НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Приладобудівний факультет

Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

До захисту допущено:

Завідувач кафедри

_____ Надія БУРАУ

«___»_____20__ р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньою програмою «*Комп'ютерно-інтегровані системи та технології в приладобудуван*ні»

спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

на тему: «Автоматизація супроводу навчального процесу»

Виконав:
студент IV курсу, групи ПГ-01
Боровський Олександр Максимович _____

Керівник:

К.т.н., доцент Цибульник С.О. _____

Рецензент:

_____ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Приладобудівний факультет
Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»
Освітньо-професійна програма «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

ЗАТВЕРДЖУЮ
Завідувачка кафедри
_____ Н.І. Бурау
«___»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Боровському Олександру Максимовичу

1. Тема роботи «Автоматизація супроводу навчального процесу», керівник роботи Цибульник Сергій Олексійович, к.т.н., доцент, затверджені наказом по університету від «___»_____ 20__ р. №_____
2. Термін подання студентом роботи 10.06.2024р.
3. Вихідні дані до роботи: а) автоматизована система має давати змогу створювати користувачів з різними правами доступу, наприклад, «викладач», «студент» та «адміністратор»; б) автоматизована система має давати змогу користувачам з роллю «викладач» створювати та керувати навчальними дисциплінами; в) автоматизована система має давати змогу користувачам з роллю «викладач» переглядати розклад, інформацію про навчальні групи, студентів та навчальні дисципліни, отримувати повідомлення про початок занять; г) автоматизована система має давати змогу

користувачам з роллю «викладач» створювати списки навчальних дисциплін, студентів (навчальних груп).

4. Зміст роботи:

1) Огляд стану проблеми.

2) Огляд та вибір технологій для реалізації автоматизованої системи.

3) Формування та формалізація вимог до системи.

4) Реалізація автоматизованої системи.

5) Тестування розробленої системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): таблиці, графіки, рисунки, тощо.

6. Дата видачі завдання 06.03.2024р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
	Огляд стану проблеми	20.03.2024	
	Огляд та вибір технологій для реалізації автоматизованої системи	03.04.2024	
	Формування та формалізація вимог до системи	17.04.2024	
	Реалізація автоматизованої системи	22.05.2024	
	Тестування розробленої системи	01.06.2024	
	Оформлення пояснювальної записки	10.06.2024	

Студент Олександр БОРОВСЬКИЙ

Керівник Сергій ЦИБУЛЬНИК

АНОТАЦІЯ

Робота містить 84 сторінок, 52 рисунка, 1 додаток та 40 посилань.

У дипломній роботі розглядається питання автоматизації супроводу навчального процесу у вищих навчальних закладах. Метою дослідження є розробка автоматизованої системи, яка покращує ефективність та зручність управління освітнім процесом для викладачів та студентів. Актуальність теми обумовлена зростаючою потребою у сучасних технологіях для організації навчання, зокрема у контексті дистанційної освіти, що стала необхідністю через пандемію COVID-19.

У процесі роботи були досліджені існуючі системи управління навчанням (LMS), такі як Google Classroom, Canvas, Schoology та інші, і виявлено їхні недоліки, зокрема відсутність спеціалізованих функцій для викладачів. На основі аналізу було сформульовано вимоги до нової системи, яка передбачає автоматизацію ряду рутинних завдань, таких як створення розкладів, управління навчальними дисциплінами та групами студентів, що дозволяє зменшити витрати часу та підвищити точність виконання завдань.

У роботі детально описано процес розробки автоматизованої системи, включаючи вибір технологій для бекенду та фронтенду, розробку графічного інтерфейсу та реалізацію основних функцій. Особлива увага приділяється тестуванню розробленої системи, що включає перевірку надійності, продуктивності та безпеки. Результати тестування підтвердили високу ефективність та надійність запропонованого рішення.

Впровадження розробленої системи дозволить значно покращити організацію навчального процесу, забезпечивши викладачів зручними інструментами для управління освітнім процесом та зменшення навантаження на адміністративний персонал.

Ключові слова: автоматизація, навчальний процес, система управління навчанням, LMS, розклад занять, дистанційна освіта.

ABSTRACT

The work consists of 84 pages, 52 figures, 1 appendix, and 40 references.

This thesis addresses the automation of the educational process in higher educational institutions. The main objective of the research is to develop an automated system that enhances the efficiency and convenience of managing the educational process for both teachers and students. The relevance of the topic is driven by the increasing need for modern technologies in education, especially in the context of distance learning, which became essential during the COVID-19 pandemic.

The study explores existing Learning Management Systems (LMS) such as Google Classroom, Canvas, Schoology, and others, identifying their shortcomings, particularly the lack of specialized functions for teachers. Based on this analysis, requirements for a new system were formulated. The proposed system aims to automate various routine tasks, including the creation of schedules, management of academic disciplines, and student groups, thereby reducing time expenditure and improving task accuracy.

The thesis provides a detailed description of the development process of the automated system, including the selection of technologies for both backend and frontend, the design of the graphical interface, and the implementation of key functionalities. Special attention is given to testing the developed system, focusing on reliability, performance, and security. The results of the testing confirmed the high efficiency and reliability of the proposed solution.

The implementation of the developed system will significantly improve the organization of the educational process by providing teachers with convenient tools for managing educational activities and reducing the administrative burden.

Keywords: automation, educational process, Learning Management System, LMS, class scheduling, distance education.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

LMS - Системи управління навчанням

ПЗ - Програмне забезпечення

HTML - HyperText Markup Language

ООП - Об'єктно-орієнтоване програмування

IDE - Integrated Development Environment

UI - User interface

UX - User experience

CSS - Cascading Style Sheets

SQL - Structured query language

NoSQL – No Structured query language

JSON - JavaScript Object Notation

iOS – iPhone operation system

HDRP - High Definition Render Pipeline

URP - Universal Render Pipeline

ID - Data name

ВСТУП

З розвитком сучасних технологій традиційні методи організації навчання такі як паперові журнали чи щоденники стрімко втрачають свою актуальність через низький рівень автоматизації. Все більше освітніх закладів перекладають частину відповідальності за організацію навчального процесу на платформи менеджменту навчанням (LMS – Learning Management System). Такі системи дозволяють автоматизувати значну частину рутинної роботи, яка раніше виконувалася викладачами та іншими учасниками організації навчального процесу власноруч, що дозволяє значно зменшити часові та економічні витрати установи. Окрім економічних та часових переваг, системи менеджменту навчанням також виключають можливість появ помилок спричинених через вплив людського фактору, оскільки більшість процесів є автоматизованими, відповідно й можливість помилок спричинених через людський фактор значно зменшується. В сучасних реаліях системи менеджменту навчанням стали невід'ємною частиною будь-якої навчальної установи. Вони дозволяють значно зменшити часові та економічні витрати, а також збільшити швидкість та точність при організації навчальних процесів.

Не зважаючи на широкий спектр LMS на ринку, не було виявлено жодної системи, яка була б розроблена спеціально для викладачів та могла б покрити їх базові рутинні потреби такі як формування розкладу викладача з динамічним повторенням занять, можливість вносити зміни до розкладу викладача або можливість переглянути список групи призначеної в розкладі. Відсутність такої системи призводить до ряду проблем в організації навчання із сторони викладачів, що в свою чергу змушує викладачів використовувати окремі додаткові програми із конкретним функціоналом, що не надає бажаного рівня зручності та автоматизації. Як результат більшість із існуючих LMS не є достатньо зручними для викладачів. Саме тому було ухвалено рішення створити LMS, яка дозволить викладачам в одному додатку автоматизувати значну частину рутинної роботи, яка стосується організації та супроводу навчально процесу.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ LMS ТА ВИЯВЛЕННЯ ЇХ НЕДОЛІКІВ

1.1 Організація освітнього процесу

Організація освітнього у всі часи була складною системою, яка відіграє ключову роль в досягненні навчальних цілей. Вона забезпечує ефективне і структуроване навчання. Вона включає в себе повний процес супроводу студентів починаючи з планування навчальних програм і закінчуючи оцінюванням знань студентів. Одним з ключових елементів цієї системи є створення навчальних журналів та розкладу занять. Кожен навчальний заклад, будь то школа, коледж або університет, обов’язково має розклад для студентів та викладачів. Це невід'ємна частина навчального процесу, без якої неможливе ефективне навчання. Розклад забезпечує структурованість і систематичність освітнього процесу, дозволяючи всім учасникам навчання знати, коли і де відбуваються заняття.

1.2 Традиційні методи організації навчання

Традиційні методи організації освітнього процесу, вже давно знаходяться у використані та зберігають свою актуальність. Паперові журнали є основним із них відомих традиційних інструментів в освіті. Вони використовуються для фіксації інформації, такої як оцінки або відвідуваність. Ці журнали, хоча і мають свої недоліки, залишаються надійними джерелами інформації, особливо в умовах технічних збоїв або відсутності доступу до інтернету [1].

Ще одним традиційним методом є ручне складання розкладу. Розклад навчальних занять є документом, який регламентує академічну роботу студентів і викладачів. Він складається з урахуванням вимог навчального плану, особливостей учасників навчального процесу та можливостей навчально-матеріальної бази. Хоча

це може зайняти багато часу, ручне складання розкладів дозволяє адміністрації школи мати повний контроль над організацією навчального процесу [2].

Згідно з традиційними способами організації навчально процесу, після формування розкладу студенти переписували його собі для навігації.

Традиційні методи організації навчання мають ряд переваг, таких як надійність та простота використання, та навіть у час розвитку цифрових технологій, вони залишаються важливою складовою освітнього процесу і частково продовжують використовуватися в сучасних навчальних закладах.

1.3 Недоліки традиційних методів організації навчання

Раніше процес організації навчання здебільшого здійснювався вручну. Викладачі та адміністрація витрачали багато часу на складання розкладів, ведення журналів та облік відвідуваності. Цей підхід мав масу недоліків.

Значний час витрачався на заповнення паперових документів, журналів. Студенти та викладачі були змушені самі записувати розклад, що також призводило до нераціональних витрат часу з обох сторін.

Традиційнийй підхід до організації мав значний ризик помилок, через людський фактор, який міг призвести до внесення хибної інформації у розклад або оцінювання.

Ще однією значно проблемою стала відсутність оперативності. Зміни в розкладі могли не доходити до всіх учасників навчального процесу вчасно, відповідно деякі люди просто не знали про перенесення заняття на новий час або про появу чергового вихідного.

1.4 Сучасні підходи до організації навчання

З початком карантину через COVID-19 багато навчальних закладів були змушені перейти на дистанційну форму навчання. Це стало вимушеним кроком для забезпечення безпеки студентів та викладачів. Перехід на дистанційне навчання призвів до значних змін в організації навчального процесу та сприяв виникненню

нових онлайн шкіл чи академій які також мають потребу в правильній організації навчального процесу.

