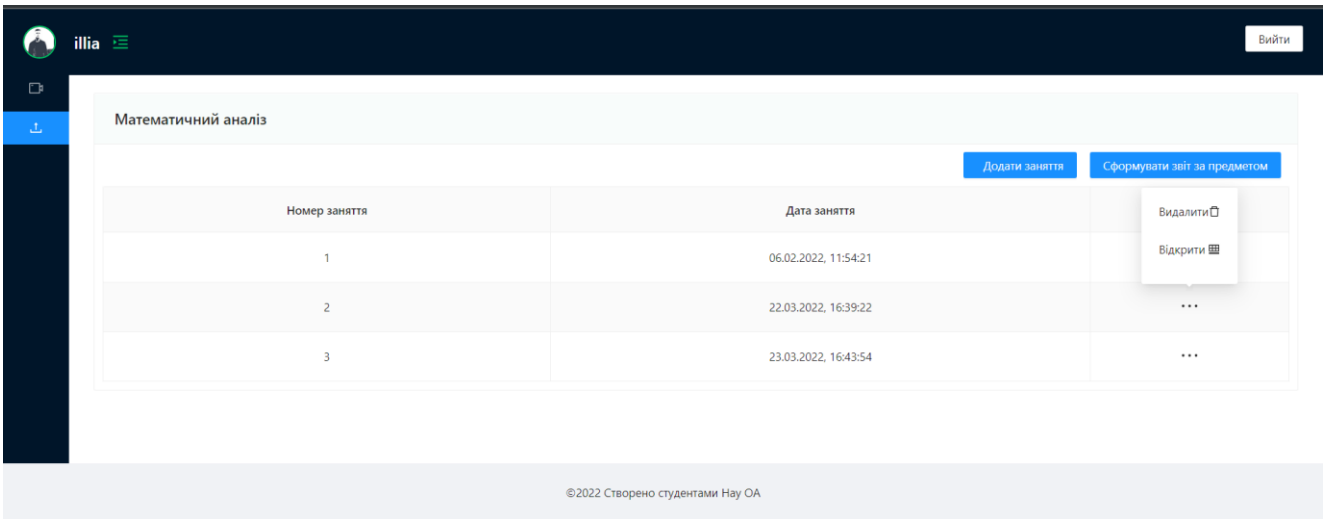


Рис. 7.4.4 Можливість видалення заняття

При переході на заняття з’являється таблиця з групою студентів, підписаних на даний курс.

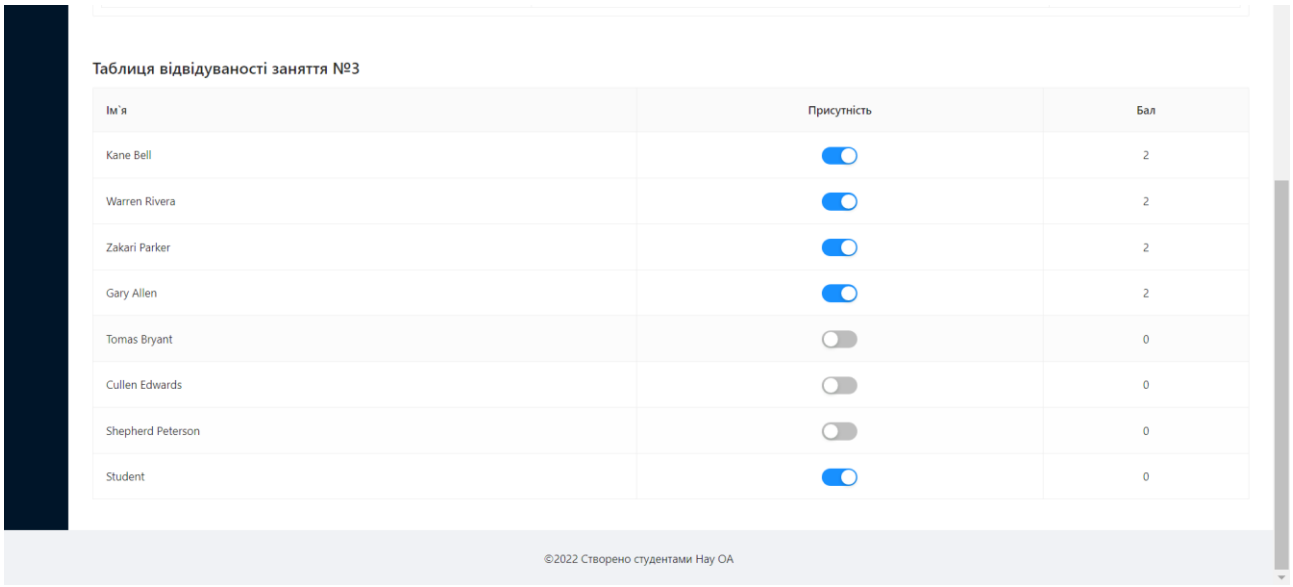


Рис. 7.4.5 Таблиця відвідуваності заняття

За допомогою слайдера викладач перевіряє присутність студентів, а в наступній клітинці може виставляти студенту бал за дане заняття.

Викладач може завантажити звіт в форматі Excel по успішності та відвідуваності на курсі, натиснувши кнопку «Сформувати звіт за предметом».

Основи програмування					
I'мя	**Email**	**Номер за**	**Присутніс**	**Бал**	**Дата заняття**
Kane Bell	KaneBell@oa.edu.ua	1	Відсутній	2	06.02.2022 11:54
Tomas Bryant	TomasBryant@oa.edu.ua	1	Відсутній	4	06.02.2022 11:54
Cullen Edwards	CullenEdwards@oa.edu.ua	1	Присутній	2	06.02.2022 11:54
Shepherd Peterson	ShepherdPeterson@oa.edu	1	Присутній	4	06.02.2022 11:54
Warren Rivera	WarrenRivera@oa.edu.ua	1	Відсутній	2	06.02.2022 11:54
Zakari Parker	ZakariParker@oa.edu.ua	1	Відсутній	2	06.02.2022 11:54
Gary Allen	GaryAllen@oa.edu.ua	1	Відсутній	0	06.02.2022 11:54
Kane Bell	KaneBell@oa.edu.ua	2	Відсутній	5	28.02.2022 20:00
Tomas Bryant	TomasBryant@oa.edu.ua	2	Відсутній	2	28.02.2022 20:00
Cullen Edwards	CullenEdwards@oa.edu.ua	2	Відсутній	2	28.02.2022 20:00
Shepherd Peterson	ShepherdPeterson@oa.edu	2	Відсутній	2	28.02.2022 20:00
Warren Rivera	WarrenRivera@oa.edu.ua	2	Відсутній	2	28.02.2022 20:00
Zakari Parker	ZakariParker@oa.edu.ua	2	Відсутній	2	28.02.2022 20:00
Gary Allen	GaryAllen@oa.edu.ua	2	Відсутній	2	28.02.2022 20:00
Kane Bell	KaneBell@oa.edu.ua	3	Присутній	2	30.03.2022 15:17
Tomas Bryant	TomasBryant@oa.edu.ua	3	Відсутній	0	30.03.2022 15:17
Cullen Edwards	CullenEdwards@oa.edu.ua	3	Відсутній	0	30.03.2022 15:17
Shepherd Peterson	ShepherdPeterson@oa.edu	3	Відсутній	0	30.03.2022 15:17
Warren Rivera	WarrenRivera@oa.edu.ua	3	Присутній	2	30.03.2022 15:17
Zakari Parker	ZakariParker@oa.edu.ua	3	Присутній	2	30.03.2022 15:17
Gary Allen	GaryAllen@oa.edu.ua	3	Присутній	2	30.03.2022 15:17

Рис.7.4.6 Звіт з курсу Основи програмування

7.5. Сторінка адміністратора

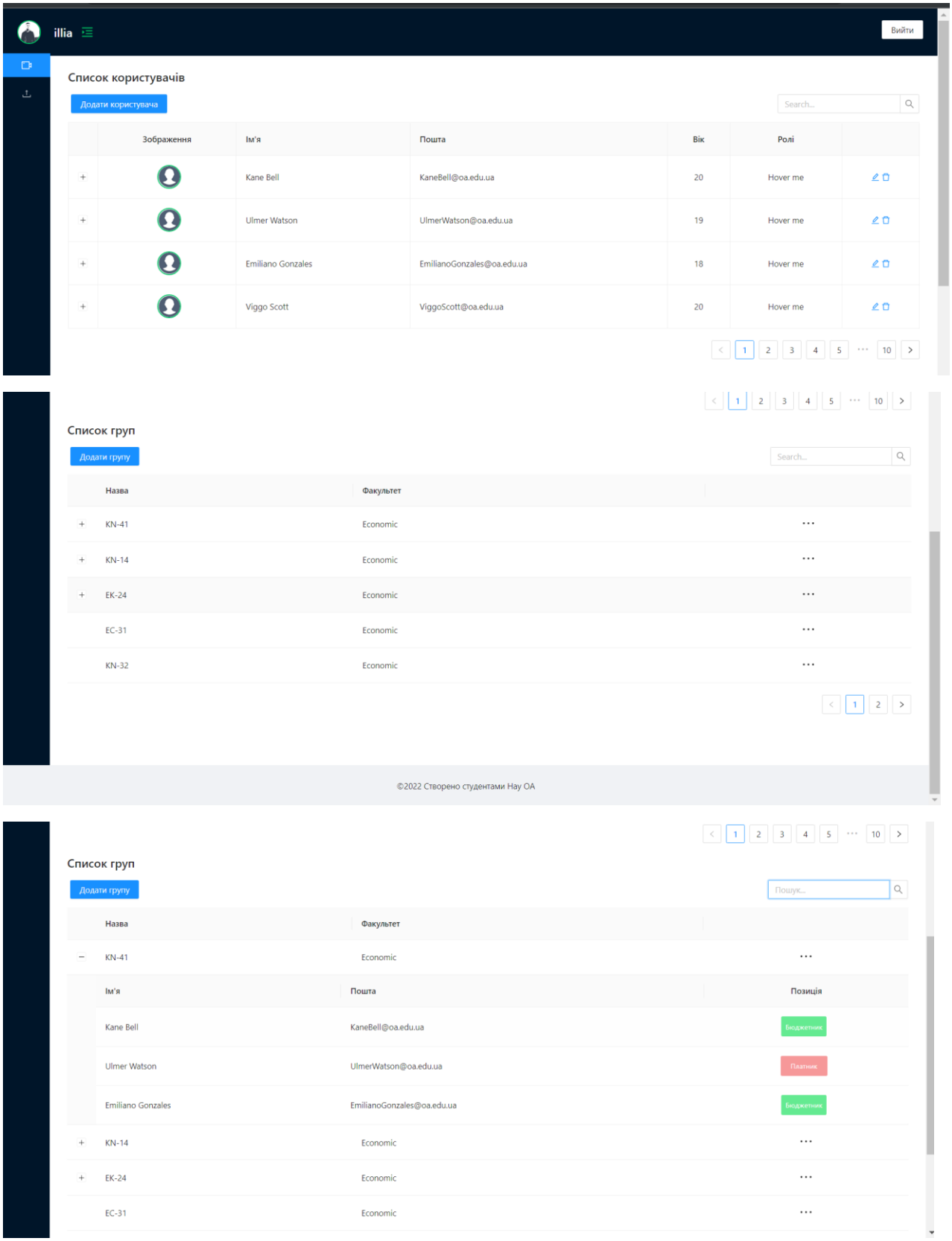


Рис. 7.5.1 Сторінка адміністратора

Адміністратор керує групами та користувачами. Адміністратор може додавати, редагувати та видаляти користувача.

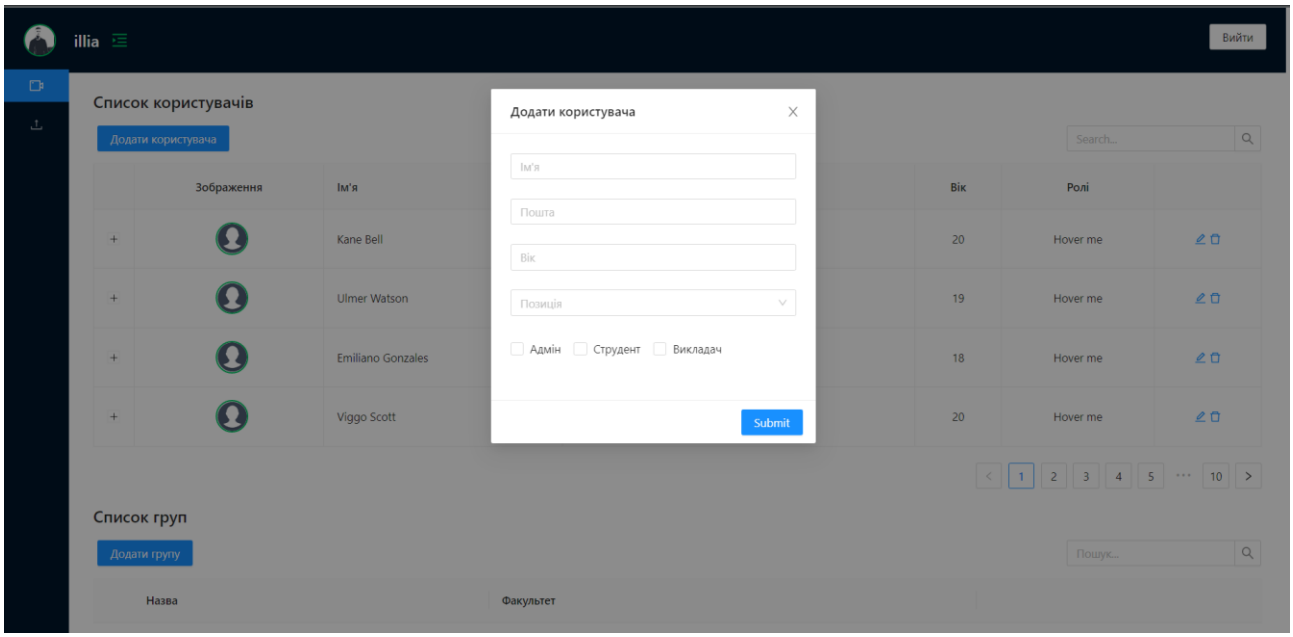


Рис. 7.5.2 Форма для створення користувача

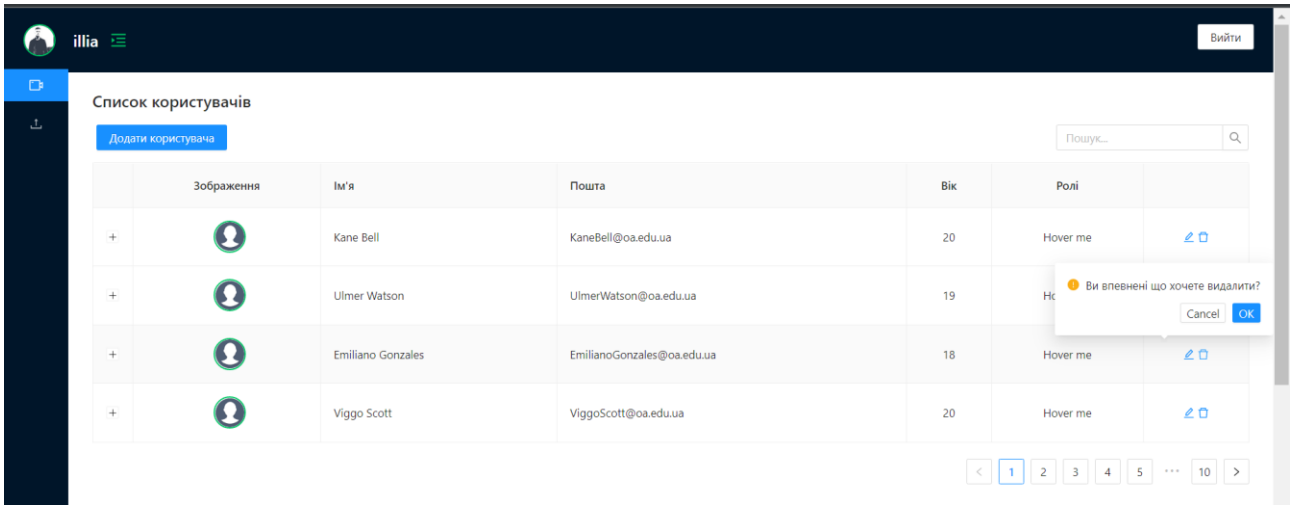


Рис. 7.5.3 Видалення користувача

Адміністратор може створювати нові групи.