Пандемія змусила більшість традиційних навчальних закладів швидко адаптуватися до нових умов, перейшовши на використання онлайн-платформ для проведення занять. Це включає платформи для відеоконференцій, такі як Zoom [3] чи Google Meet [4], а також системи управління навчанням, такі як Moodle [5], Canvas [6] і Blackboard [7]. Завдяки цим інструментам стало можливим забезпечити безперервність навчального процесу навіть під час суворих карантинних обмежень.

У цих умовах особливо важливою стала потреба в якісних системах для розкладу та організації навчального процесу. Перехід на дистанційне навчання вимагав створення гнучких і надійних інструментів, які могли б забезпечити ефективне планування і координацію занять. Такі системи дозволяють підтримувати чіткий графік занять, уникати конфліктів у розкладі і забезпечувати своєчасне інформування студентів та викладачів про зміни.

Сучасні підходи до організації навчання значно змінилися завдяки впровадженню цифрових технологій та програмного забезпечення. Використання різних додатків і програм стало не лише зручним, але й необхідним для ефективного управління навчальним процесом. Це особливо актуально у контексті дистанційного навчання.

1.5 Актуальність CRM програм

Системи управління навчанням (LMS) стали необхідним інструментом для організації та ведення навчальних програм у навчальних закладах. Ці програми розроблені для вирішення різних завдань, починаючи від складання розкладу і завершуючи веденням студентських журналів і оцінюванням успішності. Велика кількість LMS програм вже існує на ринку, і кожна з них має свої як сильні так і слабкі сторони.

У сучасному освітньому середовищі існує велика кількість різних LMS програм, розроблених як великими корпораціями, так і невеликими стартапами. Деякі

з них є загальновживаними та мають велику кількість користувачів, такі як Classroom, Blackboard і Canvas, в той час як інші можуть бути спеціалізованими на конкретних сегментах ринку, наприклад, системи для дистанційного навчання або онлайн-курсів. Основні переваги LMS систем [8]:

- Системи управління навчанням (LMS) зменшують витрати на друк та розповсюдження матеріалів, оскільки дозволяють централізовано зберігати та розповсюджувати навчальні ресурси.
- LMS надають викладачам інструменти для відстеження та оцінювання прогресу студентів.
- Автоматизовані процеси оцінювання спрощують роботу викладачів і забезпечують послідовність та об'єктивність.
- LMS забезпечують централізоване розміщення навчальних матеріалів, що спрощує доступ до них для всіх учасників навчання.
- Оновлення курсів дозволяють зберігати актуальність навчальних матеріалів та підвищувати мотивацію працівників до професійного розвитку.
- Інтерактивність навчання через використання різноманітних мультимедійних елементів робить процес більш захопливим.

LMS пропонують багато переваг, але мають і деякі незначні недоліки, такі як труднощі з технічною підтримкою чи доступом до інтернету, які можуть впливати на якість організації навчання.

1.5 Огляд існуючих LMS програм для організації навчання

1.5.1 Огляд Google classroom

На сьогоднішній день існує багато додатків, які можуть допомогти організувати навчання, кожен із них має свої особливості. Одним із найпопулярніших вважають Google classroom.

Для користувачів Google classroom може бути представлений у вигляді онлайн-платформи або додатку, який надає викладачам і студентам можливість співпрацювати у навчальному процесі.

Розглянемо основні функції Google classroom [9]:

- Створення класів або курсів, їх налаштування відповідно до певних потреб і навчальних цілей.
- Організація запису учнів на курси, надання легкого доступу до навчальних матеріалів та активностей.
- Поширення різних матеріалів, включаючи лекції, презентації, відео та додаткові ресурси.
- Створення завдань для учнів, використовуючи різноманітні формати, такі як тести, есе, проекти та інтерактивні вправи.
- Оцінювання виконаних завдань учнів, можливість надавати детальний зворотний зв'язок і відстежувати прогрес виконання завдань.

Google classroom забезпечує зручний та ефективний спосіб організації навчального процесу як для викладачів, так і для студентів. Він створює віртуальне середовище, де можливо не лише передавати знання, а й сприяти взаємодії та спільному навчанню.

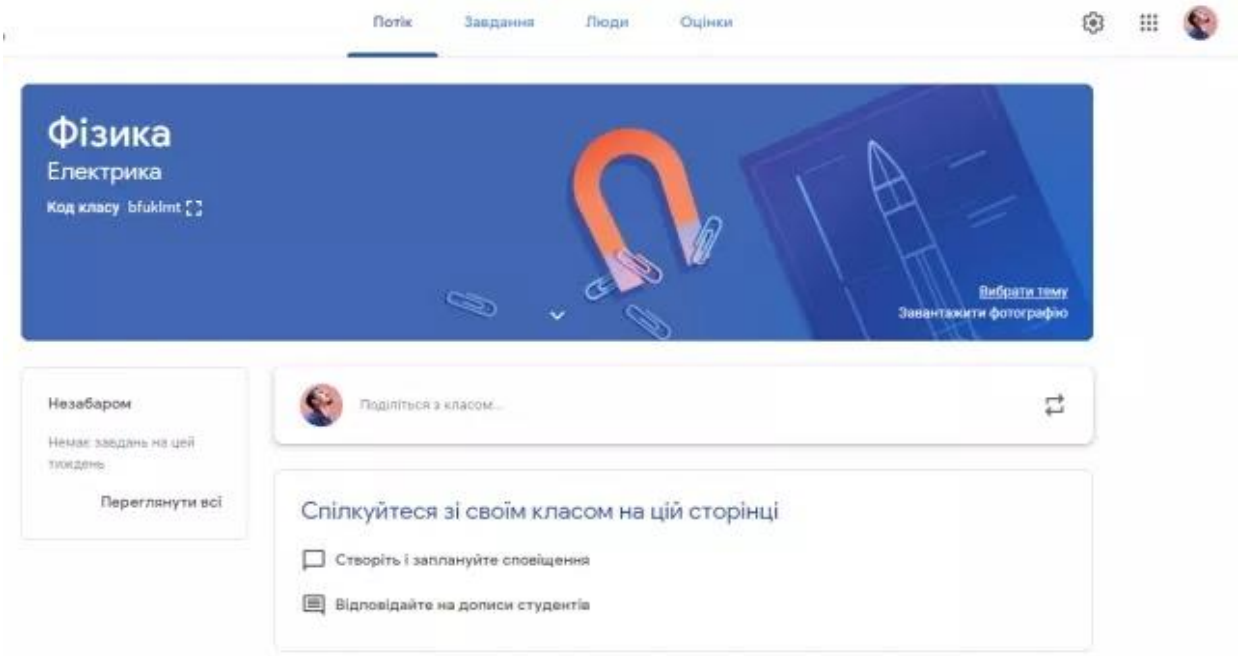

Рис. 1.1 Загальний вигляд та можливості Google classroom [9]

Проте одним із найбільших недоліків Google classroom є відсутність повноцінної системи розкладу. Викладач може призначити дату виконання роботи, але побачити це можна тільки якщо напряму зайти до конкретного завдання в Classroom або отримавши лист на електронну пошту, що є не дуже зручним та сучасним рішенням та в свою чергу може збільшити вірогідність учня пропустити завдання. Тож великою проблемою є відсутність загальної дошки із завданнями на найближчий час, та для отримання всієї інформації доводиться шукати її самому відкриваючи всі класи по черзі, що свідчить про низький рівень автоматизації.

1.5.2 Огляд Canvas

Canvas - це платформа для управління навчанням в онлайн-середовищі, яка використовується навчальними закладами, викладачами і студентами. Вона надає різноманітні інструменти для створення та управління курсами, аналізу та відстеження статистики успішності користувачів, а також можливості внутрішньої комунікації.

Розглянемо основні функції Canvas[10]:

- Управління курсами - Canvas дозволяє викладачам створювати курси з різноманітними модулями, завданнями та тестами. Інтерфейс інтуїтивно зрозумілий, що полегшує процес створення і управління курсами.
- Електронні журнали - викладачі можуть вести електронні журнали, оцінювати роботи студентів і надавати зворотний зв'язок. Студенти мають можливість бачити свої оцінки і отримувати коментарі до своїх завдань.
- Спілкування та співпраця - Canvas підтримує обговорення, повідомлення та відеоконференції. Це дозволяє викладачам і студентам легко взаємодіяти та співпрацювати.
- Аналітика і звітність - платформа надає інструменти для аналітики та звітності, що дозволяють викладачам відстежувати прогрес студентів і оцінювати ефективність навчальних програм.

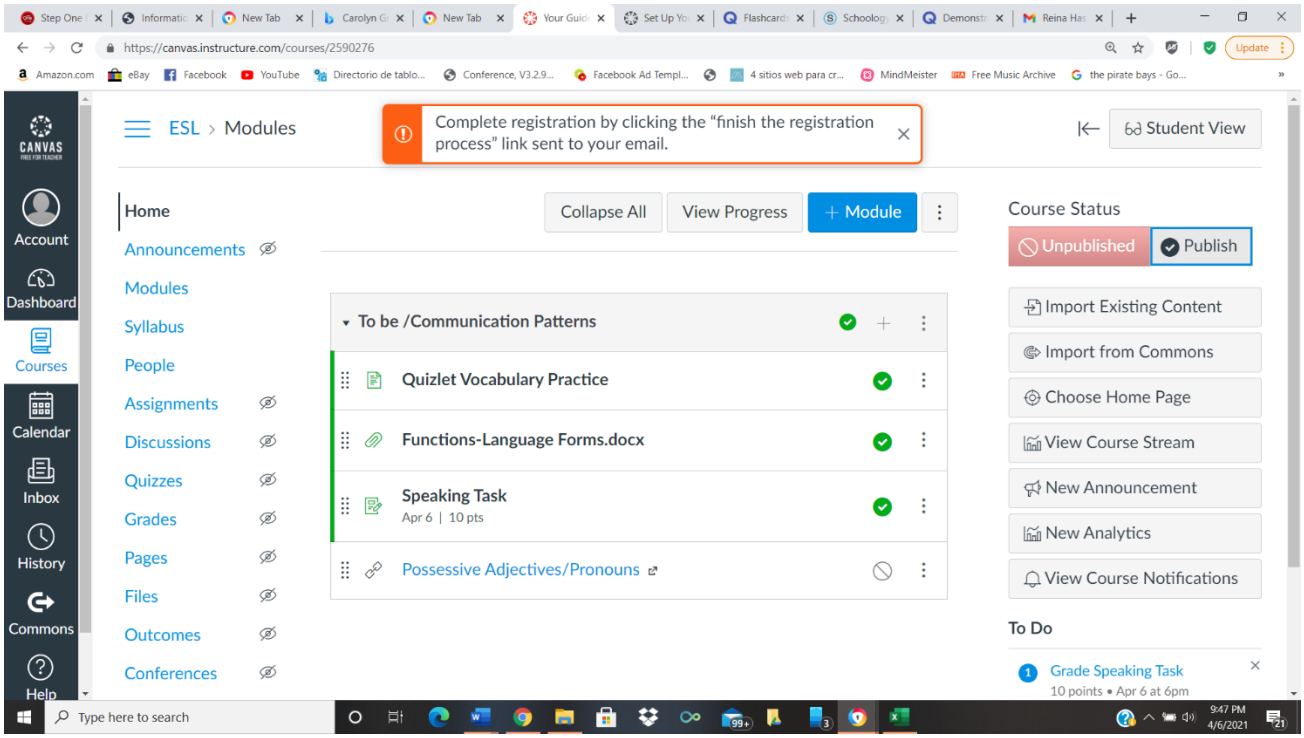

Рис. 1.2 Загальний вигляд та можливості Canvas [11]

Canvas є потужною системою управління навчанням, яка надає велику кількість інструментів для організації і підтримки навчального процесу. Використання Canvas може значно поліпшити якість навчання та все ж не забезпечує повноцінний інструмент управління розкладом занять. Це може створювати незручності для викладачів, які повинні використовувати додаткові засоби для планування свого часу.