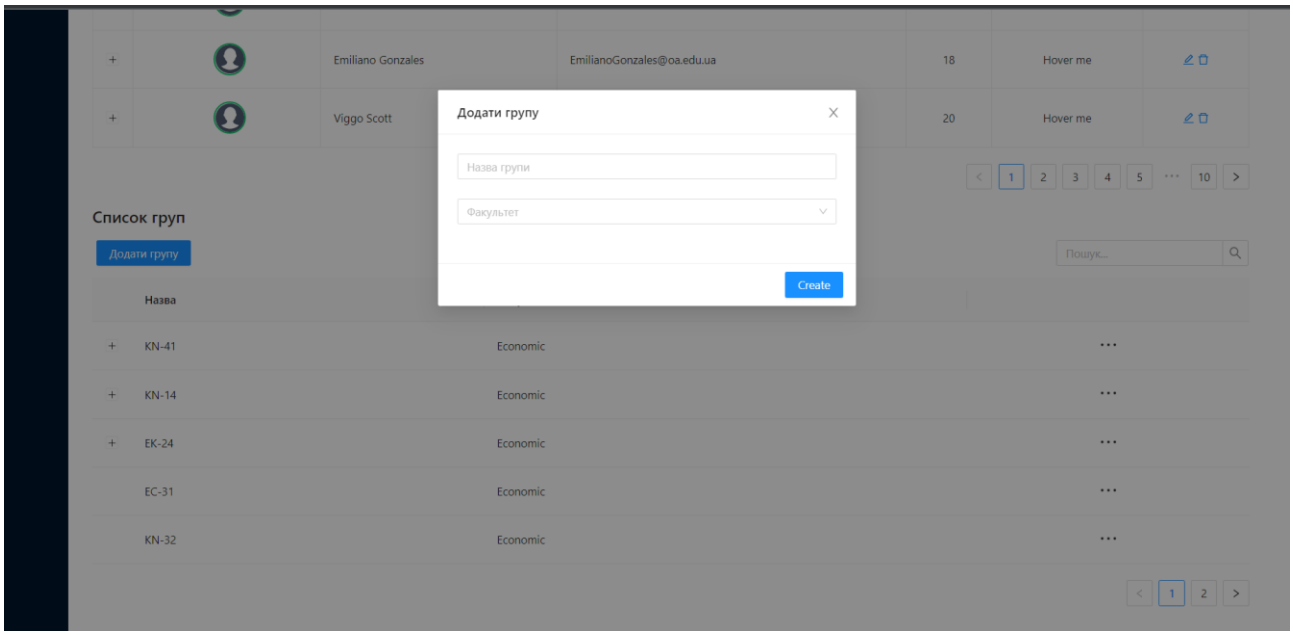


Рис. 7.5.4 Форма для створення групи

Групу можна видалити та редагувати.

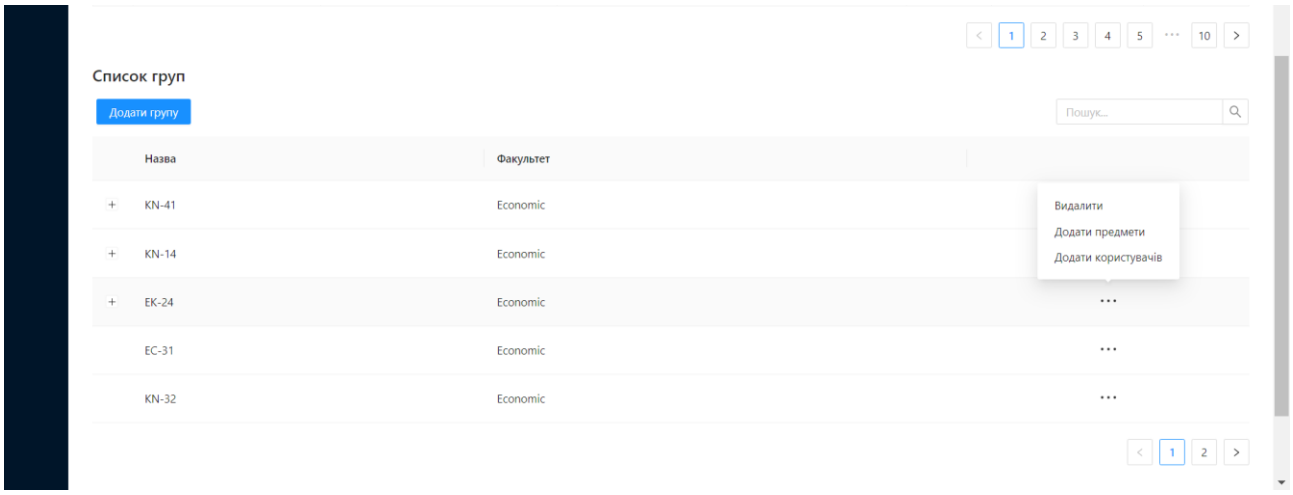


Рис. 7.5.5 Функціонал для редагування групи

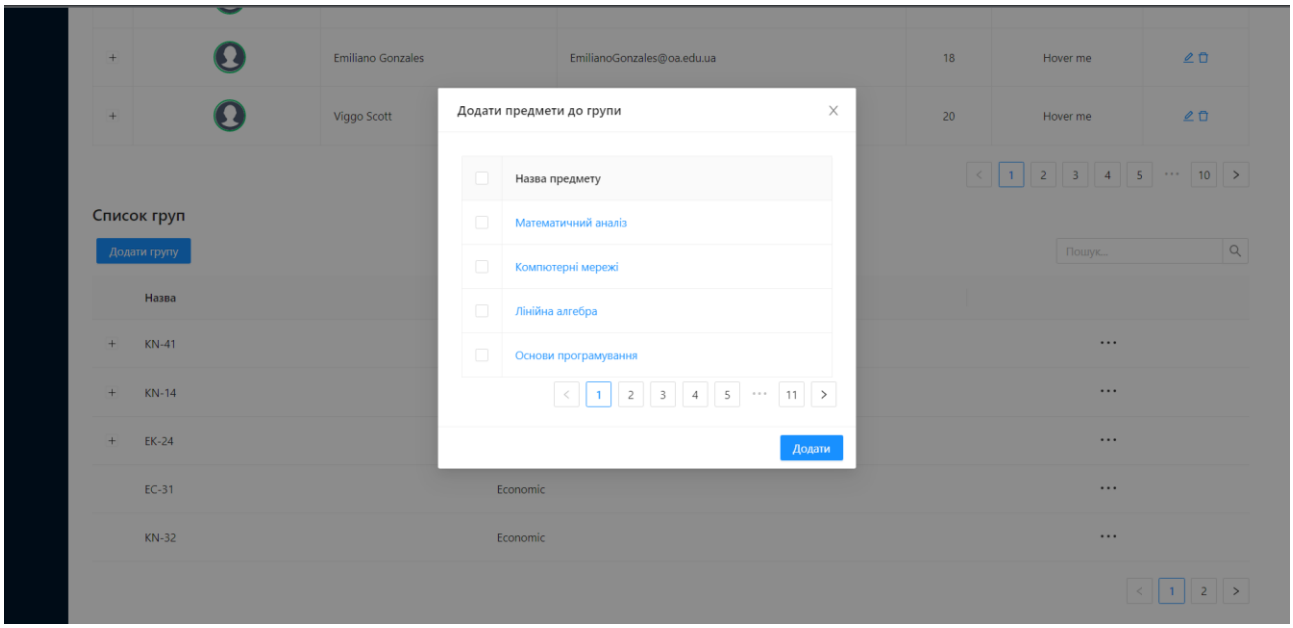


Рис. 7.5.6. Форма для додавання предметів до групи

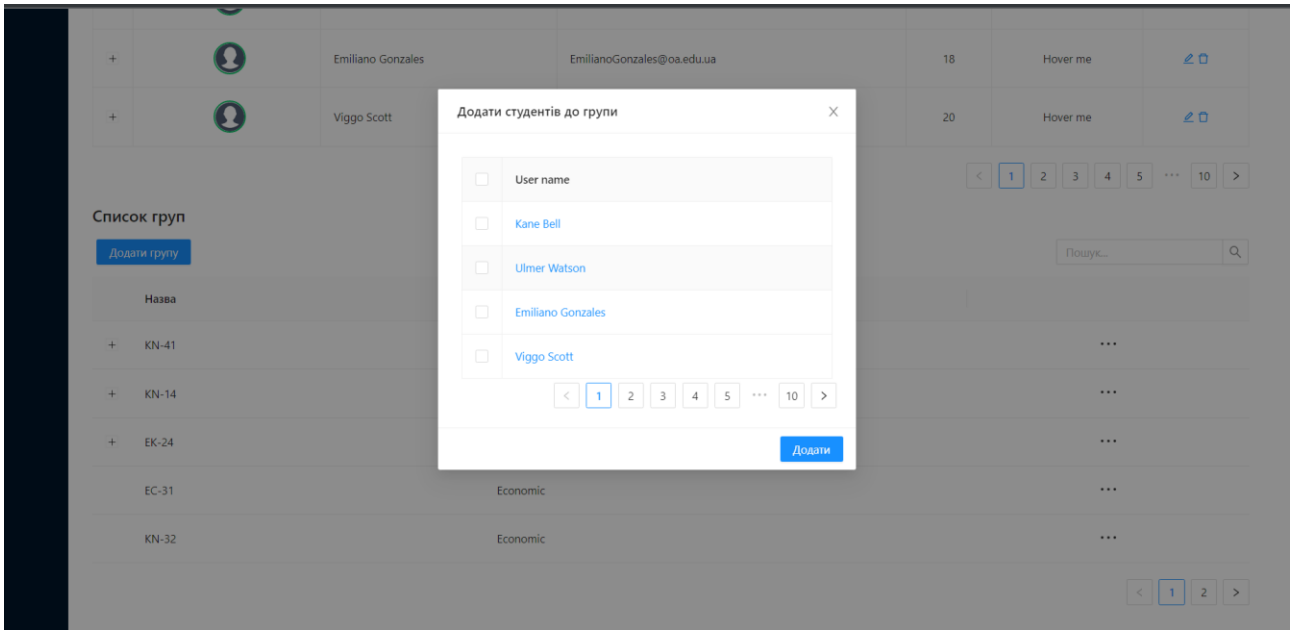


Рис. 7.5.7. Форма для додавання студентів до групи

ВИСНОВОК

У результаті виконання дипломної роботи було розроблено програмний додаток системи контролю відвідуваності занять у навчальних закладах. Користувачами додатку є студенти та викладачі.

Для досягнення поставленої мети було здійснено аналіз існуючих варіантів рішень задач і на основі цього аналізу розроблено ряд ефективних рішень для впровадження даної інформаційної системи. Користувач має власний кабінет, вся інформація є захищеною. За допомогою додатку пришвидшується комунікація між студентами та викладачами. Цей програмний продукт надає студенту всю необхідну інформацію по курсу, все зібрано в одному місці. Студент в будь який момент часу зможе переглянути свою успішність з кожного предмету, йому не потрібно на пряму зв’язуватись з викладачем, для того, щоб дізнатися про свої пропуски та підсумковий бал на курсі, все це буде доступним в його особистому кабінеті.

Студент весь час може моніторити свої підсумкові оцінки з кожного предмету, для цього на його сторінці є інтерактивна залікова книжка, вона має цікавий дизайн, адже загальний бал може змінювати свій колір залежно від оцінки за системою ECTS.

У викладача є пошук по предмету, інформація про студентів на курсі попередньо завантажена адміністратором, все, що робить викладач, це створює нове заняття, виставляє бали або пропуски студентам даного курсу. Є можливість створювати звіти.

Додаток має ряд переваг над іншими, найперше це зручність, доступність та швидкість у використанні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Excel [Електронний ресурс] // wikipedia. – 2021. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Microsoft_Excel.
2. Visual Studio - продукт Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.
3. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/?wt.mc_id=DX_841432.
4. Figma [Електронний ресурс] // wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Figma.
5. How to build and deploy a three-layer architecture application with C# [Електронний ресурс] // Enlab. – 2021. – Режим доступу до ресурсу: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-layer-architecture-application-with-c-sharp-net-in-practice.html.
6. Введение в Entity Framework Core [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://metanit.com/sharp/efcore/1.1.php.
7. What is Code-First? [Електронний ресурс] // Entity Framework Tutorial – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx.
8. EF Code-First Approach Vs Database-First Approach [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/blogs/ef-codefirst-approach-vs-database-first-approach.
9. Migration in EF 6 Code-First [Електронний ресурс] // Entity Framework Tutorial – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/code-first/migration-in-code-first.aspx.
10. Загрузка связанных данных [Електронний ресурс] – Режим доступу до ресурсу: https://professorweb.ru/my/entity-framework/6/level3/3_4.php.