1.5.3 Огляд Schoology

Schoology є популярною системою управління навчальним процесом, яка використовується у багатьох навчальних закладах для організації освітнього процесу, співпраці між викладачами та студентами та управління навчальними матеріалами. Розглянемо основні функції Schoology [12]:

- Створення папок, маркування матеріалів і використання пошуку для швидкого доступу до потрібних ресурсів підвищує ефективність навчання.

- Зв'язок між студентами та викладачами за допомогою приватних повідомлень та форумів.
- Створення та оцінювання завдань вчителями.
- Можливість відстежувати прогрес учнів та надавати їм зворотний зв'язок щодо їх продуктивності.

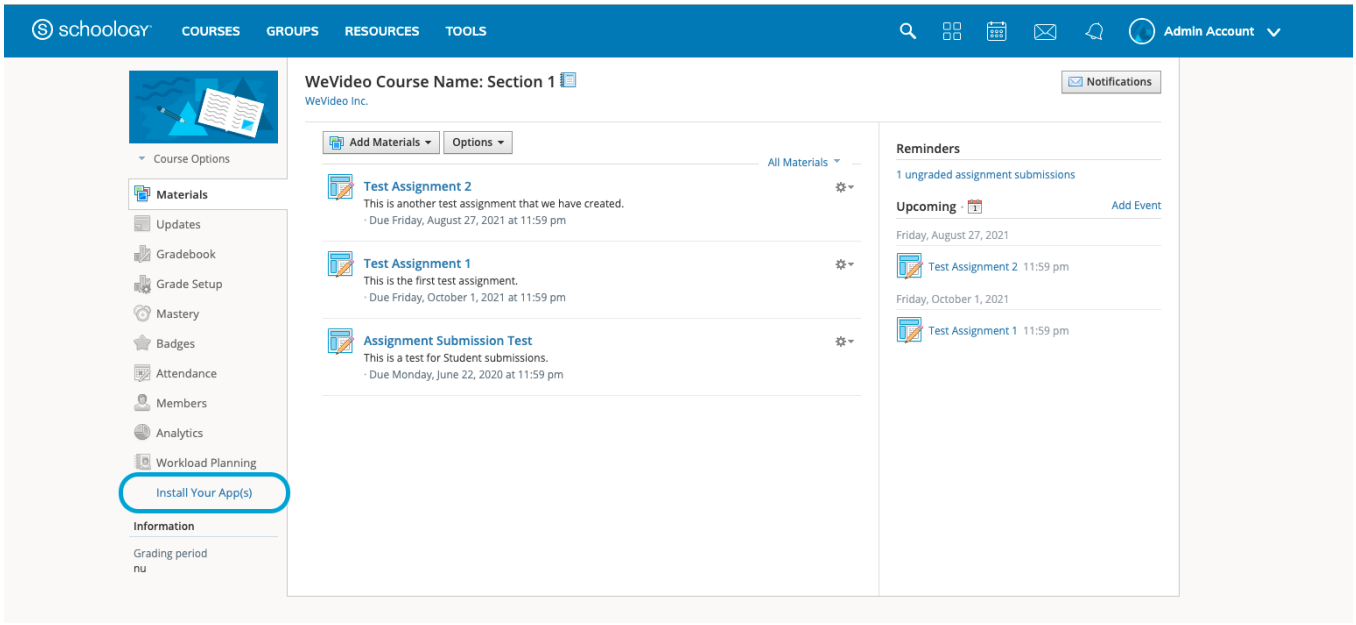

Рис 1.3 Загальний вигляд та можливості Schoology [13]

Schoology є потужною і гнучкою системою управління навчанням, яка надає широкий великий вибір функціоналу для організації освітнього процесу, та гнучкість у вигляді поєднання із іншими платформами. Що до розкладу маємо ту ж саму проблему, що й на інших платформах, немає можливості відслідковувати всі уроки разом в зручній системі розкладу.

1.5.4 Огляд Notion

Провівши аналіз декількох найпопулярніших в своєму колі LMS програм, було отримано спільну проблему. Всі програми використовуються для організації навчання, та жодна з них не має системи розкладу, що змушує користувачів використовувати додаткові програми, та відслідковувати всі завдання окремо.

Зважаючи на цю проблемо проаналізуємо Notion [14], платформу яка більш функціональною, та робить опір на прив’язку до розкладу.

Notion є багатофункціональним інструментом для організації та управління різними видами інформації, включаючи навчальні процеси. Хоча Notion не є типовою системою управління навчанням, його гнучкість та можливості роблять його популярним закладів вищої освіти для організації курсів, спільної роботи та управління завданнями.

Розглянемо основні функції Notion [15]:

- Можливість редагування даних у різних форматах, включаючи тексти, списки, коди, зображення та інші.
- Швидке та легке перетворення або переміщення даних з одного формату у інший.
- Можливість зберігання та впорядкування різноманітних медіа-файлів, таких як фотографії, відео або таблиці, у своїх документах.
- Дорожні карти, календарі та планувальники на тиждень для кращого управління часом та завданнями.
- Можливість організації та зберігання інформації з будь-яких сфер у вигляді структурованих баз даних.
- Посилання на інші сторінки для створення взаємозв'язаних документів.
- Можливість створення та редагування документів та заміток у реальному часі разом з іншими користувачами.
- Можливість створення таблиць для простих операцій, що може замінити використання Excel.
- Для складніших обчислень може знадобитися додаткове програмне забезпечення.

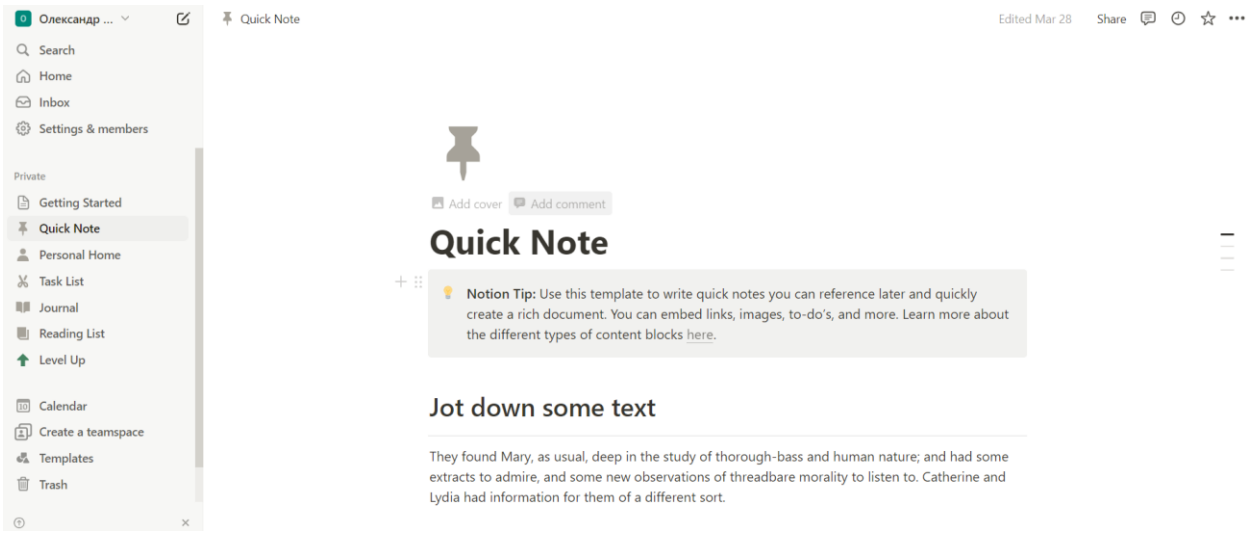

Рис 1.4 Загальний вигляд та можливості Notion [15]

Notion є потужним інструментом для організації та управління навчальним процесом. Хоча він не є спеціалізованою системою управління навчанням, його багатофункціональність та можливості кастомізації роблять його привабливим для викладачів і студентів.

Проєкти в Notion можна прив’язати до календаря, що підтверджує його зручність та багатофункціональність. Але оскільки додаток не був створений спеціально для організації навчального процесу, то більшість функцій необхідних для навчання потрібно створювати самому за допомогою більш абстрактних інструментів додатку, або ж купувати шаблони, з якими в свою чергу потрібно навчитися працювати. Тож основною проблемою додатку є низькорівневість, для того щоб справді зручно огранізувати навчальний процес в додатку потрібно спочатку витратити значну кількість зусиль і ресурсів для навчання викладачів. Тож як висновок користувачам доводиться доторкатися до тих рівнів програми, про які вони точно не хотіли б знати, навіть для таких простих дій, як додавання завдання, та планування занять.

1.5.5 Огляд сайту розкладу КПІ

Якщо користувачам потрібна простота, та постійний актуальний доступ до інформації про навчальний графік, то найкращим прикладом можуть бути локальні

системи розкладу, наприклад, сайт розкладу Київського політехнічного інституту ім. Ігоря Сікорського.

Сайт розкладу КПІ ім. Ігоря Сікорського є важливим інструментом для студентів та викладачів, що дозволяє ефективно організовувати навчальний процес. Він надає доступ до актуальної інформації про розклад занять. Він є простим та інтуїтивно зрозумілим у навчанні.