11. Loading Entities and Navigation Properties / Z.Hirani, L. Tenny, N. Gupta, B. Driscol // Entity Framework 6 Recipes / Z.Hirani, L. Tenny, N. Gupta, B. Driscol. – Нью-Йорк: Apress, 2013. – С. 129
12. Основы проектирования баз данных. Создание базы данных и таблиц [Електронний ресурс] // Metanit. – 2017. – Режим доступу до ресурсу: https://metanit.com/sql/tutorial/1.1.php
13. Ключи [Електронний ресурс] // metanit – Режим доступу до ресурсу: https://metanit.com/sql/tutorial/1.2.php.
14. JWT [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/.
15. Введение в React [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/web/react/1.1.php.
16. Основы JSX [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://metanit.com/web/react/1.3.php
17. Building the Confluent UI with React Hooks [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.confluent.io/blog/moving-to-react-benefits-and-lessons-learned-building-confluent-cloud-ui/.
18. Краткое руководство по Redux для начинающих [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://tproger.ru/translations/redux-for-beginners/.
19. props vs state [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/uberVU/react-guide/blob/master/props-vs-state.md.
20. Working with Redux: Pros and Cons [Електронний ресурс] – Режим доступу до ресурсу: https://blog.boardinfinity.com/working-with-redux-pros-and-cons/.
21. When should I use Redux? [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/faq/general#when-should-i-use-redux.
22. Ant design [Електронний ресурс] – Режим доступу до ресурсу: https://ant.design/.
23. React Router [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/docs/en/v6/getting-started/tutorial#introduction.
24. БЕМ методологія [Електронний ресурс] – Режим доступу до ресурсу: https://ru.bem.info/methodology/quick-start/.

ДОДАТОК А

Система контролю відвідуваності занять у навчальних закладах

Swagger

Аркушів 2

Острог 2022

Authentication

- POST /api/Authentication/authenticate
- POST /api/Authentication/refresh-token

Students

- POST /api/Students/addFacultyGroup
- POST /api/Students/addGroupSpecialty
- GET /api/Students/getFaculty
- GET /api/Students/getGroups
- GET /api/Students/getSpecialtys
- GET /api/Students/getAllGroups
- POST /api/Students/createFaculty

Teachers

- GET /api/Teachers/getSubjects
- GET /api/Teachers/getAllTeacherFaculty
- GET /api/Teachers/getAllTeacherGroups
- GET /api/Teachers/GetAllSubject
- GET /api/Teachers/getAllUsersWithSubjests
- GET /api/Teachers/getAllGroupsBySubject
- POST /api/Teachers/createSubject
- POST /api/Teachers/addLesson
- PUT /api/Teachers/updateSubject
- DELETE /api/Teachers/deleteSubject
- DELETE /api/Teachers/deleteLesson

Users

GET	/api/Users/getUsers
GET	/api/Users/getCurrentUser
GET	/api/Users/getTimeTable
GET	/api/Users/getUsersTimeTable
GET	/api/Users/GetSubjectTopic
GET	/api/Users/getAllTeacherSubject
GET	/api/Users/getTimeTableByUser
POST	/api/Users/addRole
POST	/api/Users/deleteRole
POST	/api/Users/createUser
POST	/api/Users/addSubject
POST	/api/Users/addGroup
POST	/api/Users/createTimeTable
POST	/api/Users/AddUsersFromExel
GET	/api/Users/exportToExcelUserTimeTable
POST	/api/Users/createGroup
PUT	/api/Users/updateUserScore
PUT	/api/Users/updateUserIsPresent
PUT	/api/Users/updateUser
DELETE	/api/Users/removeSubject
DELETE	/api/Users/deleteUser

ДОДАТОК Б

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 9

Острог 2022

AuthenticationController.cs

```
using QualificationWork.BL.Services;
using QualificationWork.DTO.Dtos;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using QualificationWork.ClaimsExtension;
using Microsoft.AspNetCore.Authorization;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        private readonly AuthenticationService authenticationService;
        private readonly UserService userService;

        public AuthenticationController(AuthenticationService authenticationService,UserService
userService)
        {
            this.userService = userService;
            this.authenticationService = authenticationService;
        }

        [HttpPost("authenticate")]
        public async Task<IActionResult> Authenticate([FromBody] GoogleAuth model)
        {
            var response = await authenticationService.Authenticate(model.googleToken,
ipAddress());
            return Ok(response);
        }

        [HttpPost("refresh-token")]
        public async Task<IActionResult> RefreshToken([FromBody] RefreshTokenRequest model)
        {
            var response = await authenticationService.RefreshToken(model.RefreshToken,
ipAddress());
            return Ok(response);
        }

        private string ipAddress()
        {
            if (Request.Headers.ContainsKey("X-Forwarded-For"))
            {
                return Request.Headers["X-Forwarded-For"];
            }
            else
            {
                return HttpContext.Connection.RemoteIpAddress.MapToIPv4().ToString();
            }
        }

        [Authorize]
        [HttpGet("getCurrentUser")]
        public async Task<ActionResult> GetCurrentUser()
        {
            var data = await userService.GetUser(User.GetUserId());
            return Ok(data);
        }

        public class GoogleAuth
```

```
        {
            public string googleToken { get; set; }
        }

        public class RefreshTokenRequest
        {
            public string RefreshToken { get; set; }
        }
    }
}
```

ExcelsController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using QualificationWork.BL.Services;
using QualificationWork.ClaimsExtension;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ExelsController : ControllerBase
    {
        private readonly UserService userService;
        private readonly ExcelService excelService;

        public ExelsController(UserService userService, ExcelService excelService)
        {
            this.userService = userService;
            this.excelService = excelService;
        }

        [HttpPost("AddUsersFromExel")]
        public async Task<ActionResult> AddUsersFromExel([FromForm] ExelDto model)
        {
            var data = await excelService.Import(model.file);
            await userService.AddRangeUsers(data);
            return Ok(data);
        }

        [HttpPost("AddSubjectsFromExel")]
        public async Task<ActionResult> AddSubjectsFromExel([FromForm] ExelDto model)
        {
            var data = await excelService.ImportSubject(model.file);
            return Ok(data);
        }
        [HttpPost("addStudentSubjectFromExel")]
        public async Task<ActionResult> AddStudentSubjectFromExel([FromForm] ExelDto model)
        {
            var data = await excelService.ImportStusentSubject(model.file);
            return Ok(data);
        }
        [HttpPost("AddFacultyFromExel")]
```

```
        public async Task<ActionResult> AddFacultyFromExel([FromForm] ExelDto model)
        {
            var data = await excelService.ImportFaculty(model.file);
            return Ok(data);
        }

        [HttpGet("exportToExcelBySubject")]
        public async Task<ActionResult> ExportToExcelBySubject(long subjectId)
        {
            var stream = await excelService.ExportToExcelBySubject(subjectId);
            Response.ContentType = new MediaTypeHeaderValue("application/octet-
stream").ToString();
            return File(stream, "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet", "users.xlsx");
        }

        [Authorize]
        [HttpGet("exportToExcelByUser")]
        public async Task<ActionResult> ExportToExcelByUser()
        {
            var stream = await excelService.ExportToExcelByUser(User.GetUserId());
            Response.ContentType = new MediaTypeHeaderValue("application/octet-
stream").ToString();
            return File(stream, "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet", "Report.xlsx");
        }

    }
}
```

StudentsController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using QualificationWork.BL.Services;
using QualificationWork.ClaimsExtension;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System.Threading.Tasks;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]

    [Authorize(Roles = "Student")]
    public class StudentsController : ControllerBase
    {
        private readonly GroupService groupService;
        private readonly SubjectService subjectService;
        private readonly UserService userService;

        public StudentsController(GroupService groupService,UserService
userService,SubjectService subjectService)
        {
            this.groupService = groupService;
            this.userService = userService;
            this.subjectService = subjectService;
        }

        [HttpPost("addFacultyGroup")]
```

```
        public async Task<ActionResult> AddFacultyGroup(long facultyId, string GroupName)
        {
          await groupService.AddFacultyGroupAsync(facultyId, GroupName);
          return Ok();
        }

        [HttpPost("addGroupSpecialty")]
        public async Task<ActionResult> AddGroupSpecialty(long groupId, string specialtyName)
        {
            await groupService.AddGroupSpecialtyAsync(groupId, specialtyName);
            return Ok();
        }

        [HttpPost("addUserGroup")]
        public async Task<ActionResult> AddUserGroup([FromBody] AddUserGroupDto model)
        {
            await groupService.AddUserGroup(model.groupId, model.arrUserId);
            return Ok();
        }


        [HttpGet("getFaculty")]
        public ActionResult GetFaculty()
        {
            var data= groupService.GetFaculty();
            return Ok(data);
        }

        [HttpGet("getGroups")]
        public ActionResult GetGroups()
        {
            var data = groupService.GetGroups();
            return Ok(data);
        }

        [HttpGet("getTimeTableByUser")]
        public async Task<ActionResult> GetTimeTableByUser(long subjectId)
        {
            var data = await userService.GetTimeTableByUser(subjectId, User.GetUserId());
            return Ok(data);
        }
        [HttpGet("GetAllSubject")]
        public async Task<ActionResult> GetAllSubject()
        {
            var data = await subjectService.GetAllSubject(User.GetUserId());
            return Ok(data);
        }

        [HttpPost("createFaculty")]
        public async Task<ActionResult> CreateFaculty([FromBody] FacultyDto model)
        {
            await groupService.CreateFacultyAsync(model);
            return Ok();
        }
    }
}
```

TeachersController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
```

```
using QualificationWork.BL.Services;
using QualificationWork.ClaimsExtension;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "Teacher")]
    public class TeachersController : ControllerBase
    {

        private readonly SubjectService subjectService;
        private readonly UserService userService;

        public TeachersController(SubjectService subjectService, UserService userService)
        {
            this.subjectService = subjectService;
            this.userService = userService;
        }

        [HttpGet("getAllTeacherFaculty")]
        public async Task<ActionResult> GetAllTeacherFaculty(long facultyId)
        {
            var data = await subjectService.GetAllTeacherFaculty(facultyId);
            var result = data.Where(x => x.UserRoles.Any(y => y.Role.Name == UserRoles.Teacher));
            return Ok(data);
        }

        [HttpGet("getAllTeacherGroups")]
        public async Task<ActionResult> GetAllTeacherGroups(long groupId)
        {
            var data = await subjectService.GetAllTeacherGroups(groupId);
            var result = data.Where(x => x.UserRoles.Any(y => y.Role.Name == UserRoles.Teacher));
            return Ok(data);
        }
        [HttpGet("getAllSubjests")]
        public async Task<ActionResult> GetAllSubjects(int pageNumber, int pageSize, string
search)
        {
             var data = await subjectService.GetAllSubjects(pageNumber, pageSize, search);
             return Ok(data);
        }
        [HttpGet("getUsersTimeTable")]
        public async Task<ActionResult> GetUsersTimeTable(long subjectId, int namberleson)
        {
            var data = await userService.GetUsersTimeTable(subjectId, namberleson);
            return Ok(data);
        }
        [HttpGet("getAllGroupsBySubject")]
        public async Task<ActionResult> GetAllGroupsBySubject(long subjectId)
        {
            var data= await subjectService.GetAllGroupsBySubject(subjectId);
            return Ok(data);
        }

        [HttpGet("getAllTeacherSubject")]
        public async Task<ActionResult> GetAllTeacherSubject()
        {
```

```
            var data = await userService.GetAllTeacherSubject(User.GetUserId());
            return Ok(data);
        }

        [HttpGet("GetSubjectTopic")]
        public async Task<ActionResult> GetSubjectTopic(long subjectId)
        {
            var data = await userService.GetSubjectTopic(subjectId);
            return Ok(data);
        }

        [HttpPost("createSubject")]
        public async Task<ActionResult> CreateSubject([FromBody]SubjectDto model)
        {
            await subjectService.CreateSubjectAsync(model);
            return Ok();
        }
        [HttpPost("addLesson")]
        public async Task<ActionResult> AddLessonAsync([FromBody] AddLessonDto model)
        {
            await subjectService.AddLessonAsync(model);
            return Ok();
        }

        [HttpPut("updateSubject")]
        public async Task<ActionResult> UpdateSubjectAsync(long subjectId, [FromBody]SubjectDto
model)
        {
            await subjectService.UpdateSubjectAsync(subjectId, model);
            return Ok();
        }

        [HttpDelete("deleteSubject")]
        public ActionResult DeleteSubject(long subjectId)
        {
            subjectService.DeleteSubject(subjectId);
            return Ok();
        }
        [HttpDelete("deleteLesson")]
        public async Task<ActionResult> DeleteLessonAsync(int lessonNumber, long subjectId)
        {
            await subjectService.DeleteLessonAsync(lessonNumber, subjectId);
            return Ok();
        }

        [HttpPut("updateUserScore")]
        public async Task<ActionResult> UpdateUserScore([FromBody] UpdateUserScoreDto model)
        {
            await userService.UpdateUserScore(model);
            return Ok();
        }

        [HttpPut("updateUserIsPresent")]
        public async Task<ActionResult> UpdateUserIsPresent([FromBody] UpdateUserIsPresentDto
model)
        {
            await userService.UpdateUserIsPresent(model);
            return Ok();
        }