Розглянемо основні функції розкладу КПІ ім. Ігоря Сікорського [16]:

- Доступ до розкладу - сайт розкладу дозволяє студентам та викладачам переглядати розклад занять у зручному форматі. Розклад доступний для всіх факультетів, кафедр та груп університету.
- Зручний інтерфейс - інтерфейс сайту інтуїтивно зрозумілий та зручний у користуванні. Користувачі можуть легко знайти необхідну інформацію за допомогою декількох кліків.
- Оновлення розкладу - розклад занять регулярно оновлюється для відображення актуальної інформації. Це дозволяє користувачам завжди бути в курсі останніх змін та уникати плутанини.

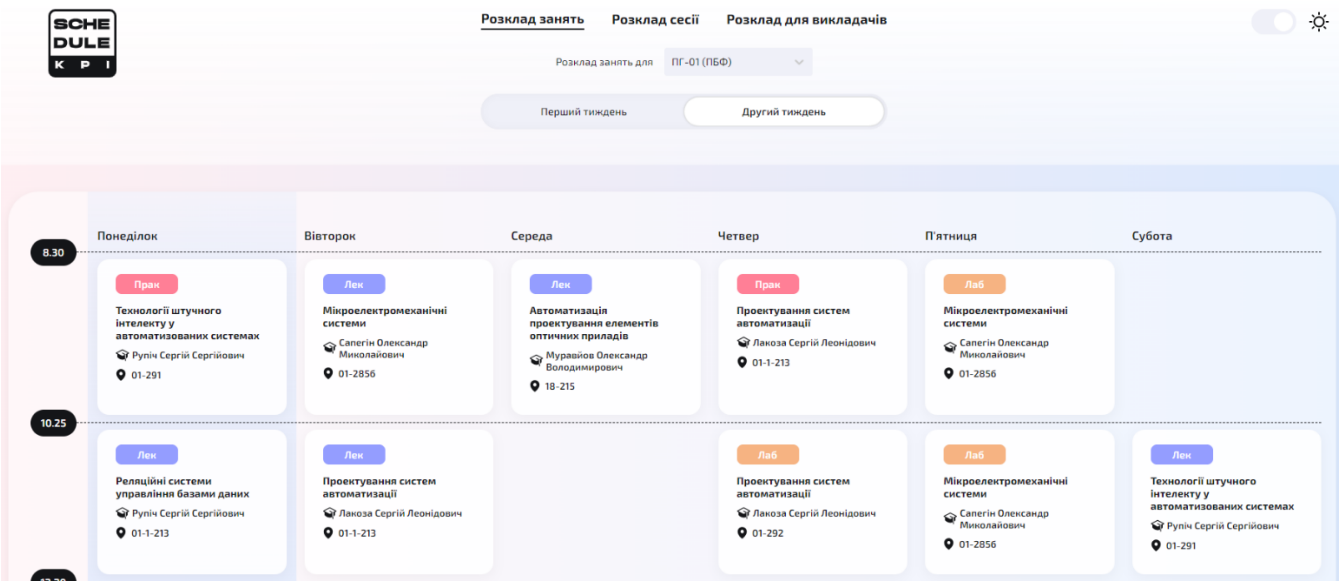

Рис. 1.5 Загальний вигляд сайту розкладу КПІ ім. Ігоря Сікорського

Сайт розкладу КПІ є одним із ключових ресурсів для організації навчального процесу, забезпечуючи зручний доступ до актуальної інформації та сприяючи ефективному плануванню часу студентів і викладачів. На відміну від вище досліджених платформ, цей сайт має набагато простіший але водночас корисний функціонал. Користувач в будь який момент може отримати актуальну інформацію про розклад занять. Але й тут є певні проблеми, через мінімалізований функціонал, викладачі не мають можливості редагувати інформацію розкладу згідно своїх потреб. Якщо викладач планує створити додаткову конференцію для студентів, то він буде змушений ставити будильник на відповідну дату та час щоб не забути. Також із розкладу викладач не має доступу до списку групи, що також не є зручним.

1.6 Загальний аналіз обраних додатків

Проаналізувавши технології організації освітніх процесів та сучасні платформи, які розроблені для супроводу освітнього процесу, було досліджено, що всі додатки є локально спеціалізованими та покривають якусь конкретну частину відповідальності щодо організації навчання. Одні додатки використовують для створення, призначення, та оцінювання завдань, але їх основною проблемою є відсутність системи розкладу і переліку всіх завдань, що змушує користувачів по черзі моніторити кожну дисципліну окремо. Інші групи додатків мають завищену абстрактність, що призводить до потреби додаткового навчання та негативно впливають на відсоток автоматизації процесу, оскільки окрім призначення завдань, викладачі мають пройти додаткове навчання та спочатку створити шаблон для створення цих завдань. Найпростіші з них надають максимум користі за мінімальну кількість зусиль прикладених із сторони користувача, але мають обмежений функціонал.

Проаналізувавши всі деталі досліду стає зрозуміло, що в даній ситуації на ринку просто відсутній мінімалістичний додаток із простим інтерфейсом, який покриє базові потреби викладача. Особливою проблемою стає завдання додати заняття із інтервалом повторення більше одного тижня.

Тому було ухвалено рішення розробити такий додаток, який буде мати мінімалістичний, інтуїтивно зрозумілий інтерфейс, можливість перегляду та редагування розкладу зі сторони викладача. Можливість додавання, редагування та видалення навчальних груп із переліком всіх студентів. Можливість призначення розкладу із необмеженим значенням інтервалу повторення заняття, яке можна повторити як кожен тиждень, так і раз на місяць чи раз на рік, що буде корисно для призначення заліків, та модульних контрольних робіт та автоматизує процес їх призначення.

1.7 Формування та формалізація вимог до автоматизованої системи

Системні вимоги є детальним описом функціональності та характеристик, яких очікують від програмного забезпечення або системи. Цей опис визначає, що саме повинна робити система і які її технічні параметри. Важливо розуміти, що системні вимоги не визначають, як саме повинна бути реалізована система, а лише описують, що вона має виконувати. Ці вимоги зазвичай є основою для укладання контракту на розробку програмного продукту, забезпечуючи повну специфікацію системи та її функціональності [17].

Вимоги до системи можуть бути розділені на дві основні категорії:

- Функціональні вимоги - це вимоги, що визначають, які конкретні функції та операції повинна виконувати система. Вони описують можливості системи, її взаємодію з користувачем та інші системи, а також обробку даних. Наприклад, функціональні вимоги можуть включати можливість реєстрації користувачів, вхід у систему, додавання товарів у кошик покупок тощо [18].
- Нефункціональні вимоги - це вимоги, що визначають якість та характеристики системи, які не стосуються конкретних функцій. Вони описують вимоги до продуктивності, надійності, безпеки, масштабованості та інших аспектів системи. Наприклад, нефункціональні вимоги можуть включати швидкодію системи, рівень безпеки даних, сумісність із різними платформами тощо [19].

Перед початком роботи над проєктом всі вимоги детально формалізуються, це дозволяє розробникам та замовникам чітко зрозуміти, яким має бути результат роботи та якими критеріями він буде оцінюватися.

1.7.1 Функціональні вимоги

Згідно з проведеними опитуваннями викладачів, було відокремлено певні функціональні вимоги до автоматизованої системи.

1.7.1.1 Функціональні вимоги до системи автентифікації та доступу до бази даних

Перший функціонал який планується реалызувати в додатку, це система реєстрації та автентифікації користувачів. Функціональні вимоги цієї системи:

- Система реєстрації та автентифікації:
    - Функціональність реєстрації нових користувачів передбачає створення облікового запису в системі, включаючи введення особистих даних користувача та вибір унікального логіну та пароля.
    - Механізм автентифікації гарантує, що лише користувачі з правильними обліковими даними мають доступ до функціоналу системи. Після введення логіну та пароля система перевіряє їх на відповідність тим, які зберігаються в базі даних.
- Доступ до бази даних в реальному часі
    - Функціональність доступу до бази даних забезпечує миттєвий доступ до інформації для користувачів. Це означає, що будь-які зміни, зроблені в базі даних, будуть відображені без затримок у веб-інтерфейсі додатку.

1.7.1.2 Функціональні вимоги до створення навчальних груп

- Управління навчальними групами:
    - Функціонал додавання нових груп. Надає користувачам можливість створювати нові навчальні групи.
    - Функціонал редагування навчальних груп. Після створення групи користувач може виконати редагування її параметрів, таких як назва та список студентів.
    - Функціонал видалення навчальних груп дозволяє користувачам прибрати непотрібні або застарілі групи зі списку доступних.

- Управління студентами в навчальних групах:
    - Користувачі можуть додавати нових студентів до навчальних груп. Можна вказати ім’я, прізвище та вік студента після чого додати його до групи.
    - Функціонал видалення студентів з навчальних груп дозволяє користувачам видаляти студентів, які більше не повинні навчатися у даній групі.

1.7.1.3 Функціональні вимоги до створення навчальних дисциплін

Додаток має передбачати можливість додавання нових навчальних дисциплін, які потім будуть прив’язуватись до розкладу. Функціональні вимоги до створення навчальних дисциплін:

- Система додавання нових навчальних дисциплін:
    - Додаток повинен надавати користувачам можливість додавання нових навчальних дисциплін до системи. Дисципліни мають мати два параметри назву та вартість навчання.
    - Користувачі повинні мати можливість видаляти навчальні дисципліни, які більше не актуальні або помилково додані.

1.7.1.4 Функціональні вимоги до календаря

Календар є важливою частиною додатку, яка дозволяє користувачам зручно планувати свої заняття та інші події. Він повинен забезпечувати простий і зрозумілий інтерфейс, і відповідати таким функціональним вимогам:

- Доступ до календаря:
    - Відкриття календаря - користувачі повинні мати можливість відкривати календар з головного меню або панелі навігації додатку. Календар має бути доступний з будь-якої частини додатку.
    - Відображення поточної дати - при відкритті календаря, за замовчуванням повинна відображатися поточна дата, місяць і рік. Це дозволяє користувачам одразу бачити актуальний день та планувати події на найближчий час.
- Перегляд і навігація:
    - Навігація по календарю - користувачі повинні мати можливість пролистувати дні, місяці та роки вперед і назад. Це дозволяє переглядати розклад як на минулі дати, так і на майбутні. Переміщення між датами повинно бути інтуїтивним і зручним, з мінімальною кількістю кліків або свайпів.
    - Вибір конкретного дня - користувачі повинні мати можливість обрати конкретний день у календарі для перегляду подій або додавання нових. Обраний день повинен бути виділений візуально, щоб користувач міг легко зрозуміти, яку дату він вибрав.

1.7.1.5 Функціональні вимоги до призначення занять

Головним завданням календаря є можливість прив’язки занять до конкретних дат. Робота із заняттями в календарі має відповідати таким функціональним вимогам:

- Форма додавання уроку - після вибору конкретного дня, користувачі повинні мати можливість додати урок за допомогою спеціальної форми. У формі повинні бути доступні наступні поля для введення та налаштування параметрів уроку:
    - Вибір часу уроку - для вибору часу початку уроку користувачі повинні мати доступ до зручних слайдерів, які дозволяють швидко та точно встановити потрібний час. Це забезпечує зручність у налаштуванні тривалості уроку.
    - Вибір дисципліни - користувачі повинні мати можливість обрати дисципліну з існуючого списку, що зберігається в базі даних. Список повинен відображати доступні дисципліни, які можна легко вибрати за допомогою випадаючого меню або пошуку.
    - Вибір навчальної групи - користувачі повинні мати можливість обрати навчальну групу з існуючого списку, що зберігається в базі даних. Список груп повинен бути організований таким чином, щоб користувач міг легко знайти та вибрати потрібну групу.
    - Введення повторення заняття - користувачі повинні мати можливість встановити частоту повторення заняття, вказуючи інтервал у тижнях. Це дозволяє автоматично додавати заняття у календар на регулярній основі (наприклад, кожен тиждень, кожні два тижні тощо).
- Відображення і редагування занять:
    - Відображення уроків у календарі Усі додані заняття повинні відображатися у календарі на відповідні дати. заняття повинні бути виділені таким чином, щоб користувачі могли легко їх побачити та ідентифікувати.
    - Видалення уроків - користувачі повинні мати можливість видаляти заняття, які більше не актуальні або були додані помилково. Видалення уроків повинно супроводжуватися підтвердженням, щоб уникнути випадкового видалення важливої інформації.

1.7.2 Нефункціональні вимоги

Нефункціональні вимоги є важливою складовою розробки програмного забезпечення, оскільки вони визначають якість системи та її взаємодію з користувачами і зовнішніми системами. Вони не стосуються конкретної функціональності, а зосереджені на загальних характеристиках додатку, таких як продуктивність, надійність, безпека та зручність використання.

Згідно з проведеними опитуваннями викладачів, було відокремлено певні нефункціональні вимоги до автоматизованої системи.

1.7.2.1 Вимоги до продуктивності

Під час опитування було відокремлено такі вимоги до продуктивності:

- Швидкість завантаження - календар та інші розділи додатку повинні завантажуватися протягом не більше 2 секунд на середньостатистичному інтернет-з'єднанні.
- Швидкість відгуку - додаток повинен відгукуватися на дії користувача (відкриття календаря, додавання подій тощо) протягом 2 секунд.

1.7.2.2 Вимоги до надійності

Під час опитування було відокремлено такі вимоги до надійності:

- Доступність системи - додаток повинен бути доступним для користувачів не менше ніж 99,9% часу протягом місяця.

1.7.2.3 Вимоги до безпеки

Під час опитування було відокремлено такі вимоги до безпеки:

- Захист даних - усі дані, передані між клієнтом і сервером, повинні бути якісно зашифровані.

- Захист доступу до даних – доступ до перегляду чи читання даних повинен бути наданий тільки зареєстрованим користувачам згідно з їх унікальними ID.

1.7.2.4 Вимоги до зручності використання

Під час опитування було відокремлено такі вимоги до зручності використання:

- Інтуїтивний інтерфейс - інтерфейс додатку повинен бути простим і зрозумілим, дозволяючи користувачам виконувати основні дії з мінімальної кількістю кліків.
- Нейтральні кольори - інтерфейс додатку повинен бути виконананий в нейтральних або пастельних тонах, задля запобігання псування зору користувачів.

1.8 Результати аналізу пердметної області

В першому розділі було проведено аналіз організації навчальної системи. Досліджено традиційні та сучасні методи організації навчальної системи. Проведено огляд існуючих LMS систем, виявлено їх сильні та слабкі сторони. При аналізі систем було виявлено ряд недоліків, такі як відсутність доступу до списку групи всередині системи, відсутність системи розкладу з прив’язкою до календаря та можливістю динамічно виставляти інтервал повторень заняття, ускладнений функціонал, який потребує додатково навчання для використання системи, відсутність можливості для викладача призначати додаткові заняття самостійно. Деякі із розглянутих систем частково перекривають деякі із перерахованих недоліків, та немає жодної LMS, яка б могла вирішити всі виявлені проблеми.

Для вирішення проблеми було ухвалено рішення розробити додаток для автоматизації ряду рутинних процесів з якими зіштовхуються викладачі. Для створюваної системи було сформовано ряд функціональних та нефункціональних вимог, які допоможуть зробити процес створення додатку більш конкретним.

Серед функціональних вимог було сформовано чотири основні частини до функціоналу створюваного додатку такі як можливість реєстрації та автентифікації, можливість оперування навчальними групами та студентами групи, можливість оперування навчальними дисциплінами та можливість використання календаря для планування занять викладача із можливістю налаштування динамічного повторення занять через вказаний період часу.

Серед нефункціональних вимог було також було сформовано чотири основні частини вимог таких як вимоги до продуктивності, дотримання яких, надасть потрібний рівень швидкодії додатку, вимоги до надійності та безпеки, дотримання яких, зробить використання додатку більш безпечним, а також надасть доступ до використання додатку протягом 99.9% часу та вимоги до зручності використання, дотримання яких, надасть користувачам зручний інтерфейс, та функціонал для використання якого не потрібно використовувати додатковий час на навчання.

Дотримання сформованих вимог забезпечить якість, швидкодію та зручність використання додатку, мінімалізму ризики втрати даних, та допоможе додатку перекрити всі досліджені недоліки вже існуючих LMS.

РОЗДІЛ 2

СТВОРЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ

2.1 Вибір та огляд технологій для створюваного додатку

Важливою частиною розробки додатку є вибір технологій, які будуть використовуватись у процесі створення та підтримки системи. Від правильного вибору технологій залежить продуктивність, масштабованість та зручність використання додатку як для викладачів, так і для студентів. Технології, які обираються на початкових етапах розробки, впливають на всі аспекти роботи додатку, починаючи від інтерфейсу користувача і закінчуючи обробкою даних на сервері.

Вибір технологій охоплює кілька ключових напрямків, кожен з яких вимагає ретельного аналізу та оцінки. Основні напрями включають вибір технологій для фронтенду, бекенду, бази даних, інтеграційних сервісів, інструментів для тестування та управління проектом.

2.1.1 Вибір технологій для бекенду

Бекенд, або серверна частина додатку, є фундаментальною складовою будь-якого додатку. Ця частина системи обробляє логіку додатку, управляє базами даних, взаємодіє з іншими сервісами та забезпечує безпеку додатку. Основна роль бекенду полягає у виконанні всіх операцій, які не видимі користувачу, але є критичними для функціонування додатку. Функції бекенду включають обробку запитів від клієнтів, управління базами даних, аутентифікацію та авторизацію користувачів, виконання бізнес-логіки додатку та інтеграцію з іншими сервісами.

Однією з найважливіших функцій бекенду є обробка запитів від клієнтської частини (фронтенду) додатку. Бекенд отримує запити, обробляє їх та повертає відповіді, виконуючи маршрутизацію запитів до відповідних контролерів та

генерацію відповідей у вигляді даних або HTML-сторінок. Це забезпечує злагоджену роботу додатку та швидкий відгук на дії користувачів. Управління базами даних є ще однією ключовою функцією бекенду.

Для успішної реалізації бекенду критично необхідно вибрати потужну мову програмування. Мова має підтримувати об'єктно-орієнтоване програмування (ООП), яке забезпечує організованість та гнучкість коду. ООП дозволяє структурувати код у вигляді окремих частин (класів та об'єктів), що робить його зрозумілішим та легшим для підтримки, сприяє повторному використанню коду, полегшує тестування та забезпечує можливість майбутнього масштабування додатку [20].

Основні принципи ООП включають[20]:

- Інкапсуляція - це концепція програмування, яка дозволяє обмежити доступ до деяких компонентів об'єкта, та розділяє код на логічні модулі, що полегшує розробку, тестування та підтримку додатку.
- Наслідування це технологія, яка новим компонентам успадкувати частину коду від вже існуючих, що дозволяє використовувати та розширювати функціональність цього класу без необхідності повторювати код.
- Поліморфізм - це можливість об'єктів робити різні речі, в залежності від того, який саме об'єкт використовується. Що дозволяє значно скоротити код та спростити його структуру.

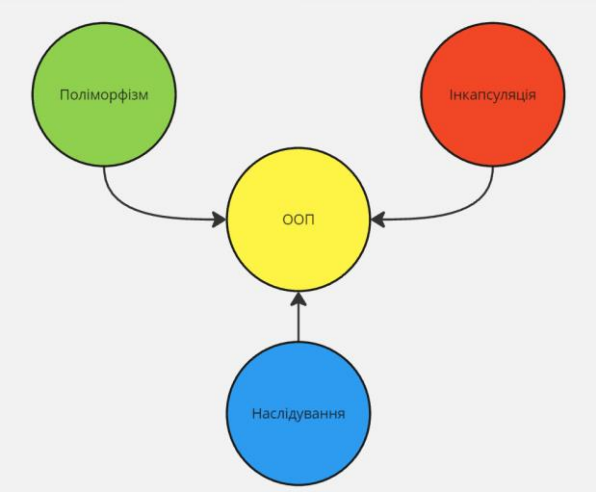

Рис 2.1 Основні принципи ООП

Отож для розробки додатку, потрібна сучасна мова програмування із широким функціоналом, підтримкою ООП, великою кількістю фреймворків та можливістю керування базами даних. Проаналізувавши всі доступні варіанти фаворитом стала мова програмування C#.

C# (сі шарп) — це універсальна високорівнева об'єктно-орієнтована мова програмування, яка з моменту свого появи у 2001 році стала ключовою мовою для розробки програмного забезпечення під операційні системи Windows. Синтаксис C# є відносно простим і інтуїтивно зрозумілим, що робить її ідеальним вибором для новачків. Відповідно до німецької компанії Statista, майже 74% всіх користувачів використовують ОС Windows, що підкреслює важливість C# у світі програмування [21].

Проте використання C# не обмежується тільки Windows. З виходом платформи .NET Core, екосистема C#/.NET отримала зручний інструментарій для написання кросплатформенних додатків під всі популярні операційні системи, включаючи macOS, Linux.NET, Android та iOS. Це робить C# відмінним вибором для створення мультиплатформенного програмного забезпечення для різних сфер життя, включаючи розважальний контент, відеоігри, мобільні додатки, хмарні та корпоративні рішення.

Кросплатформенність та універсальність C#, а також постійний розвиток платформи .NET, дозволяють створювати практично будь-яке програмне забезпечення для будь-якої операційної системи. C# успішно застосовується у створенні серверної сторони веб-додатків, настільних та мобільних ігор, корпоративних та настільних додатків, мобільного софту, хмарних сервісів, Big Data рішень та у машинному навчанні, тестуванні та багатьох інших галузях. Завдяки цьому, мова програмування C# стала невід'ємною частиною сучасного світу технологій, дозволяючи створювати ефективні, надійні та масштабовані рішення для різноманітних задач.