    }
```

```
}
```

UsersController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using QualificationWork.BL.Services;
using QualificationWork.ClaimsExtension;
using QualificationWork.DTO.Dtos;
using System.Threading.Tasks;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "Admin")]
    public class UsersController : ControllerBase
    {
        private readonly UserService userService;
        private readonly SubjectService subjectService;
        private readonly GroupService groupService;

        public UsersController(GroupService groupService,SubjectService subjectService,
UserService userService)
        {
            this.userService = userService;
            this.subjectService = subjectService;
            this.groupService = groupService;
        }

        [HttpGet("getUsers")]
        public async Task<ActionResult> GetUsers()
        {
            var data = await userService.GetUsers();
            return Ok(data);
        }

        [HttpGet("getAllGroups")]
        public async Task<ActionResult> GetAllGroupsAsync(int pageNumber, int pageSize, string
search)
        {
            var data = await groupService.GetAllGroups(pageNumber, pageSize, search);
            return Ok(data);
        }

        [HttpGet("getAllUsersWithSubjests")]
        public async Task<ActionResult> GetAllUsersWithSubjests(int pageNumber, int pageSize,
string search)
        {
            var data = await subjectService.GetAllUsersWithSubjests(pageNumber, pageSize,
search);
            return Ok(data);
        }

        [HttpGet("getTimeTable")]
        public async Task<ActionResult> GetTimeTable()
        {
            var data = await userService.GetTimeTable();
            return Ok(data);
        }
```

```
[HttpPost("addRole")]
public async Task<ActionResult> AddRole([FromBody] RoleDto model)
{
    await userService.AddRoleAsync(model.UserId, model.RoleName);
    return Ok();
}

[HttpPost("deleteRole")]
public async Task<ActionResult> DeleteRole([FromBody] RoleDto model)
{
    await userService.DeleteRolesAsync(model.UserId, model.RoleName);
    return Ok();
}

[HttpPost("createUser")]
public async Task<ActionResult> CreateUser([FromBody] UserDto model)
{
    await userService.CreateUserAsync(model);
    return Ok();
}

[HttpPost("addGroup")]
public async Task<ActionResult> AddGroup(long userId, long groupId)
{

    await userService.AddGroup(userId, groupId);
    return Ok();
}

[HttpPost("addUserGroup")]
public async Task<ActionResult> AddUserGroup([FromBody] AddUserGroupDto model)
{
    await groupService.AddUserGroup(model.groupId, model.arrUserId);
    return Ok();
}

[HttpPost("createTimeTable")]
public async Task<ActionResult> CreateTimeTable([FromBody] TimeTableDto model)
{
    await userService.CreateTimeTableAsync(model);
    return Ok();
}

[HttpPost("createGroup")]
public async Task<ActionResult> CreateGroup([FromBody] CreateGroupDto model)
{
    await userService.CreateGroup(model);
    return Ok();
}

[HttpPut("updateUser")]
public async Task<ActionResult> UpdateUserAsync( [FromBody] EditeUserDto model)
{
    await userService.UpdateUserAsync(model);
    return Ok();
}

[HttpDelete("removeSubject")]
public ActionResult RemoveSubject([FromBody] UserSubjectDto model)
{
    userService.RemoveSubject(model.UserId, model.SubjectId);
    return Ok();
```

```
        }

        [HttpPost("addUserSubject")]
        public async Task<ActionResult> AddGroupSubject([FromBody] AddUserGroupDto model)
        {
            await groupService.AddGroupSubject(model.groupId, model.arrUserId);
            return Ok();
        }

        [HttpGet("getSubjects")]
        public ActionResult GetSubjects()
        {
            var data = subjectService.GetSubjects();
            return Ok(data);
        }

        [HttpDelete("deleteUser")]
        public ActionResult DeleteUser(long userId)
        {
            userService.DeleteUser(userId);
            return Ok();
        }

        [HttpDelete("deleteGroup")]
        public ActionResult DeleteGroup(long groupId)
        {
            groupService.DeleteGroup(groupId);
            return Ok();
        }

    }
}
```

ДОДАТОК В

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 18

Острог 2022

AuthenticationService.cs

```
using QualificationWork.DAL;
using QualificationWork.DAL.Command;
using QualificationWork.DTO.Dtos;
using System.Threading.Tasks;

namespace QualificationWork.BL.Services
{
    public class AuthenticationService
    {
        private readonly AuthenticationCommand authenticationCommand;
        private readonly ApplicationContext context;

        public AuthenticationService(AuthenticationCommand authenticationCommand,
ApplicationContext context)
        {

            this.authenticationCommand = authenticationCommand;
            this.context = context;
        }

        public async Task<AuthenticateResponseDto> Authenticate(string accessToken, string
ipAddress)
        {
            var authenticate = await authenticationCommand.Authenticate(accessToken, ipAddress);

            await context.SaveChangesAsync();

            return authenticate;

        }
        public async Task<AuthenticateResponseDto> RefreshToken(string token, string ipAddress)
        {
            var refreshToken = await authenticationCommand.RefreshToken(token, ipAddress);

            await context.SaveChangesAsync();

            return refreshToken;
        }

    }
}
```

GroupService.cs

```
using QualificationWork.DAL;
using QualificationWork.DAL.Command;
using QualificationWork.DAL.Models;
using QualificationWork.DAL.Query;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace QualificationWork.BL.Services
{
    public class GroupService
    {
        private readonly GroupCommand groupCommand;
```

```
        private readonly GroupQuery groupQuery;

        private readonly ApplicationContext context;

        public GroupService(ApplicationContext context, GroupQuery groupQuery, GroupCommand
groupCommand)
        {
            this.context = context;
            this.groupCommand = groupCommand;
            this.groupQuery = groupQuery;
        }

        public async Task AddFacultyGroupAsync(long facultyId, string GroupName)
        {
            await groupCommand.AddFacultyGroupAsync(facultyId, GroupName);
            await context.SaveChangesAsync();
        }
        public async Task CreateFacultyAsync(FacultyDto model)
        {
            await groupCommand.CreateFacultyAsync(model);
            await context.SaveChangesAsync();
        }
        public async Task AddGroupSpecialtyAsync(long groupId, string specialtyName)
        {
            await groupCommand.AddGroupSpecialtyAsync(groupId, specialtyName);
            await context.SaveChangesAsync();
        }

        public List<Group> GetGroups()
        {
            return groupQuery.GetGroups();
        }

        public List<Faculty> GetFaculty()
        {
            return groupQuery.GetFaculty();
        }

        public async Task<Pagination<Group>> GetAllGroups(int pageNumber, int pageSize, string
search)
        {
            return await groupQuery.GetAllGroups(pageNumber, pageSize, search);
        }

        public async Task AddUserGroup(long groupId, long[] arrUserId)
        {
            await groupCommand.AddUserGroup(groupId, arrUserId);
            await context.SaveChangesAsync();
        }

        public void DeleteGroup(long groupId)
        {
            groupCommand.DeleteGroup(groupId);
            context.SaveChanges();
        }

        public async Task AddGroupSubject(long groupId, long[] arrSubjectId)
        {
            await groupCommand.AddUserGroup(groupId, arrSubjectId);
            await context.SaveChangesAsync();
        }
    }
```

```
}
```

SubjectService.cs

```
using QualificationWork.DAL;
using QualificationWork.DAL.Command;
using QualificationWork.DAL.Models;
using QualificationWork.DAL.Query;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using static QualificationWork.DAL.Query.SubjectQuery;

namespace QualificationWork.BL.Services
{
    public class SubjectService
    {
        private readonly SubjectQuery subjectQuery;

        private readonly SubjectCommand subjectCommand;

        private readonly ApplicationContext context;

        public SubjectService(ApplicationContext context, SubjectCommand subjectCommand,
SubjectQuery subjectQuery)
        {
            this.context = context;
            this.subjectQuery = subjectQuery;
            this.subjectCommand = subjectCommand;
        }

        public async Task CreateSubjectAsync(SubjectDto model)
        {
            await subjectCommand.CreateSubjectAsync(model);
            await context.SaveChangesAsync();
        }

        public async Task UpdateSubjectAsync(long subjectId, SubjectDto model)
        {
            await subjectCommand.UpdateSubjectAsync(subjectId, model);
            await context.SaveChangesAsync();
        }

        public void DeleteSubject(long subjectId)
        {
            subjectCommand.DeleteSubject(subjectId);
            context.SaveChanges();
        }

        public List<Subject> GetSubjects()
        {
            return subjectQuery.GetSubjects();
        }

        public async Task<Pagination<Subject>> GetAllSubjects(int pageNumber, int pageSize,
string search)
        {
            return await subjectQuery.GetAllSubjects(pageNumber, pageSize, search);
        }
```

```
        public async Task<Pagination<ApplicationUser>> GetAllUsersWithSubjests(int pageNumber,
int pageSize, string search)
        {
            return await subjectQuery.GetAllUsersWithSubjests(pageNumber, pageSize, search);
        }

        public async Task<List<ApplicationUser>> GetAllTeacherFaculty(long facultyId)
        {
            return await subjectQuery.GetAllTeacherFaculty(facultyId);
        }

        public async Task<List<ApplicationUser>> GetAllTeacherGroups(long groupId)
        {
            return await subjectQuery.GetAllTeacherGroups(groupId);
        }


        public async Task<List<Group>> GetAllGroupsBySubject(long subjectId)
        {
            return await subjectQuery.GetAllGroupsBySubject(subjectId);
        }

        public async Task<List<Subject>> GetAllSubject(long userId)
        {
            return await subjectQuery.GetAllSubject(userId);
        }

        public async Task AddLessonAsync(AddLessonDto model)
        {
            await subjectCommand.AddLessonAsync(model);
            await context.SaveChangesAsync();
        }

        public async Task DeleteLessonAsync(int lessonNumber, long subjectId)
        {
            await subjectCommand.DeleteLessonAsync(lessonNumber, subjectId);
            await context.SaveChangesAsync();
        }
    }
}
```

UserService.cs

```
using QualificationWork.DAL;
using QualificationWork.DAL.Command;
using QualificationWork.DAL.Models;
using QualificationWork.DAL.Query;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using static QualificationWork.DAL.Command.UserCommand;

namespace QualificationWork.BL.Services
{
    public class UserService
    {

        private readonly UserCommand userCommand;
```

```
        private readonly UserQuery userQuery;
        private readonly ApplicationContext context;

        public UserService(ApplicationContext context, UserCommand userCommand, UserQuery
userQuery)
        {
            this.userCommand = userCommand;
            this.context = context;
            this.userQuery = userQuery;
        }

        public async Task<List<ApplicationUser>> GetUsers()
        {
            return await userQuery.GetUsers();
        }

        public async Task<List<TimeTable>> GetTimeTable()
        {
            return await userQuery.GetTimeTable();
        }

        public async Task AddRoleAsync(string userId, string roleName)
        {
            await userCommand.AddRoleAsync(userId, roleName);
            await context.SaveChangesAsync();
        }

        public async Task DeleteRolesAsync(string userId, string roleName)
        {
            await userCommand.DeleteRolesAsync(userId, roleName);
            await context.SaveChangesAsync();
        }

        public async Task CreateUserAsync(UserDto model)
        {
            await userCommand.CreateUserAsync(model);
            await context.SaveChangesAsync();
        }

        public async Task UpdateUserAsync(EditeUserDto model)
        {
            await userCommand.UpdateUserAsync(model);
            await context.SaveChangesAsync();
        }

        public async Task AddGroup(long userId, long groupId)
        {
            await userCommand.AddGroup(userId, groupId);
            await context.SaveChangesAsync();

        }

        public void RemoveSubject(long userId, long subjectId)
        {
            userCommand.RemoveSubject(userId, subjectId);
            context.SaveChanges();

        }

        public async Task<ApplicationUser> GetUser(long userId)
        {
            return await userQuery.GetUser(userId);
```

```
        }

        public async Task CreateTimeTableAsync(TimeTableDto model)
        {
            await userCommand.CreateTimeTableAsync(model);
            await context.SaveChangesAsync();
        }

        public void DeleteUser(long userId)
        {
            userCommand.DeleteUser(userId);
            context.SaveChanges();
        }

        public async Task AddRangeUsers(List<UserFromExcelDto> model)
        {
           await userCommand.AddRangeUsers(model);
           await context.SaveChangesAsync();
        }

        public async Task<List<Subject>> GetAllTeacherSubject(long userId)
        {
            return await userQuery.GetAllTeacherSubject(userId);
        }

        public async Task<List<ApplicationUser>> GetUsersTimeTable(long subjectId, int
namberleson)
        {
            return await userQuery.GetUsersTimeTable(subjectId, namberleson);
        }

        public async Task<SubjectLessonsDto<TimeTable>> GetSubjectTopic(long subjectId)
        {
            return await userQuery.GetSubjectTopic(subjectId);
        }

        public async Task UpdateUserScore(UpdateUserScoreDto model)
        {
            await userCommand.UpdateUserScore(model);
            await context.SaveChangesAsync();
        }

        public async Task UpdateUserIsPresent(UpdateUserIsPresentDto model)
        {
            await userCommand.UpdateUserIsPresent(model);
            await context.SaveChangesAsync();
        }

        public async Task<Subject> GetTimeTableByUser(long subjectId, long userId)
        {
            return await userQuery.GetTimeTableByUser(subjectId, userId);
        }

        public async Task CreateGroup(CreateGroupDto model)
        {
            await userCommand.CreateGroup(model);
            await context.SaveChangesAsync();
        }

    }
}
```