C# є універсальною у використанні. Вона дуже зручна для розробки як 2D, так і 3D ігор, а також особливо добре зарекомендувала себе у розробці корпоративних

додатків та серверних веб-додатків. А отже ця мова програмування є однією із найкращих для вирішення отриманих проблем.

База даних - це організована колекція даних, яка зберігається та управляється таким чином, щоб її було легко доступно, оновлювати та використовувати. Вона може включати в себе інформацію про користувачів, продукти, транзакції, історію, текстові дані, графічні зображення та багато іншого [22].

Бази даних дозволяють зберігати великі обсяги структурованих даних у відсортованому та організованому вигляді. Це дозволяє додаткам ефективно зберігати та управляти інформацією, необхідною для їх роботи.

Також бази даних забезпечують швидкий доступ до великої кількості даних. Це дозволяє додаткам ефективно виконувати операції з даними, такі як пошук, оновлення, видалення та вставка.

В даному випадку є потреба в мінімалістичній, та максимально швидкій базі даних, яка може легко приєднатися до Unity, надати можливість для збереження та зчитування даних в реальному часі в тому числі даних необхідних для процесу автентифікації користувачів. Після аналізу вищеописаних вимог, найкращим варіантом в даному випадку є платформа Firebase та її інструменти такі як Firebase Autentification для системи автентифікації користувачів, а також Firebase Realtime Database для доступу до бази даних в реальному часі.

Firebase - це високорівнева платформа, що надає розробникам широкий спектр інструментів для створення мобільних і десктопних додатків та веб-сайтів з величезним функціоналом. Ця платформа, починаючи як стартап, з часом стала однією з найбільш популярних серед розробників, завдяки своїй ефективності та надійності [23].

Одна з головних переваг Firebase полягає в тому, що вона дозволяє розробникам уникнути витрат на створення власного backend'у, або серверної частини проекту. Замість цього, Firebase пропонує готові рішення для збереження даних, автентифікації користувачів, хостингу веб-сторінок та багато іншого.

Крім того, Firebase надає розробникам різноманітні інструменти для аналізу та моніторингу продуктивності додатків, що дозволяє зрозуміти, як користувачі взаємодіють з програмою та як можна покращити її функціонал.

Для системи аутентифікації та збереження даних користувачів було прийнято рішення використати Firebase Authentication.

Firebase Authentication - це служба автентифікації, яка надається Firebase, котра дозволяє розробникам додавати зручні та безпечні механізми аутентифікації користувачів у свої додатки та сайти. Ця служба дозволяє використовувати різні методи аутентифікації, такі як електронна пошта та пароль, номер телефону, облікові записи соціальних мереж (такі як Google, Facebook, Twitter) та інші [24].

Firebase Authentication надає розробникам можливість зосередитися на функціональності їх додатків, маючи при цьому високий рівень безпеки і зручності для користувачів. Вона також інтегрується з іншими сервісами Firebase, такими як Firebase Realtime Database та Firebase Cloud Functions, щоб надати повний стек засобів для розробки сучасних додатків.

Важливою перевагою є Безпека Firebase Authentication. Система забезпечує високий рівень безпеки захищеним з'єднанням та шифруванням даних, а також застосовує передові методи аутентифікації, щоб гарантувати, що доступ до додатків здійснюється тільки автентифікованими користувачами.

Загалом, Firebase Authentication - це потужний інструмент, який дозволяє розробникам легко додавати та управляти аутентифікацією користувачів у своїх додатках і веб-сайтах, забезпечуючи високий рівень безпеки та зручності для користувачів.

Як вже було вище описано, для збереження даних в додатку, буде використано Firebase Realtime Database, яка є не реляційною або NoSQL базою даних типу “Ключ-значення”. NoSQL бази даних відрізняються від традиційних SQL баз даних своїм підходом до організації, зберігання та доступу до даних. Основні характеристики NoSQL включають горизонтальне масштабування для легкого збільшення обсягів даних, високу продуктивність та підтримку різних типів даних, включаючи структуровані, напівструктуровані та неструктуровані дані [25].

Firebase Realtime Database - це сервіс, розроблений Google, який надає можливість реального часу зберігати та синхронізувати дані між різними клієнтами додатків. Основним принципом роботи бази даних є робота з даними за принципом “ключ - значення” у форматі JSON, що дозволяє надавати миттєві оновлення для всіх підключених клієнтів. Це означає, що будь-які зміни, зроблені одним користувачем, автоматично відображаються на всіх інших підключених пристроях без необхідності оновлення. Крім того, база даних забезпечує синхронізацію між різними платформами, такими як Android, iOS та веб, що дозволяє користувачам отримувати однаковий досвід користувача незалежно від платформи [26].

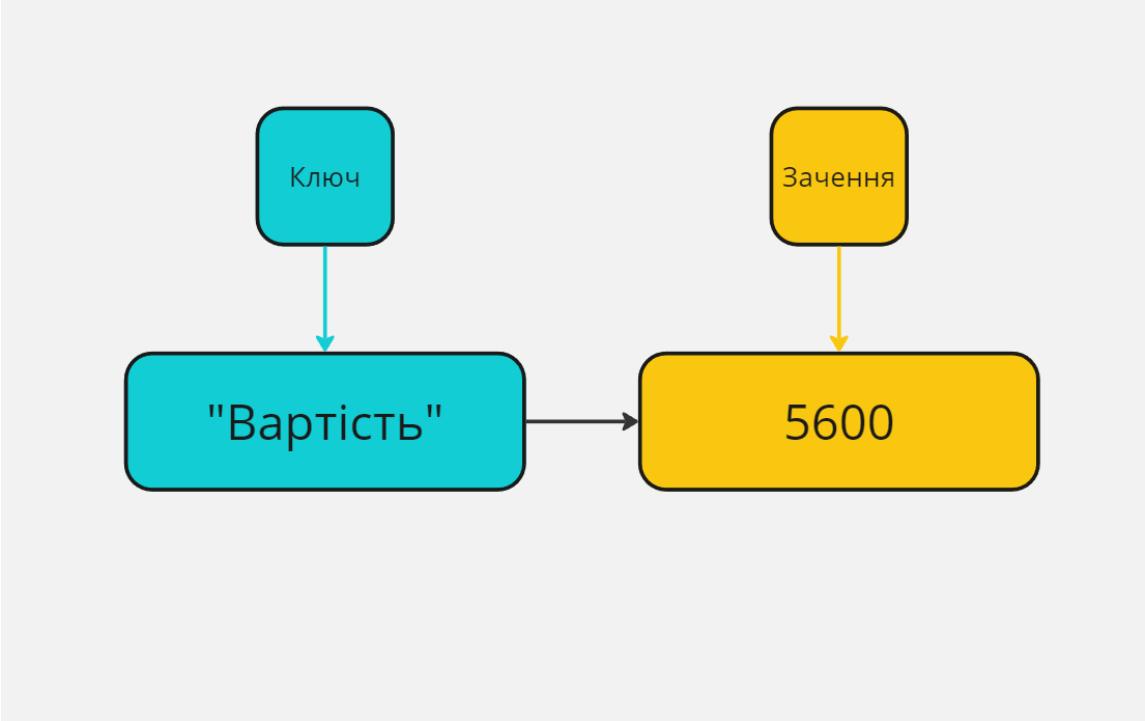

Рис. 2.2 Принцип збереження даних у форматі JSON

Цікавою особливістю формату JSON є те, що буль-який ключ має можливість мати необмежену кількість дочірніх ключів. Це відкриває широкі можливості для проектування архітектури бази даних, та забезпечує велику швидкість відклику, оскільки для отримання певної групи інформації потрібно лише надати батьківський ключ, а всю іншу інформацію можна з легкістю отримати звертаючись до його дочірніх елементів.

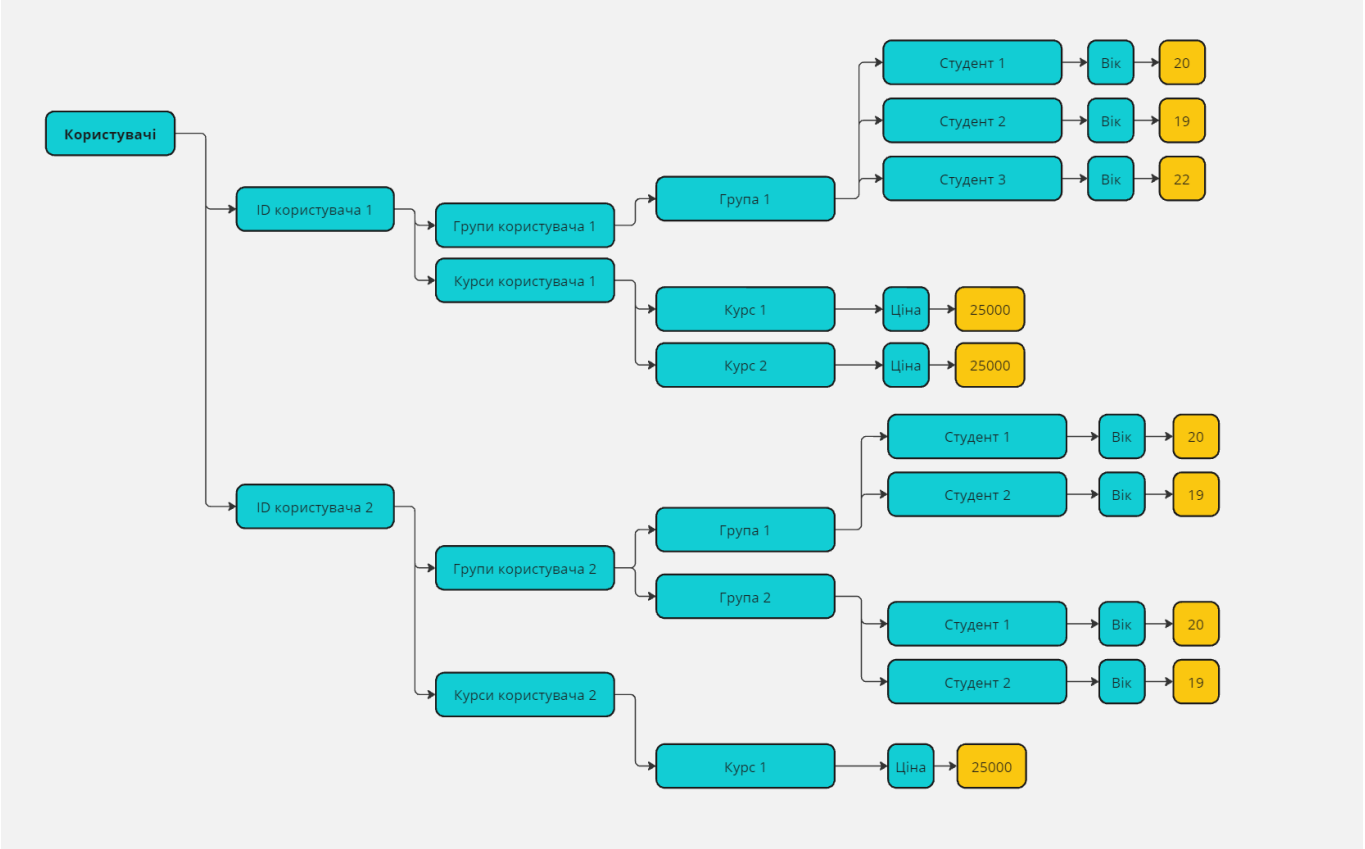

Рис 2.3 Приклад використання можливостей архітектури бази даних на основі JSON

Також Firebase Realtime Database підтримує роботу в офлайн-режимі, що дозволяє користувачам працювати з даними навіть у відсутність Інтернет-з'єднання, а пізніше автоматично синхронізувати їх при відновленні зв'язку. Також можна налаштовувати права доступу до даних на рівні користувачів, що забезпечує конфіденційність та безпеку інформації.