ExcelService.cs

```
using Microsoft.AspNetCore.Http;
using OfficeOpenXml;
using OfficeOpenXml.Style;
using QualificationWork.DAL;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;
using QualificationWork.DAL.Query;
using System.Linq;
using System.Data.Entity;
using QualificationWork.DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace QualificationWork.BL.Services
{
    public class ExcelService
    {
        private readonly ApplicationContext context;
        private readonly GroupQuery groupQuery;

        public ExcelService(ApplicationContext context, GroupQuery groupQuery)
        {
            this.context = context;
            this.groupQuery = groupQuery;
        }

        public async Task<List<UserFromExcelDto>> Import(IFormFile file)
        {
            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
```

```
            var list = new List<UserFromExcelDto>();
            using (var stream = new MemoryStream())
            {
                await file.CopyToAsync(stream);
                using (var package = new ExcelPackage(stream))
                {
                    ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
                    var rowcount = worksheet.Dimension.Rows;
                    for (int row = 2; row <= rowcount; row++)
                    {
                        list.Add(new UserFromExcelDto
                        {
                            UserName = worksheet.Cells[row, 1].Value.ToString().Trim(),
                            UserEmail = worksheet.Cells[row, 2].Value.ToString().Trim(),
                            Age = Convert.ToInt32(worksheet.Cells[row,
3].Value.ToString().Trim()),
                            IsContract = Convert.ToBoolean(worksheet.Cells[row,
4].Value.ToString().Trim())
                        });
                    }
                }
            }
            return list;
        }

        public async Task<List<GroupDto>> ImportFacultyGroups(IFormFile file)
        {
            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
            var listGroup = new List<GroupDto>();
            var listFaculty = new List<FacultyDto>();

            using (var stream = new MemoryStream())
            {
                await file.CopyToAsync(stream);
                using (var package = new ExcelPackage(stream))
                {
```

```
                ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
                var rowcount = worksheet.Dimension.Rows;
                for (int row = 2; row <= rowcount; row++)
                {
                    listGroup.Add(new GroupDto
                    {
                        GroupName = worksheet.Cells[row, 1].Value.ToString().Trim()
                    });

                    listFaculty.Add(new FacultyDto
                    {
                        FacultyName = worksheet.Cells[row, 2].Value.ToString().Trim()
                    });
                }
            }
        }

        foreach (var group in listGroup)
        {


            foreach (var faculty in listFaculty)
            {

                var facultyData = context.Faculties.FirstOrDefault(x => x.FacultyName ==
faculty.FacultyName);
                var dataGroup = new Group
                {
                    GroupName = group.GroupName,
                    FacultyId = facultyData.Id
                };

                await context.AddAsync(dataGroup);
            }
```

```
    }
    await context.SaveChangesAsync();
    return listGroup;
}

public async Task<List<SubjectDto>> ImportStusentSubject(IFormFile file)
{
    ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
    var listSubject = new List<SubjectDto>();
    var listUser = new List<UserDto>();

    using (var stream = new MemoryStream())
    {
        await file.CopyToAsync(stream);
        using (var package = new ExcelPackage(stream))
        {
            ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
            var rowcount = worksheet.Dimension.Rows;
            for (int row = 2; row <= rowcount; row++)
            {
                listUser.Add(new UserDto
                {
                    UserName = worksheet.Cells[row, 1].Value.ToString().Trim()
                });

                listSubject.Add(new SubjectDto
                {
                    SubjectName = worksheet.Cells[row, 2].Value.ToString().Trim()
                });
            }
        }
    }

    foreach (var user in listUser)
    {
```

```
                var userData = context.Users.FirstOrDefault(x => x.UserName == user.UserName);



                foreach (var subject in listSubject)
                {

                    var subjectData = context.Subjects.FirstOrDefault(x => x.SubjectName ==
subject.SubjectName);
                    var data = new TimeTable
                    {
                        UserId = userData.Id,
                        SubjectId = subjectData.Id,
                        IsPresent = false,
                        Score = 0,
                        LessonDate = DateTime.Now,
                        LessonNumber = 1,
                    };

                    await context.AddAsync(data);
                }


            }
            await context.SaveChangesAsync();
            return listSubject;
        }

        public async Task<List<Faculty>> ImportFaculty(IFormFile file)
        {
            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
            var listFaculty = new List<Faculty>();
            using (var stream = new MemoryStream())
            {
                await file.CopyToAsync(stream);
                using (var package = new ExcelPackage(stream))
                {
```

```
                ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
                var rowcount = worksheet.Dimension.Rows;
                for (int row = 2; row <= rowcount; row++)
                {
                    listFaculty.Add(new Faculty
                    {
                        FacultyName = worksheet.Cells[row, 1].Value.ToString().Trim()
                    });
                }
            }
        }

        foreach (var faculty in listFaculty)
        {
            var chekFaculty = context.Faculties.FirstOrDefault(x => x.FacultyName ==
faculty.FacultyName);

            if (chekFaculty == null)
            {
                await context.AddAsync(faculty);
            }
        }
        await context.SaveChangesAsync();

        return listFaculty;
    }
    public async Task<List<Subject>> ImportSubject(IFormFile file)
    {
        ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
        var listSubjects = new List<Subject>();
        using (var stream = new MemoryStream())
        {
            await file.CopyToAsync(stream);
            using (var package = new ExcelPackage(stream))
            {
```

```
                    ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
                    var rowcount = worksheet.Dimension.Rows;
                    for (int row = 2; row <= rowcount; row++)
                    {
                        listSubjects.Add(new Subject
                        {
                            SubjectName = worksheet.Cells[row, 1].Value.ToString().Trim(),
                            IsActive = Convert.ToBoolean(worksheet.Cells[row,
2].Value.ToString().Trim()),
                            AmountCredits = Convert.ToInt32(worksheet.Cells[row,
3].Value.ToString().Trim()),
                            SubjectClosingDate = Convert.ToDateTime(worksheet.Cells[row,
4].Value.ToString().Trim())
                        });
                    }
                }
            }

            foreach (var subject in listSubjects)
            {
                var chekSubject = context.Subjects.FirstOrDefault(x => x.SubjectName ==
subject.SubjectName);

                if (chekSubject == null)
                {
                    await context.AddAsync(subject);
                }
            }
            await context.SaveChangesAsync();

            return listSubjects;
        }

        public async Task<Stream> ExportToExcelBySubject(long subjectId)
        {
            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
```

```
            var subject = context.Subjects.FirstOrDefault(x => x.Id == subjectId);

            var stream = new MemoryStream();
            using (var xlPackage = new ExcelPackage(stream))
            {
                var worksheet = xlPackage.Workbook.Worksheets.Add(subject.SubjectName);
                var namedStyle = xlPackage.Workbook.Styles.CreateNamedStyle("HyperLink");
                namedStyle.Style.Font.UnderLine = true;
                namedStyle.Style.Font.Color.SetColor(Color.Blue);

                worksheet.Cells["A1"].Value = subject.SubjectName;
                worksheet.Cells["A1"].Style.Font.Bold = true;
                worksheet.Cells["A2"].Value = "І'мя";
                worksheet.Cells["B2"].Value = "Email";
                worksheet.Cells["C2"].Value = "Номер заняття";
                worksheet.Cells["D2"].Value = "Присутність";
                worksheet.Cells["E2"].Value = "Бал";
                worksheet.Cells["F2"].Value = "Дата заняття";
                worksheet.Cells["A1:F2"].Style.Fill.PatternType = ExcelFillStyle.Solid;
                worksheet.Cells["A1:F1"].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(24,
159, 24));
                worksheet.Cells["A2:F2"].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(184,
204, 228));
                worksheet.Cells["A2:F2"].Style.Font.Bold = true;

                var row = 3;

                var usersTimeTables = await groupQuery.GetTimeTableBySubject(subjectId);

                foreach (var timeTable in usersTimeTables)
                {
                    worksheet.Cells[row, 1].Value = timeTable.User.UserName;
                    worksheet.Cells[row, 2].Value = timeTable.User.Email;
                    worksheet.Cells[row, 3].Value = timeTable.LessonNumber;
```

```
                worksheet.Cells[row, 4].Value = !timeTable.IsPresent ? "Відсутній" :
"Присутній";
                worksheet.Cells[row, 5].Value = timeTable.Score;
                worksheet.Cells[row, 6].Style.Numberformat.Format = "m/d/yy h:mm";
                worksheet.Cells[row, 6].Value = timeTable.LessonDate;
                row++;
            }

            var listUniqueTimetable = new List<TimeTable>();

            foreach (var item in usersTimeTables)
            {
                var leson = listUniqueTimetable.FirstOrDefault(x => x.UserId == item.UserId);

                if (leson == null)
                {
                    listUniqueTimetable.Add(item);
                }
            }

            var rowCount = 2;
            var resultSheet = xlPackage.Workbook.Worksheets.Add("Підсумки");
            resultSheet.Cells["A1"].Value = "І'мя";
            resultSheet.Cells["B1"].Value = "Оцінка";
            resultSheet.Cells["A1:B1"].Style.Fill.PatternType = ExcelFillStyle.Solid;
            resultSheet.Cells["A1:B1"].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(24,
159, 24));
            resultSheet.Cells["A1:B2"].Style.Font.Bold = true;
            foreach (var timeTable in listUniqueTimetable)
            {
                var totaCount = 0;

                var userTimeTable = context.TimeTable
                                           .Where(x => x.UserId == timeTable.UserId)
                                           .Where(x => x.SubjectId == subjectId).ToList();
```

```
            foreach (var i in userTimeTable)
            {
                totaCount += i.Score;
            }

            resultSheet.Cells[rowCount, 1].Value = timeTable.User.UserName;
            resultSheet.Cells[rowCount, 2].Value = totaCount;
            rowCount++;
        }

        xlPackage.Workbook.Properties.Title = "User List";
        xlPackage.Workbook.Properties.Author = "Mohamad Lawand";
        xlPackage.Workbook.Properties.Subject = "User List";
        xlPackage.Save();
    }
    stream.Position = 0;
    return stream;
}

public async Task<Stream> ExportToExcelByUser(long userId)
{
    ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

    var stream = new MemoryStream();
    using (var xlPackage = new ExcelPackage(stream))
    {
        var worksheet = xlPackage.Workbook.Worksheets.Add("Звіт");
        var namedStyle = xlPackage.Workbook.Styles.CreateNamedStyle("HyperLink");
        namedStyle.Style.Font.UnderLine = true;

        worksheet.Cells["A1"].Value = "Предмет";
        worksheet.Cells["B1"].Value = "Номер заняття";
        worksheet.Cells["C1"].Value = "Присутність";
        worksheet.Cells["D1"].Value = "Бал";
        worksheet.Cells["E1"].Value = "Дата заняття";
```

```
                worksheet.Cells["A1:E1"].Style.Fill.PatternType = ExcelFillStyle.Solid;
                worksheet.Cells["A1:E1"].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(24,
159, 24));
                worksheet.Cells["A1:E1"].Style.Font.Bold = true;
                worksheet.Cells["A1:K20"].AutoFitColumns();

                var row = 2;

                var userTimeTable = await groupQuery.GetTimeTableByUser(userId);

                foreach (var timeTable in userTimeTable)
                {
                    worksheet.Cells[row, 1].Value = timeTable.Subject.SubjectName;
                    worksheet.Cells[row, 2].Value = timeTable.LessonNumber;
                    worksheet.Cells[row, 3].Value = !timeTable.IsPresent ? "Відсутній" :
"Присутній";
                    worksheet.Cells[row, 4].Value = timeTable.Score;
                    worksheet.Cells[row, 5].Style.Numberformat.Format = "m/d/yy h:mm";
                    worksheet.Cells[row, 5].Value = timeTable.LessonDate;
                    row++;
                }

                var listUniqueTimetable = new List<TimeTable>();

                foreach (var item in userTimeTable)
                {
                    var leson = listUniqueTimetable.FirstOrDefault(x => x.SubjectId ==
item.SubjectId);

                    if (leson == null)
                    {
                        listUniqueTimetable.Add(item);
                    }
                }
                var resultSheet = xlPackage.Workbook.Worksheets.Add("Підсумки");
```

```
                resultSheet.Cells["A1"].Value = "Предмет";
                resultSheet.Cells["B1"].Value = "Оцінка";
                resultSheet.Cells["A1"].Value = "І'мя";
                resultSheet.Cells["B1"].Value = "Оцінка";
                resultSheet.Cells["A1:B1"].Style.Fill.PatternType = ExcelFillStyle.Solid;
                resultSheet.Cells["A1:B1"].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(24,
159, 24));
                resultSheet.Cells["A1:B2"].Style.Font.Bold = true;
                var rowCount = 2;
                foreach (var timeTable in listUniqueTimetable)
                {
                    var totaCount = 0;

                    var subjectTimeTable = context.TimeTable
                                                .Where(x => x.UserId == userId)
                                                .Where(x => x.SubjectId ==
timeTable.SubjectId).ToList();

                    foreach (var i in subjectTimeTable)
                    {
                        totaCount += i.Score;
                    }

                    resultSheet.Cells[rowCount, 1].Value = timeTable.Subject.SubjectName;
                    resultSheet.Cells[rowCount, 2].Value = totaCount;
                    rowCount++;
                }

                xlPackage.Workbook.Properties.Title = "User List";
                xlPackage.Workbook.Properties.Author = "Mohamad Lawand";
                xlPackage.Workbook.Properties.Subject = "User List";
                xlPackage.Save();
            }
            stream.Position = 0;
            return stream;
}
```

ДОДАТОК Г

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 11

Острог 2022

AuthenticationCommand.cs

```
using Microsoft.Extensions.Options;
using QualificationWork.DAL.HelperServise;
using QualificationWork.DAL.Models;
using System;
using System.Threading.Tasks;
using System.Linq;
using QualificationWork.Middleware;
using QualificationWork.DTO.Dtos;
using System.Collections.Generic;
using Google.Apis.Auth;
using Microsoft.AspNetCore.Identity;
using System.Data.Entity;

namespace QualificationWork.DAL.Command
{
    public class AuthenticationCommand
    {
        private readonly ApplicationContext context;

        private readonly JwtUtils jwtUtils;

        private readonly AppSettings appSettings;

        private readonly UserManager<ApplicationUser> userManager;

        public AuthenticationCommand(
                ApplicationContext context,
                JwtUtils jwtUtils,
                IOptions<AppSettings> appSettings,
                UserManager<ApplicationUser> userManager
            )
        {
            this.context = context;
            this.jwtUtils = jwtUtils;
            this.appSettings = appSettings.Value;
            this.userManager = userManager;

        }

        public async Task<AuthenticateResponseDto> Authenticate(string accessToken, string
ipAddress)
        {
            var payload = GoogleJsonWebSignature.ValidateAsync(accessToken, new
GoogleJsonWebSignature.ValidationSettings()).Result;

            string domenOA = "@oa.edu.ua";

            var user =  context.Users.FirstOrDefault(x=>x.Email==payload.Email);

            if (user == null)
            {
                ApplicationUser userData = new ApplicationUser
                {
                    Email = payload.Email,
                    SecurityStamp = Guid.NewGuid().ToString(),
                    UserName = payload.GivenName,
                    Age = 18
                };
                await userManager.CreateAsync(userData);
```

```
                await userManager.AddToRolesAsync(userData, new List<string>() {
UserRoles.Student });
            }
            string AdminEmail = "illia.pantiushko@oa.edu.ua";

            if (AdminEmail == payload.Email)
            {
                await userManager.AddToRolesAsync(user, new List<string>() { UserRoles.Admin });
            }


            var roles = await userManager.GetRolesAsync(user);



            var jwtToken = await jwtUtils.GenerateJwtToken(user);

            var refreshToken = jwtUtils.GenerateRefreshToken(ipAddress);

            user.RefreshTokens.Add(refreshToken);

            RemoveOldRefreshTokens(user);

            context.Update(user);

            return new AuthenticateResponseDto(user.UserName, jwtToken, refreshToken.Token,
roles);
        }

        public async Task<AuthenticateResponseDto> RefreshToken(string token, string ipAddress)
        {
            var user = GetUserByRefreshToken(token);

            var refreshToken = user.RefreshTokens.Single(x => x.Token == token);

            if (refreshToken.IsRevoked)
            {
                RevokeDescendantRefreshTokens(refreshToken, user, ipAddress, $"Attempted reuse of
revoked ancestor token: {token}");
                context.Update(user);
            }

            if (!refreshToken.IsActive)
            {
                throw new AppException("Invalid token");
            }

            var newRefreshToken = RotateRefreshToken(refreshToken, ipAddress);

            user.RefreshTokens.Add(newRefreshToken);

            RemoveOldRefreshTokens(user);

            var roles = await userManager.GetRolesAsync(user);

            context.Update(user);

            var jwtToken = await jwtUtils.GenerateJwtToken(user);

            return new AuthenticateResponseDto(user.UserName, jwtToken, newRefreshToken.Token,
roles);
        }
```

```
        private void RemoveOldRefreshTokens(ApplicationUser user)
        {
            user.RefreshTokens.RemoveAll(x =>
                !x.IsActive &&
                x.Created.AddDays(appSettings.RefreshTokenTTL) <= DateTime.UtcNow);
        }

        private void RevokeDescendantRefreshTokens(RefreshToken refreshToken, ApplicationUser
user, string ipAddress, string reason)
        {

            if (!string.IsNullOrEmpty(refreshToken.ReplacedByToken))
            {
                var childToken = user.RefreshTokens.SingleOrDefault(x => x.Token ==
refreshToken.ReplacedByToken);

                if (childToken.IsActive)
                {
                    RevokeRefreshToken(childToken, ipAddress, reason);
                }
                else
                {
                    RevokeDescendantRefreshTokens(childToken, user, ipAddress, reason);
                }
            }
        }

        private ApplicationUser GetUserByRefreshToken(string token)
        {
            var user = context.Users.SingleOrDefault(u => u.RefreshTokens.Any(t => t.Token ==
token));

            if (user == null)
            {
                throw new AppException("Invalid token");
            }
            return user;
        }

        private void RevokeRefreshToken(RefreshToken token, string ipAddress, string reason =
null, string replacedByToken = null)
        {
            token.Revoked = DateTime.UtcNow;
            token.RevokedByIp = ipAddress;
            token.ReasonRevoked = reason;
            token.ReplacedByToken = replacedByToken;
        }

        private RefreshToken RotateRefreshToken(RefreshToken refreshToken, string ipAddress)
        {
            var newRefreshToken = jwtUtils.GenerateRefreshToken(ipAddress);

            RevokeRefreshToken(refreshToken, ipAddress, "Replaced by new token",
newRefreshToken.Token);

            return newRefreshToken;
        }

    }
}
```

GroupCommand.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Threading.Tasks;

namespace QualificationWork.DAL.Command
{
    public class GroupCommand
    {

        private readonly ApplicationContext context;
        private readonly UserManager<ApplicationUser> userManager;

        public GroupCommand(ApplicationContext context, UserManager<ApplicationUser> userManager)
        {
            this.context = context;
            this.userManager = userManager;
        }

        public async Task AddFacultyGroupAsync(long facultyId, string groupName)
        {

            var group = new Group
            {
                GroupName = groupName,
                FacultyId = facultyId
            };

            await context.AddAsync(group);

        }

        public async Task CreateFacultyAsync(FacultyDto model)
        {
            var data = new Faculty
            {
                FacultyName = model.FacultyName
            };

            await context.AddAsync(data);
        }

        public async Task AddGroupSpecialtyAsync(long groupId, string specialtyName)
        {
            var specialty = new Specialty
            {
                SpecialtyName = specialtyName,
                GroupId = groupId
            };

            await context.AddAsync(specialty);

        }
```

```
public void DeleteGroup(long groupId)
{
    var group = context.Groups.FirstOrDefault(m => m.Id == groupId);

    if (group != null)
    {
        context.Remove(group);

    }
}

public async Task AddUserGroup(long groupId, long[] arrUserId)
{
    var group = await context.Groups.FirstOrDefaultAsync(m => m.Id == groupId);

    foreach (var userId in arrUserId) {

        var user = await context.Users.FirstOrDefaultAsync(m => m.Id == userId);

        var checkStudentRole= await userManager.IsInRoleAsync(user, UserRoles.Student);

        if (checkStudentRole) {
            var check = context.UserGroups
                               .Where(x => x.UserId == userId)
                               .FirstOrDefault(m => m.GroupId == groupId);

            if (check == null)
            {
                var userGroup = new UserGroup
                {
                    GroupId = group.Id,
                    UserId = userId
                };

                await context.AddAsync(userGroup);
            }
        }
    }
}

public async Task AddGroupSubject(long groupId, long[] arrSubjectId)
{
    var group = await context.Groups.FirstOrDefaultAsync(m => m.Id == groupId);

    foreach (var subjectId in arrSubjectId)
    {
            var check = context.SubjectGroups
                               .Where(x => x.SubjectId == subjectId)
                               .FirstOrDefault(m => m.GroupId == groupId);

            if (check == null)
            {
                var subjectGroups = new SubjectGroup
                {
                    GroupId = group.Id,
                    SubjectId = subjectId
                };

                await context.AddAsync(subjectGroups);
            }
    }
}
```

```
    }
}
```

SubjectCommand.cs

```
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Command
{
    public class SubjectCommand
    {
        private readonly ApplicationContext context;

        public SubjectCommand(ApplicationContext context)
        {
            this.context = context;
        }

        public async Task CreateSubjectAsync(SubjectDto model)
        {
            var data = new Subject
            {
                SubjectName = model.SubjectName,
                IsActive = model.IsActive,
                AmountCredits = model.AmountCredits,
                SubjectClosingDate = model.SubjectClosingDate,
            };

            await context.AddAsync(data);
        }

        public async Task UpdateSubjectAsync(long subjectId, SubjectDto model)
        {
            var data = await context.Subjects.FirstOrDefaultAsync(m => m.Id == subjectId);

            if (data != null)
            {
                data.SubjectName = model.SubjectName;
                data.IsActive = model.IsActive;
                data.AmountCredits = model.AmountCredits;
                data.SubjectClosingDate = model.SubjectClosingDate;
            }
        }

        public async Task AddLessonAsync(AddLessonDto model)
        {

            var timeTables = await context.TimeTable
                                        .Where(x => x.SubjectId== model.SubjectId)
                                        .Where(x => x.LessonNumber == 1)
                                        .ToListAsync();

            foreach (var userSubject in timeTables)
            {
```

```
                var timeTable = new TimeTable
                {
                    SubjectId = userSubject.SubjectId,
                    UserId =userSubject.UserId,
                    LessonNumber = model.LessonNumber,
                    LessonDate = model.Date.ToUniversalTime(),
                    IsPresent = false,
                    Score = 0
                };
                await context.AddAsync(timeTable);
            }
        }

        public async Task DeleteLessonAsync(int lessonNumber, long subjectId)
        {

            var timeTables = await context.TimeTable
                                .Where(x => x.SubjectId == subjectId)
                                .Where(x => x.LessonNumber == lessonNumber).ToListAsync();

            if (timeTables != null)
            {
                context.RemoveRange(timeTables);
            }
        }

        public void DeleteSubject(long subjectId)
        {
            var subject = context.Subjects.FirstOrDefault(m => m.Id == subjectId);

            if (subject != null)
            {
                context.Remove(subject);

            }
        }
    }
}
```

UserCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using QualificationWork.Middleware;

namespace QualificationWork.DAL.Command
{
    public class UserCommand
    {
        private readonly ApplicationContext context;

        private readonly UserManager<ApplicationUser> userManager;

        public UserCommand(ApplicationContext context, UserManager<ApplicationUser> userManager)
        {
```

```
        this.context = context;
        this.userManager = userManager;
    }

    public async Task AddRoleAsync(string userId, string roleName)
    {
        var user = await userManager.FindByIdAsync(userId);

        if (roleName == "Admin")
        {

            await userManager.AddToRoleAsync(user, UserRoles.Admin);

        }

        else if (roleName == "Teacher")
        {
            await userManager.AddToRoleAsync(user, UserRoles.Teacher);
        }
        else if (roleName == "Student")
        {
            await userManager.AddToRoleAsync(user, UserRoles.Student);
        }
    }

    public async Task DeleteRolesAsync(string userId, string roleName)
    {
        var user = await userManager.FindByIdAsync(userId);

        await userManager.RemoveFromRoleAsync(user, UserRoles.Admin);

        if (roleName == "Admin")
        {
            await userManager.RemoveFromRoleAsync(user, UserRoles.Admin);
        }

        else if (roleName == "Teacher")
        {
            await userManager.RemoveFromRoleAsync(user, UserRoles.Teacher);
        }
        else if (roleName == "Student")
        {
            await userManager.RemoveFromRoleAsync(user, UserRoles.Student);
        }
    }
    public async Task CreateUserAsync(UserDto model)
    {
        var check = await context.Users.FirstOrDefaultAsync(x => x.Email == model.UserEmail);

        if (check != null)
        {
            throw new AppException("User already exists");
        }

            var userData = new ApplicationUser
        {
            Email = model.UserEmail,
            SecurityStamp = Guid.NewGuid().ToString(),
            UserName = model.UserName,
            IsContract = model.IsContract,
            Age = model.Age,
        };
```

```
            await userManager.CreateAsync(userData);
            await userManager.AddToRolesAsync(userData, model.Roles);
        }

        public async Task AddRangeUsers(List<UserFromExcelDto> list)
        {
            foreach (var user in list)
            {
                var check = context.Users.FirstOrDefault(x => x.Email == user.UserEmail);

                if (check==null) {

                    ApplicationUser userData = new ApplicationUser
                    {
                        Email = user.UserEmail,
                        SecurityStamp = Guid.NewGuid().ToString(),
                        UserName = user.UserName,
                        IsContract = user.IsContract,
                        Age = user.Age,
                    };
                    await context.AddAsync(userData);
                }
            }
        }

        public async Task AddGroup(long userId, long groupId)
        {
            var data = new UserGroup
            {
                UserId = userId,
                GroupId = groupId
            };

            var check = context.UserGroups.FirstOrDefault(w => w.UserId == userId && w.GroupId ==
groupId);

            if (check==null)
            {
                await context.AddAsync(data);
            }
        }

        public async Task UpdateUserAsync(EditeUserDto model)
        {
            var data = await context.Users.FirstOrDefaultAsync(m => m.Id == model.Id);

            if (data != null)
            {
                data.UserName= model.UserName;
                data.Email = model.UserEmail;
                data.Age = model.Age;
                data.IsContract = model.IsContract;

                List<string> roles = new List<string>() { UserRoles.Admin, UserRoles.Teacher,
UserRoles.Student };

                await userManager.RemoveFromRolesAsync(data, roles);

                await userManager.AddToRolesAsync(data, model.Roles);
            }
        }
```

```
public void RemoveSubject(long userId, long subjectId)
{
    var data = context.TimeTable
                      .Where(pub => pub.UserId == userId)
                      .FirstOrDefault(pub => pub.SubjectId == subjectId);

    if (data != null)
    {
        context.Remove(data);
    }
}

public async Task CreateTimeTableAsync(TimeTableDto model)
{
    var data = new TimeTable
    {
        UserId = model.UserId,
        SubjectId = model.SubjectId,
        LessonDate = model.LessonDate,
        IsPresent = model.IsPresent,
        LessonNumber = model.LessonNumber,
        Score = model.Score,
    };

    await context.AddAsync(data);
}

public void DeleteUser(long userId)
{
    var user = context.Users.FirstOrDefault(m => m.Id == userId);

    if (user != null)
    {
        context.Remove(user);

    }
}

public async Task UpdateUserScore(UpdateUserScoreDto model)
{
    var data = await context.TimeTable
        .Where(x=>x.LessonNumber==model.LessonNumber)
        .FirstOrDefaultAsync(m => m.UserId == model.Id);

    if (data != null)
    {
        data.Score = model.Score;
    }
}

public async Task UpdateUserIsPresent(UpdateUserIsPresentDto model)
{
    var data = await context.TimeTable
        .Where(x => x.LessonNumber == model.LessonNumber)
        .FirstOrDefaultAsync(m => m.UserId == model.Id);

    if (data != null)
    {
        data.IsPresent = model.IsPresent;
    }
}
```

```
        public async Task AddFacultyGroupAsync(long facultyId, string groupName)
        {
            var group = new Group
            {
                GroupName = groupName,
                FacultyId = facultyId
            };

            await context.AddAsync(group);
            await context.SaveChangesAsync();
        }

        public async Task CreateGroup(CreateGroupDto model)
        {
            var faculty = await context.Faculties.FirstOrDefaultAsync(x => x.FacultyName ==
model.NameFaculty);

            await AddFacultyGroupAsync(faculty.Id, model.NameGroup);
        }

    }

}
```

ДОДАТОК Д

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 7

Острог 2022

ApiResponse.cs

```
using Microsoft.AspNetCore.Identity;
using System;
using System.Linq;
using System.Net;
using System.Text.Json.Serialization;

public static class ApiResponse
{
    public static IApiResponse<TData> Ok<TData>(TData data)
    {
        return new ApiResponse<TData> { Data = data, StatusCode = HttpStatusCode.OK };
    }

    public static IApiResponse Ok()
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.OK };
    }

    public static IApiResponse NotFound(params string[] messages)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.NotFound, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> NotFound<TData>(params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.NotFound, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> NotFound<TData>(TData value, params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.NotFound, Message =
string.Join(Environment.NewLine, messages), Data = value };
    }

    public static IApiResponse BadRequest(params string[] messages)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.BadRequest, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> BadRequest<TData>(params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.BadRequest, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> BadRequest<TData>(TData value, params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.BadRequest, Message =
string.Join(Environment.NewLine, messages), Data = value };
    }

    public static IApiResponse<TData> BadRequest<TData>(IdentityResult result)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.BadRequest, Message =
string.Join(Environment.NewLine, result.Errors.Select(x => x.Description)) };
    }
```

```
    public static IApiResponse BadRequest(IdentityResult result)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.BadRequest, Message =
string.Join(Environment.NewLine, result.Errors.Select(x => x.Description)) };
    }

    public static IApiResponse Conflict(params string[] messages)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.Conflict, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> Conflict<TData>(params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.Conflict, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse Forbidden(params string[] messages)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.Forbidden, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> Forbidden<TData>(params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.Forbidden, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse Unauthorized(params string[] messages)
    {
        return new ApiResponse<object> { StatusCode = HttpStatusCode.Unauthorized, Message =
string.Join(Environment.NewLine, messages) };
    }

    public static IApiResponse<TData> Unauthorized<TData>(params string[] messages)
    {
        return new ApiResponse<TData> { StatusCode = HttpStatusCode.Unauthorized, Message =
string.Join(Environment.NewLine, messages) };
    }
}

public class ApiResponse<TData> : IApiResponse<TData>
{
    public TData Data { get; set; }

    public string Message { get; set; }

    [JsonIgnore]
    public HttpStatusCode StatusCode { get; set; }
}
```

ApiResponseExtention.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Data.Entity.Core.Objects;
using ObjectResult = Microsoft.AspNetCore.Mvc.ObjectResult;

public static class ApiResponseExtension
{
    public static ActionResult ToResult(this IApiResponse response) => new ObjectResult(response)
    {
```

```
        StatusCode = (int)response.StatusCode
    };

    public static ActionResult<T> ToResult<T>(this IApiResponse response) => new
ObjectResult(response)
    {
        StatusCode = (int)response.StatusCode
    };
}
```

ApiSettings.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace QualificationWork.DAL.HelperServise
{
    public class AppSettings
    {
        public string Secret { get; set; }


        public int RefreshTokenTTL { get; set; }
    }
}
```