2.1.2 Вибір IDE

IDE (Integrated Development Environment) - це комплексний інструмент, який надає розробникам усі необхідні засоби для написання, тестування, налагодження та керування програмним кодом в одному інтерфейсі. Вона всі інструменти, які полегшують процес розробки програмного забезпечення. Завдяки IDE розробники можуть ефективно працювати над своїми проектами, отримуючи доступ до усіх необхідних інструментів в одному зручному середовищі.

Вибір Integrated Development Environment (IDE) є важливим кроком для ефективного програмування. Для обраних завдань та для програмування на C#, в цілому фаворитом на сьогодні є Visual Studio 2022 [27]. Visual Studio 2022 - це один з найпопулярніших та потужніших IDE, який ідеально підходить для роботи з мовою програмування C#.

Visual Studio 2022 має багато переваг, які роблять його привабливим для C# програмістів. Також Visual Studio 2022 має широкий набір додаткових функцій і плагінів, які полегшують розробку, такі як IntelliSense - інтелектуальне доповнення коду, вбудована допомога та документація, та багато інших. Весь цей функціонал, допомагає швидко, та максимально безпомилково писати велику кількість коду, що робить його явним фаворитом для програмування на C# [28].

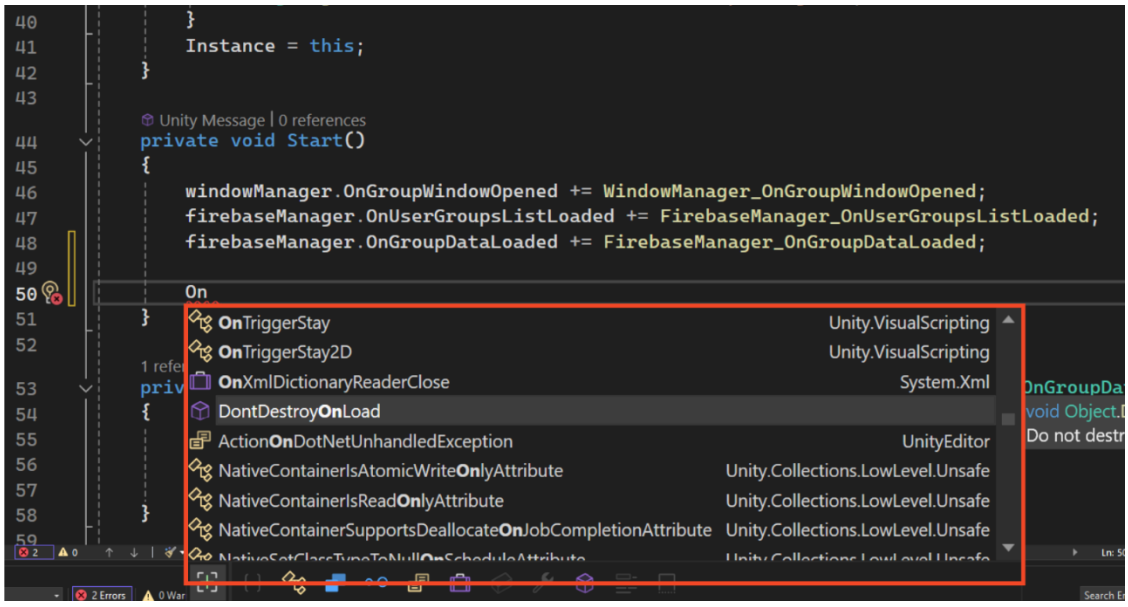

Рис 2.4 Результат роботи інтелектуального доповнення коду в Visual Studio 2022

2.1.3 Вибір інструментів для управління проектом

Ефективне управління проектом потребує використання систем контролю версій, інструментів для планування та відстеження завдань, а також інструментів для автоматизації збірки та розгортання додатку. Вибір цих інструментів допомагає

організувати роботу команди, забезпечити прозорість процесів та швидке впровадження змін.

Для кожного із вище розглянутих завдань зазвичай використовують різні програми та фреймворки для вирішення конкретної задачі, але є і варіанти де весь цей функціонал зібраний в одному інтерфейсі. Для цих функцій ідеально підходить Unity [29].

Unity – це потужний і надзвичайно популярний багатоплатформений рушій для створення комп'ютерних ігор, а також інтерактивних 2D і 3D-додатків. Він надає розробникам всі необхідні інструменти для створення ігор та додатків [30].

Однією з ключових можливостей Unity є його багатоплатформність. Цей рушій дозволяє розробляти проєкти для багатьох платформ, таких як ПК, мобільні пристрої на базі iOS та Android, ігрові консолі (PlayStation, Xbox), а також для пристроїв віртуальної реальності. Це лише частина можливих варіантів розробки.

Unity – це не тільки потужний інструмент для створення ігор, але й універсальний рушій, який широко використовується для розробки різноманітних додатків. Unity підтримує розробку як 2D, так і 3D-додатків, що дозволяє створювати інтерактивні візуалізації, моделювання та анімації, які можуть використовуватися у таких сферах як освіта, архітектура, медицина та індустрія розваг.

Unity широко використовується для створення навчальних додатків та симуляторів. Інтерактивні елементи, анімації та можливість створення інтерактивних сценаріїв дозволяють розробникам створювати ефективні навчальні інструменти для різних аудиторій, що може бути корисним у школах, університетах та корпоративному навчанні. Також Unity можна використовувати для створення додатків, які візуалізують дані в реальному часі. Це може бути корисним для аналітики, моніторингу процесів та презентацій, оскільки візуалізація складних наборів даних у 3D значно полегшує їх аналіз та інтерпретацію.

У Unity, скрипти є основним засобом програмування та керування поведінкою об'єктів у віртуальному середовищі. Їхнє використання дозволяє розробникам створювати різноманітні та складні інтерактивні взаємодії, які формують геймплей та відчуття віртуального світу для гравців.

Скрипти можуть бути додані до будь-якого об'єкта у сцені Unity. Вони використовуються для визначення реакцій об'єктів на різні події, такі як натискання клавіш чи взаємодія з інтерфейсом. Наприклад, скрипт може керувати обробляти взаємодію користувача з елементами UI, або відтворювати анімації чи візуальні ефекти.

Однією з ключових переваг використання скриптів у Unity є можливість програмувати у C#, що є ключовим аспектом, який повпливав на вибір саме цього інструменту.

У великих проєктах, де потрібно керувати багатьма об'єктами та складними системами, використання скриптів стає незамінним. Вони дозволяють структурувати код, розділяючи логіку та функціонал на окремі модулі, що полегшує розуміння та підтримку проєкту в майбутньому.

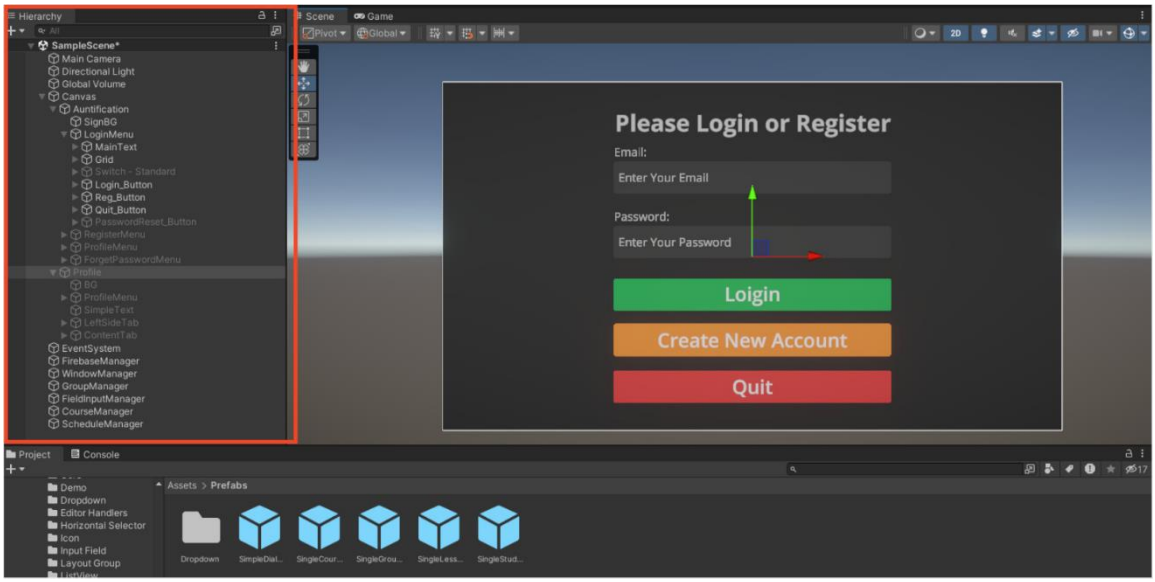

Рис 2.5 Приклад реалізації структури проєкту за домогою функціоналу Unity

Таким чином, Unity є не тільки потужним інструментом для розробки ігор, але і для створення інших інтерактивних додатків, що робить його універсальним вибором для розробників по всьому світу.

2.1.4 Вибір технологій для фронтенду

Фронтенд є обличчям додатку, саме з ним взаємодіють користувачі. Вибір фреймворків та бібліотек для фронтенду впливає на швидкість розробки, зручність інтерфейсу та його продуктивність. Важливо обрати такі технології, які забезпечать швидке завантаження сторінок, легку адаптацію до різних пристроїв та інтуїтивно зрозумілий інтерфейс.