DBInitializer.cs

```
using Hangfire;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.DAL.HelperServise
{
    public class DBInitializer
    {
        private readonly IRecurringJobManager recurringJobManager;
        private readonly RoleManager<ApplicationRole> roleManager;
        private readonly UserManager<ApplicationUser> userManager;
        private readonly ApplicationContext context;

        public DBInitializer(RoleManager<ApplicationRole> roleManager,
UserManager<ApplicationUser> userManager, ApplicationContext context, IRecurringJobManager
recurringJobManager)
        {
            this.roleManager = roleManager;
            this.context = context;
            this.userManager = userManager;
            this.recurringJobManager = recurringJobManager;
        }

        public async Task SeedAsync()
        {

            await CreateRoles();
```

```
            await CreateAdmin();
            //await CreateUsers();
            await CreateFaculty();
            await CreateSubjects();
            await context.SaveChangesAsync();
            //BackgroundJobCheckingSubject();
        }

        public async Task CreateRoles()
        {
            List<string> roles = new List<string>() { UserRoles.Admin, UserRoles.Teacher,
UserRoles.Student };

            foreach (var role in roles)
            {
                var roleName = await context.Roles.FirstOrDefaultAsync(p => p.Name == role);

                if (roleName == null)
                {
                    await roleManager.CreateAsync(new ApplicationRole { Name = role });
                }
            }
        }

        public async Task CreateAdmin()
        {
            string adminEmail = "illia.pantiushko@oa.edu.ua";

            var admin = await context.Users.FirstOrDefaultAsync(p => p.Email == adminEmail);

            if (admin == null)
            {
                ApplicationUser userData = new ApplicationUser
                {
                    Email = adminEmail,
                    SecurityStamp = Guid.NewGuid().ToString(),
                    UserName = adminEmail,
                };

                await userManager.CreateAsync(userData);

                await userManager.AddToRoleAsync(userData, UserRoles.Admin);
            }
        }

        public async Task CreateFaculty()
        {
            var data = new Faculty
            {
                FacultyName = "Economic",
                Groups = new List<Group>
                {
                  new Group { GroupName = "KN-41" },
                }
            };

            var check = await context.Faculties.FirstOrDefaultAsync(p => p.FacultyName ==
data.FacultyName);

            if (check == null)
            {
                await context.AddAsync(data);
```

```
                await context.SaveChangesAsync();
            }

        }

        public async Task CreateSubjects()
        {
            var listSubjects = new List<Subject>()
            {
                    new Subject{ SubjectName="Компютерні
мережі",AmountCredits=200,IsActive=true,SubjectClosingDate= DateTime.UtcNow},
                    new Subject{ SubjectName="Математичний
аналіз",AmountCredits=300,IsActive=true,SubjectClosingDate= DateTime.UtcNow},
                    new Subject{ SubjectName="Лінійна
алгебра",AmountCredits=250,IsActive=true,SubjectClosingDate= DateTime.UtcNow}
                        };

            foreach (var subject in listSubjects)
            {
                var checkSubject = await context.Subjects.FirstOrDefaultAsync(p => p.SubjectName
== subject.SubjectName);

                if (checkSubject == null)
                {
                    await context.AddAsync(subject);
                }
            }
        }

        public async Task CreateUsers()
        {
            var listUsers = new List<ApplicationUser>()
                    {
                        new ApplicationUser{ UserName="pasha",Email="pasha.pupkin@oa.edu.ua"},
                        new ApplicationUser{ UserName="sofia",Email="sofiia.prusik@oa.edu.ua"},
                        new ApplicationUser{ UserName="dima",Email="dima.kravchuk@oa.edu.ua"},
                    };

            foreach (var user in listUsers)
            {
                var checkUser = await context.Users.FirstOrDefaultAsync(p => p.UserName ==
user.UserName);

                if (checkUser == null)
                {
                    ApplicationUser userData = new ApplicationUser
                    {
                        Email = user.Email,
                        SecurityStamp = Guid.NewGuid().ToString(),
                        UserName = user.UserName,
                    };

                    await userManager.CreateAsync(userData);

                    await userManager.AddToRoleAsync(userData, UserRoles.Student);
                 }
            }
        }

        public async Task CheckSubject()
        {
            var subjects = await context.Subjects.ToListAsync();
```

```
            foreach (var subject in subjects)
            {
                if (subject.SubjectClosingDate > DateTime.Today)
                {
                    subject.IsActive = false;
                }
            }

        }

        public void BackgroundJobCheckingSubject()
        {
            // every day at 9:30
            recurringJobManager.AddOrUpdate("Checking the activity of the subject", () =>
CheckSubject(), "0 30 9 * **");
        }

    }
}
```

IApiResponse.cs

```
using System.Net;
using System.Text.Json.Serialization;

public interface IApiResponse
{
    [JsonIgnore]
    public HttpStatusCode StatusCode { get; }

    public string Message { get; }
}

public interface IApiResponse<out TData> : IApiResponse
{
    public TData Data { get; }
}
```

JwtUtils.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using QualificationWork.DAL.Models;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace QualificationWork.DAL.HelperServise
{
    public class JwtUtils
    {
        private readonly AppSettings appSettings;
        private readonly UserManager<ApplicationUser> userManager;

        public JwtUtils(IOptions<AppSettings> appSettings, UserManager<ApplicationUser>
userManager)
        {
```

```
            this.appSettings = appSettings.Value;
            this.userManager = userManager;
        }

        public async Task<string> GenerateJwtToken(ApplicationUser user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();

            var key = Encoding.ASCII.GetBytes(appSettings.Secret);

            var claims = new List<Claim>();

            claims.Add(new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()));

            var roles = await userManager.GetRolesAsync(user);

            foreach (var i in roles)
            {
                var item = new Claim(ClaimTypes.Role, i);

                claims.Add(item);
            }

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.UtcNow.AddMinutes(60),
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(token);
        }

        public RefreshToken GenerateRefreshToken(string ipAddress)
        {
            using var rngCryptoServiceProvider = new RNGCryptoServiceProvider();

            var randomBytes = new byte[64];

            rngCryptoServiceProvider.GetBytes(randomBytes);

            var refreshToken = new RefreshToken
            {
                Token = Convert.ToBase64String(randomBytes),
                Expires = DateTime.UtcNow.AddDays(7),
                Created = DateTime.UtcNow,
                CreatedByIp = ipAddress
            };

            return refreshToken;
        }
    }
}
```

ДОДАТОК Е

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 6

Острог 2022

GroupQuery.cs

```
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Query
{
    public class GroupQuery
    {
        private readonly ApplicationContext context;

        public GroupQuery(ApplicationContext context)
        {
            this.context = context;
        }

        public List<Group> GetGroups()
        {
            var groups = context.Groups
                            .ToList();
            return groups;
        }

        public List<Faculty> GetFaculty()
        {
            return context.Faculties
                        .ToList();
        }

        public async Task<List<TimeTable>> GetTimeTableByUser(long userId)
        {
            var response = await context.TimeTable.Where(x => x.UserId == userId)
                                        .Include(x => x.Subject)
                                        .ToListAsync();
            return response;

        }

        public async Task<List<TimeTable>> GetTimeTableBySubject(long subjectId)
        {
            var response = await context.TimeTable.Where(x => x.SubjectId == subjectId)
                                        .Include(x => x.User)
                                        .ToListAsync();
            return response;
        }

        public async Task<Pagination<Group>> GetAllGroups(int pageNumber, int pageSize, string search)
        {
```

```
            IQueryable<Group> groups = context.Groups;

            if (!string.IsNullOrEmpty(search))
            {
                groups = groups.Where(e => e.GroupName.ToLower().Contains(search));
            }

            int totalCount = context.Groups.Count();

            var response = await groups.Skip((pageNumber - 1) * pageSize)
                                .Take(pageSize)
                                .Include(pub=>pub.Faculty)
                                .Include(pub => pub.UserGroups)
                                .ThenInclude(pub => pub.User)
                                .ToListAsync();

            return new Pagination<Group>(totalCount, response);
        }

    }
}
```

SubjectQuery.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;

namespace QualificationWork.DAL.Query
{
    public class SubjectQuery
    {
        private readonly ApplicationContext context;

        private readonly UserManager<ApplicationUser> userManager;

        public SubjectQuery(ApplicationContext context, UserManager<ApplicationUser> userManager)
        {
            this.context = context;
            this.userManager = userManager;
        }

        public List<Subject> GetSubjects()
        {
            var data = context.Subjects
                        .ToList();
            return data;
        }
```

```
    public async Task<Pagination<ApplicationUser>> GetAllUsersWithSubjests(int pageNumber, int pageSize, string
search)
    {
        IQueryable<ApplicationUser> users = context.Users;

        if (!string.IsNullOrEmpty(search))
        {
            users = users.Where(e => e.UserName.ToLower().Contains(search)
                                || e.Email.ToLower().Contains(search.ToLower()));
        }

        int totalCount = context.Users.Count();

        var response = await users.Skip((pageNumber - 1) * pageSize)
                                  .Take(pageSize)
                                  .Include(pub => pub.TimeTables.Where(x => x.LessonNumber == 1))
                                  .ThenInclude(pub => pub.Subject)
                                  .Include(pub => pub.UserRoles)
                                  .ThenInclude(pub => pub.Role)
                                  .AsNoTracking()
                                  .ToListAsync();

        return new Pagination<ApplicationUser>(totalCount, response);
    }

    //вивести всіх викладачі в певної групи
    public async Task<List<ApplicationUser>> GetAllTeacherGroups(long groupId)
    {
        var response = await context.Users.Where(x => x.UserGroups.Any(y => y.GroupId == groupId))
                                  .Where(x => x.UserRoles.Any(y => y.Role.Name == UserRoles.Teacher)).ToListAsync();

        return response;
    }

    //вивести всіх викладачі в факультету
    public async Task<List<ApplicationUser>> GetAllTeacherFaculty(long facultyId)
    {
        var response = await context.Users
                              .Where(x => x.UserGroups.Any(y => y.Group.FacultyId == facultyId)).ToListAsync(); ;

        return response;
    }

    //вивести всі групи для яких читається певний предмет
    public async Task<List<Group>> GetAllGroupsBySubject(long subjectId)
    {
        var response = await context.Groups
                              .Where(x => x.SubjectGroups.Any(y => y.SubjectId == subjectId)).ToListAsync();
        return response;
    }

    public async Task<List<Subject>> GetAllSubject(long userId)
    {
        var response = await context.Subjects
                              .Where(x => x.TimeTables.Any(y => y.UserId == userId))
```

```
                        .Include(x => x.TimeTables.Where(y=>y.UserId==userId))
                        .Include(x=>x.TeacherSubjects)
                        .ThenInclude(x=>x.User)
                        .ToListAsync();
            return response;
        }

        public async Task<Pagination<Subject>> GetAllSubjects(int pageNumber, int pageSize, string search)
        {
            IQueryable<Subject> subjects = context.Subjects;

            if (!string.IsNullOrEmpty(search))
            {
                subjects = subjects.Where(e => e.SubjectName.ToLower().Contains(search));
            }

            int totalCount = context.Subjects.Count();

            var response = await subjects.Skip((pageNumber - 1) * pageSize)
                        .Take(pageSize)
                        .AsNoTracking()
                        .ToListAsync();

            return new Pagination<Subject>(totalCount, response);
        }
    }
}
```

UserQuery.cs

```
using Microsoft.EntityFrameworkCore;
using QualificationWork.DAL.Models;
using QualificationWork.DTO.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Query
{
    public class UserQuery
    {
        private readonly ApplicationContext context;


        public UserQuery(ApplicationContext context)
        {
            this.context = context;

        }

        public async Task<List<ApplicationUser>> GetUsers()
        {
```

```
    var data = await context.Users
                .ToListAsync();
    return data;
}

public async Task<ApplicationUser> GetUser(long userId)
{
    var user = await context.Users
                    .Include(x=>x.UserGroups)
                    .ThenInclude(x=>x.Group)
                    .Include(x=>x.UserRoles)
                    .ThenInclude(x => x.Role)
                    .FirstOrDefaultAsync(w => w.Id == userId);

    return user;
}

public async Task<List<TimeTable>> GetTimeTable()
{
    var data = await context.TimeTable
                .ToListAsync();
    return data;
}

////вивести всі предмети викладача та студенті в які належать до предмета
public async Task<List<Subject>> GetAllTeacherSubject(long userId)
{
    var response = await context.Subjects
                        .Where(x=>x.TeacherSubjects.Any(y=>y.UserId==userId))
                        .Include(x => x.TimeTables.Where(x=>x.LessonNumber==1))
                        .ThenInclude(x => x.User)
                        .ToListAsync();
    return response;
}

// time table

// вивести користувачі в по предмету та номеру заняття
public async Task<List<ApplicationUser>> GetUsersTimeTable(long subjectId, int namberleson)
{

    var response = await context.Users
                    .Where(x=>x.TimeTables.Any(x=>x.SubjectId==subjectId))
                    .Include(pub => pub.TimeTables.Where(x=>x.LessonNumber==namberleson))
                    .ToListAsync();
    return response;
}

public async Task<SubjectLessonsDto<TimeTable>> GetSubjectTopic(long subjectId)
{
    var list = new List<TimeTable>();
    var subject = context.Subjects.FirstOrDefault(x => x.Id == subjectId);

    var response = await context.TimeTable
                        .Where(x => x.SubjectId == subjectId)
```

```
                .ToListAsync();
        foreach (var item in response)
        {
            var leson = list.FirstOrDefault(x => x.LessonNumber == item.LessonNumber);

            if (leson == null)
            {
                list.Add(item);
            }
        }

        var result = new SubjectLessonsDto<TimeTable>(subject.SubjectName, list);
        return result;
    }

    public async Task<Subject> GetTimeTableByUser(long subjectId, long userId)
    {
        var data = await context.Subjects
                .Where(x => x.Id == subjectId)
                .Include(x => x.TimeTables.Where(x => x.SubjectId == subjectId && x.UserId == userId))
                .FirstOrDefaultAsync();
        return data;
    }

  }
}
```

ДОДАТОК Є

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 4

Острог 2022

Users.cs

```
using System.Collections.Generic;
using System.Text.Json.Serialization;
using Microsoft.AspNetCore.Identity;

namespace QualificationWork.DAL.Models
{
    public class User : IdentityUser<long>
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
        public bool IsContract { get; set; }
        public string ProfilePicture { get; set; }

        public virtual ICollection<ApplicationUserRole> UserRoles { get; set; }

        public virtual ICollection<UserGroup> UserGroups { get; set; }

        public virtual ICollection<UserSubject> UserSubjects { get; set; }

        [JsonIgnore]
        public List<RefreshToken> RefreshTokens { get; set; }
    }
}
```

UserRoles.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Models
{
    public class UserRoles
    {
        public const string Admin = "Admin";
        public const string Teacher = "Teacher";
        public const string Student = "Student";
    }
```

```
}
```

UserSubjects.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace QualificationWork.DAL.Models
{
    public class UserSubject
    {

        public long Id { get; set; }
        public long UserId { get; set; }
        public virtual ApplicationUser User { get; set; }
        public long SubjectId { get; set; }
        public virtual Subject Subject { get; set; }

        public virtual ICollection<TimeTable> TimeTable { get; set; }
    }
}
```

UserGroups.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace QualificationWork.DAL.Models
{
     public class UserGroup
    {
        public long Id { get; set; }
        public long UserId { get; set; }
        public virtual ApplicationUser User { get; set; }
        public long GroupId { get; set; }
        public virtual Group Group { get; set; }

    }
}
```

SubjectGroup.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace QualificationWork.DAL.Models
{
  public  class SubjectGroup
    {
        public long Id { get; set; }
        public long SubjectId { get; set; }
        public virtual Subject Subject { get; set; }
        public long GroupId { get; set; }
        public virtual Group Group { get; set; }

    }
}
```

Group.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace QualificationWork.DAL.Models
{
    public class Group
    {
        [Key]
        public long Id { get; set; }

        public string GroupName { get; set; }

        public virtual ICollection<Specialty> Specialtys { get; set; }

        public virtual long FacultyId { get; set; }
        public virtual Faculty Faculty { get; set; }

        public virtual ICollection<UserGroup> UserGroups { get; set; }

        public virtual ICollection<SubjectGroup> SubjectGroups { get; set; }


    }
}
```

Faculty.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Models
{
    public class Faculty
    {
        public long Id { get; set; }
        public string FacultyName { get; set; }


        public virtual ICollection<Group> Groups { get; set; }
    }
}
```

TimeTable.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace QualificationWork.DAL.Models
{
    public class TimeTable
    {
        public long Id { get; set; }
        public int LessonNumber { get; set; }
        public bool IsPresent { get; set; }
        public int Score { get; set; }
        public DateTime LessonDate { get; set; }

        public long UserSubjectId { get; set; }
        public virtual UserSubject UserSubject { get; set; }
    }
}
```

ДОДАТОК Ж

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 2

Острог 2022

AuthenticationController.cs

```
using QualificationWork.BL.Services;
using QualificationWork.DTO.Dtos;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace QualificationWork.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        private readonly AuthenticationService authenticationService;

        public AuthenticationController(AuthenticationService authenticationService)
        {
            this.authenticationService = authenticationService;
        }

        [HttpPost("authenticate")]
        public async Task<IActionResult> Authenticate([FromBody] GoogleAuth model)
        {
            var response = await authenticationService.Authenticate(model.googleToken, ipAddress());
            return Ok(response);
        }

        [HttpPost("refresh-token")]
        public async Task<IActionResult> RefreshToken([FromBody] RefreshTokenRequest model)
        {
            var response = await authenticationService.RefreshToken(model.RefreshToken, ipAddress());
            return Ok(response);
        }

        private string ipAddress()
        {
            if (Request.Headers.ContainsKey("X-Forwarded-For"))
            {
                return Request.Headers["X-Forwarded-For"];
            }
            else
            {
                return HttpContext.Connection.RemoteIpAddress.MapToIPv4().ToString();
            }
        }

        public class GoogleAuth
        {
            public string googleToken { get; set; }
        }

        public class RefreshTokenRequest
        {
            public string RefreshToken { get; set; }
        }
```

```
    }
}
```

AuthenticationSevice.cs

```
using QualificationWork.DAL;
using QualificationWork.DAL.Command;
using QualificationWork.DTO.Dtos;
using System.Threading.Tasks;

namespace QualificationWork.BL.Services
{
    public class AuthenticationService
    {
        private readonly AuthenticationCommand authenticationCommand;
        private readonly ApplicationContext context;

        public AuthenticationService(AuthenticationCommand authenticationCommand, ApplicationContext context)
        {

            this.authenticationCommand = authenticationCommand;
            this.context = context;
        }

        public async Task<AuthenticateResponseDto> Authenticate(string accessToken, string ipAddress)
        {
            var authenticate = await authenticationCommand.Authenticate(accessToken, ipAddress);

            await context.SaveChangesAsync();

            return authenticate;

        }
        public async Task<AuthenticateResponseDto> RefreshToken(string token, string ipAddress)
        {
            var refreshToken = await authenticationCommand.RefreshToken(token, ipAddress);

            await context.SaveChangesAsync();

            return refreshToken;
        }

    }
}
```

ДОДАТОК З

Система контролю відвідуваності занять у навчальних закладах

Лістинг програми

Аркушів 10

Острог 2022

Store.js

```
import { combineReducers } from 'redux';
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';

import AuthReducer from './Auth-reducer';
import AdminReducer from './Admin-reducer';
import TeacherReducer from './Teacher-reducer';
import ProfileReducer from './Profile-reducer';

const rootReducer = combineReducers({
  Auth: AuthReducer,
  ProfilePage: ProfileReducer,
  AdminPage: AdminReducer,
  TeacherPage: TeacherReducer,
});

export const store = createStore(rootReducer,
composeWithDevTools(applyMiddleware(thunk)));
```

Admin-reducer.js

```
const SET_USERS = 'SET_USERS';
const DELETE_USER = 'DELETE_USER';
const SET_GROUPS = 'SET_GROUPS';
const ADD_USER = 'ADD_USER';
const ADD_USERS = 'ADD_USERS';
const ADD_LIST_SUBJECTS = ' ADD_LIST_SUBJECTS';
const UPDATE_USER = 'UPDATE_USER';
const DELETE_USER_ROLE = 'DELETE_USER_ROLE';
const ADD_GROUP = 'ADD_GROUP';
const DELETE_GROUP = ' DELETE_GROUP';

let initialState = {
  users: [],
  usersTotalCount: null,
  isFetchingUsers: true,
  groups: [],
  groupsTotalCount: null,
  isFetchingGroups: true,
  subjects: [],
};

const AdminReducer = (state = initialState, action) => {
  switch (action.type) {
```

```
    case SET_USERS:
      return {
        ...state,
        users: action.payload.users,
        usersTotalCount: action.payload.totalCount,
        isFetchingUsers: false,
      };

    case SET_GROUPS:
      return {
        ...state,
        groups: action.payload.groups,
        groupsTotalCount: action.payload.totalCount,
        isFetchingGroups: false,
      };
    case ADD_USER:
      return {
        ...state,
        users: [...state.users, action.payload],
      };

    case ADD_GROUP:
      return {
        ...state,
        groups: [...state. groups, action.payload],
      };
    case ADD_USERS:
      return {
        ...state,
        users: [...state.users, ...action.payload],
      };
    case DELETE_USER:
      return {
        ...state,
        users: [...state.users.filter((user) => user.id !== action.payload)],
      };

    case DELETE_GROUP:
      return {
        ...state,
        groups: [...state.groups.filter((group) => group.id !== action.payload)],
      };

    case UPDATE_USER:
      const newUserData = state.users.map((item) => {
        if (item.id === action.payload.id) {
          return {
            ...item,
            userName: action.payload.userName,
```

```
            userEmail: action.payload.userEmail,
            age: action.payload.age,
            isContract: action.payload.isContract,
            userRoles: action.payload.roles,
          };
        }
        return item;
      });
      return { ...state, users: newUserData };

    case DELETE_USER_ROLE:
      const deleteUserRole = state.users.map((item) => {
        if (item.id === action.data.id) {
          return {
            ...item,
            userRoles: [...item.userRoles.filter((i) => i.role.id !==
action.data.roleId)],
          };
        }
        return item;
      });
      return { ...state, users: deleteUserRole };
    case ADD_LIST_SUBJECTS:
      return {
        ...state,
        subjects: action.payload,
      };

    default:
      return state;
  }
};

export const setUsers = (users, totalCount) => {
  return {
    type: SET_USERS,
    payload: { users, totalCount },
  };
};

export const setGroups = (groups, totalCount) => {
  return {
    type: SET_GROUPS,
    payload: { groups, totalCount },
  };
};

export const addUser = (data) => {
  return {
```

```
    type: ADD_USER,
    payload: data,
  };
};

export const addGroup = (data) => {
  return {
    type: ADD_GROUP,
    payload: data,
  };
};

export const updateUser = (userData) => {
  return {
    type: UPDATE_USER,
    payload: userData,
  };
};

export const addUsers = (users) => {
  return {
    type: ADD_USERS,
    payload: users,
  };
};

export const deleteUser = (id) => {
  return {
    type: DELETE_USER,
    payload: id,
  };
};

export const deleteGroup = (id) => {
  return {
    type: DELETE_GROUP,
    payload: id,
  };
};

export const deleteUserRole = (data) => {
  return {
    type: DELETE_USER_ROLE,
    data,
  };
};

export const setListSubjects = (subjects) => {
  return {
```

```
    type: ADD_LIST_SUBJECTS,
    payload: subjects,
  };
};

export default AdminReducer;
```

Auth-reducer.js

```
const SET_USER_DATA = 'SET_USER_DATA';
const SET_REDIRECT = 'SET_USER_DATA';
const LOGOUT = 'LOGOUT';

let initialState = {
  name: !localStorage.getItem('name') ? false : localStorage.getItem('name'),
  roles: !localStorage.getItem('roles') ? [] :
JSON.parse(localStorage.getItem('roles')),
  isAuth: !localStorage.getItem('token') ? false : true,
  redirectTo:"/"
};

const AuthReducer = (state = initialState, action) => {
  switch (action.type) {
    case SET_USER_DATA:
      return {
        ...state,
        ...action.payload,
      };

    case LOGOUT:
      return {
        ...state,
        ...action.payload,
      };

    default:
      return state;
  }
};

export const setAuthUserData = (name, role, isAuth, redirectTo) => {
  return {
    type: SET_USER_DATA,
    payload: { name, role, isAuth, redirectTo},
  };
};
```

```
export const logout = (name, role, isAuth) => {
  return {
    type: LOGOUT,
    payload: { name, role, isAuth },
  };
};

export default AuthReducer;
```

Profile-reducer.js

```
const SET_SUBJECTS = 'SET_SUBJECTS';
const SET_SUBJECT_DETAILS = 'SET_SUBJECT_DETAILS';
const SET_USER_INFO = 'SET_USER_INFO';
const SET_SUBJECTS_FETCHING = 'SET_SUBJECTS_FETCHING';

let initialState = {
  profile: null,
  subjects: [],
  isFetchingSubjects: true,
  subjectDetails: [],
  isFetching: true,
};

const ProfileReducer = (state = initialState, action) => {
  switch (action.type) {
    case SET_SUBJECTS:
      return {
        ...state,
        subjects: action.payload,
        isFetchingSubjects: false,
      };
    case SET_USER_INFO:
      return {
        ...state,
        profile: action.payload,
        isFetching: false,
      };
    case SET_SUBJECT_DETAILS:
      return {
        ...state,
        subjectDetails: action.payload,
      };
    case SET_SUBJECTS_FETCHING:
      return {
        ...state,
        isFetchingSubjects: true,
```

```
      };
    default:
      return state;
  }
};

export const setSubjects = (users) => {
  return {
    type: SET_SUBJECTS,
    payload: users,
  };
};

export const setUserInfo = (userInfo) => {
  return {
    type: SET_USER_INFO,
    payload: userInfo,
  };
};

export const setSubjectDetails = (data) => {
  return {
    type: SET_SUBJECT_DETAILS,
    payload: data,
  };
};

export const setSubjectFetching = () => {
  return {
    type: SET_SUBJECTS_FETCHING,
  };
};

export default ProfileReducer;
```

Teacher-reducer.js

```
const SET_SUBJECTS = 'SET_SUBJECTS';
const SET_SUBJECT_LESONS = 'SET_SUBJECT_LESONS';
const SET_ATTENDANCE_LIST = 'SET_ATTENDANCE_LIST';
const UPDATE_USER_SCORE = 'UPDATE_USER_SCORE';
const UPDATE_USER_IS_PRESENT = 'UPDATE_USER_IS_PRESENT';
const ADD_NEW_LESSON = 'ADD_NEW_LESSON';
const DELETE_LESSON = 'DELETE_LESSON';

let initialState = {
  subjects: [],
```

```
  isFetchingSubjects: true,
  subjectName:'',
  subjectLesons: [],
  attendanceList: [],
  isFetching: false,
};

const TeacherReducer = (state = initialState, action) => {
  switch (action.type) {
    case SET_SUBJECTS:
      return {
        ...state,
        subjects: action.payload,
        isFetchingSubjects: false,
      };

    case SET_SUBJECT_LESONS:
      return {
        ...state,
        subjectName: action.payload?.subjectName,
        subjectLesons: action.payload?.lessons,
        isFetching: false,
      };
    case ADD_NEW_LESSON:
      return {
        ...state,
        subjectLesons: [...state.subjectLesons, action.payload],
      };
    case DELETE_LESSON:
      return {
        subjectLesons: [
          ...state.subjectLesons.filter((lesson) => lesson.lessonNumber !==
action.payload),
        ],
      };
    case SET_ATTENDANCE_LIST:
      return {
        ...state,
        attendanceList: action.payload,
        isFetching: false,
      };
    case UPDATE_USER_SCORE:
      const newScore = state.attendanceList.map((item) => {
        if (item.id === action.row.id) {
          return {
            ...item,
            score: action.row.score,
          };
        }
```

```
        return item;
      });
      return { ...state, attendanceList: newScore };
    case UPDATE_USER_IS_PRESENT:
      const newIsPresent = state.attendanceList.map((item) => {
        if (item.id === action.row.id) {
          return {
            ...item,
            isPresent: action.row.isPresent,
          };
        }
        return item;
      });
      return { ...state, attendanceList: newIsPresent };
    default:
      return state;
  }
};

export const setSubjects = (subjects) => {
  return {
    type: SET_SUBJECTS,
    payload: subjects,
  };
};

export const setSubjectLesons = (subjectLeson) => {
  return {
    type: SET_SUBJECT_LESONS,
    payload: subjectLeson,
  };
};

export const addNewLesson = (data) => {
  return {
    type: ADD_NEW_LESSON,
    payload: data,
  };
};

export const setAttendanceList = (attendanceList) => {
  return {
    type: SET_ATTENDANCE_LIST,
    payload: attendanceList,
  };
};

export const updateUserScore = (row) => {
  return {
```

```
    type: UPDATE_USER_SCORE,
    row,
  };
};

export const updateUserIsPresent = (row) => {
  return {
    type: UPDATE_USER_IS_PRESENT,
    row,
  };
};

export const deleteLesson = (lessonNumber) => {
  return {
    type: DELETE_LESSON,
    payload: lessonNumber,
  };
};

export default TeacherReducer;
```