Фреймворки дозволяють зменшити кількість повторюваного коду та значно скорочують час, необхідний для розробки. Це зменшує обсяг ручної роботи, необхідної для вирішення стандартних задач, і дозволяє розробникам зосередитися на специфічних аспектах проєкту, підвищуючи продуктивність і ефективність роботи. Фреймворки також допомагають організувати робочий процес, забезпечуючи структуру та правила, які полегшують роботу в команді та роблять код більш підтримуваним.Фреймворки для фронтенд-розробки значно спрощують створення веб-продуктів, надаючи розробникам готові рішення для типових задач і забезпечуючи високу продуктивність роботи [31].

Оскільки основним інструментом для управління проєктом було обрано двигун Unity, який також має широкий спектр інструментів, та бібліотек для створення візуальної частини додатку – є сенс використати двигун і для фронтенду [32].

2.1.5 Вибір графічних бібліотек для створення графічного інтерфейсу на Unity

У середовищі Unity, бібліотеки для візуального інтерфейсу - це набір інструментів та компонентів, які допомагають розробникам створювати користувацький інтерфейс для своїх ігор чи додатків. Ці бібліотеки мають різні функції, такі як створення кнопок, вікон, текстових полів, меню, переходів між сценами тощо, що дозволяє створювати зручний та привабливий інтерфейс для користувачів [33].

Під час розробки додатку було використано дві основних бібліотеки для графічного інтерфейсу:

- UnityEngine.UI - це набір компонентів, доступних у Unity, які дозволяють створювати та керувати інтерфейсом користувача (UI) в іграх та додатках. За допомогою цих компонентів розробники можуть швидко створювати різноманітні елементи, такі як кнопки, тексти, поля введення, вікна, списки та скроллбари. Використання цього фреймворку дозволяє розробникам легко і швидко створювати якісний та привабливий інтерфейс для своїх проектів [32].
- TextMesh Pro (TMPro) - це потужне інтегроване розширення для Unity, яке створене з метою поліпшення відображення тексту в ігрових проектах. За допомогою функціонала розробники можуть легко контролювати шрифти, розміри, колір, стилізацію та позицію тексту. Крім того, TMPro підтримує використання спеціальних ефектів, таких як тіні, вигини, обводки, розмиття та інші, що дозволяє створювати більш естетичний та привабливий вигляд текстових об'єктів [34].

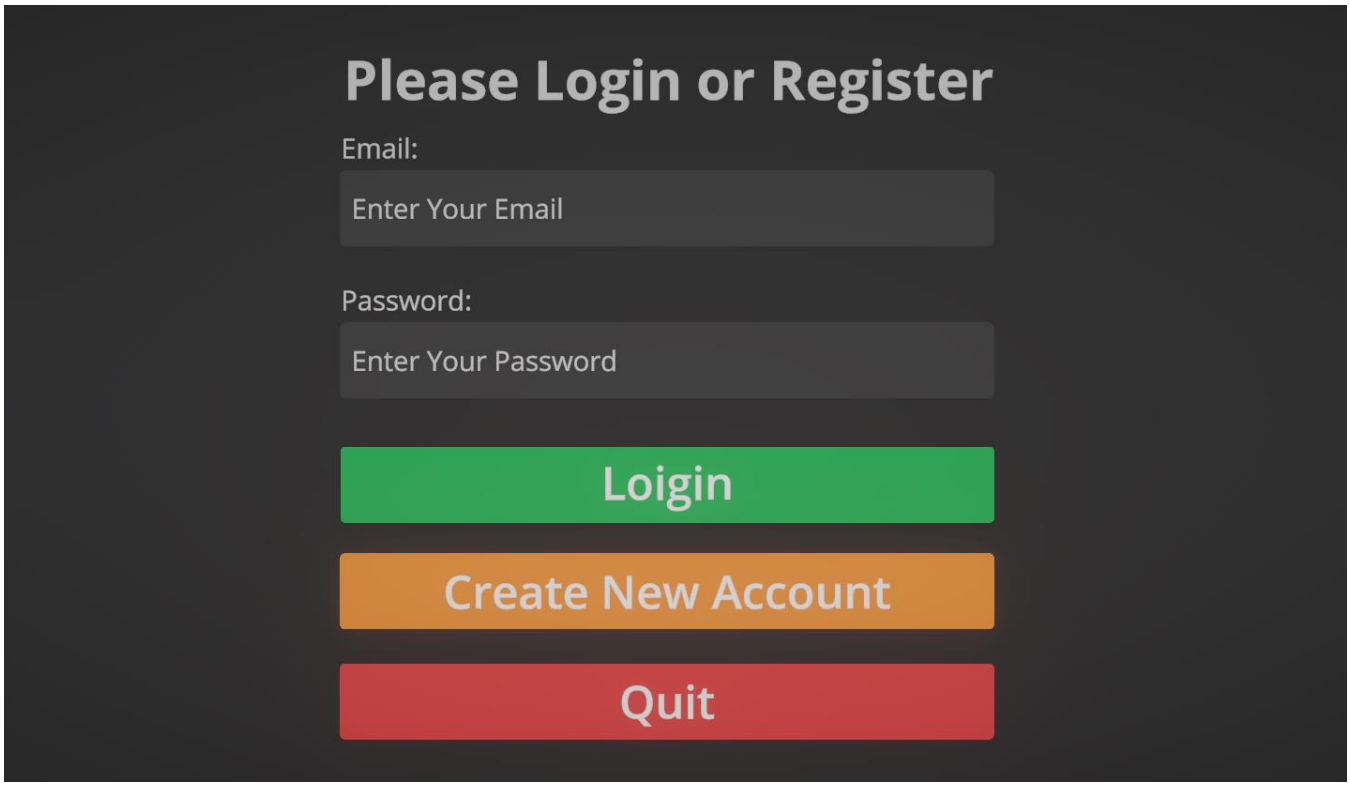

Рис 2.6 Приклад інтерфейсу реалізованого за допомогою обраних бібліотек

Ці бібліотеки допомагають розробникам Unity створювати привабливий та функціональний візуальний інтерфейс для своїх ігор та додатків, що покращує користувацький досвід і забезпечує зручність взаємодії з продуктом.

2.1.6 Вибір асетів із набором необхідної графіки та функціоналом

Асети в Unity - це всі елементи, що використовуються в проекті, такі як моделі 3D, текстури, аудіофайли, анімації, шрифти, скрипти, і багато іншого. Вони представляють собою різноманітні ресурси, які необхідні для створення і відтворення візуального та звукового контенту в грі або додатку.

Асети в Unity можуть бути імпортовані з різних джерел, таких як зовнішні файли, сторонні програми для моделювання або анімації, або можуть бути створені прямо в середовищі Unity за допомогою вбудованих інструментів. Вони організовані в ієрархічну структуру в проекті Unity, що дозволяє легко управляти ними та використовувати їх в різних частинах проекту.

Загалом, асети є ключовою складовою будь-якого проекту в Unity і дозволяють розробникам швидше створювати контент для своїх ігор або додатків.

Для розробки додатку було ухвалено рішення використати асет “Modern UI Pack” для спрощення створення візуальної частини додатку, а також асет “FlatCalendar” для додавання календаря, щоб не спускатися на рівень бізнес логіки.

FlatCalendar - це інструмент, який забезпечує можливість відображення календаря у веб-додатках або мобільних додатках. Він включає в себе різноманітні компоненти, призначені для різних аспектів роботи з календарем [35].

Modern UI Pack - це набір інтерфейсних компонентів для розробки сучасних та естетично приємних інтерфейсів користувача в програмах і іграх. Цей пакет компонентів дозволяє розробникам швидко та ефективно створювати зручні та привабливі UI для своїх продуктів [36].

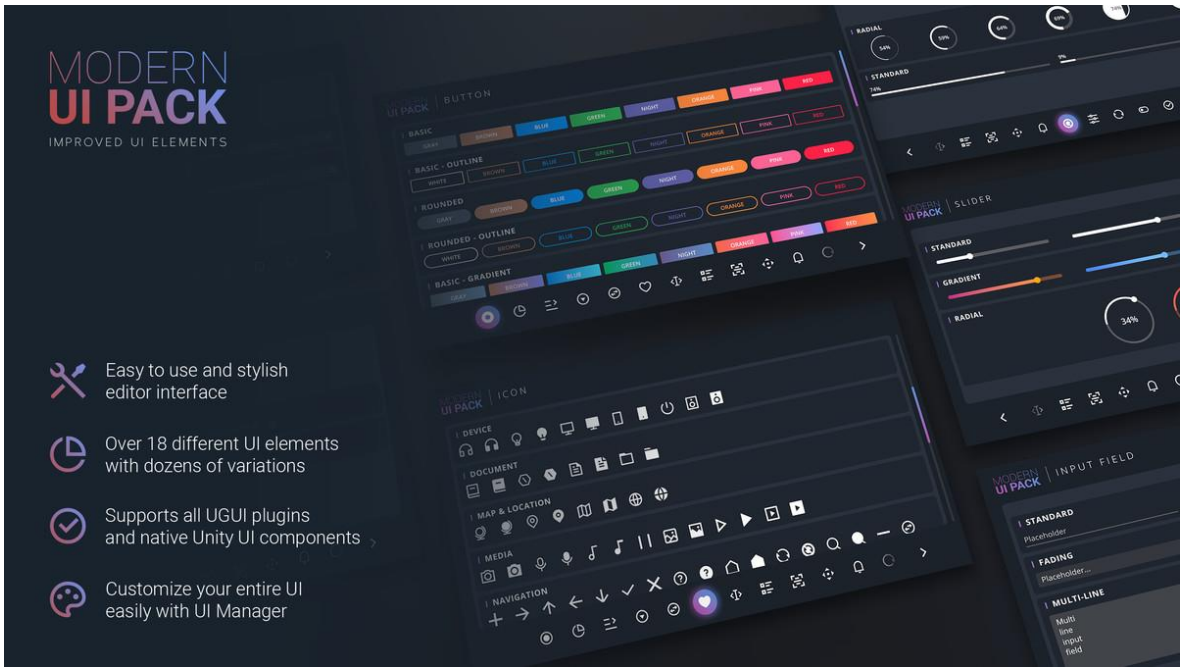

Рис. 2.7 Приклади шаблонів доступних в Modern UI pack [36]

У склад Modern UI Pack входять різноманітні компоненти, такі як кнопки, поля введення, списки, вікна, вкладки, повідомлення та інші елементи інтерфейсу. Кожен з цих компонентів має свої унікальні властивості та налаштування, що дозволяє розробникам кастомізувати їх відповідно до потреб свого проекту.

2.2 Створення автоматизованої системи супроводу навчального процесу

2.2.1 Вибір формату. Створення та налаштування проєкту

Один з найважливіших аспектів на початковому етапі - вибір формату проєкту, що впливає на продуктивність і візуальну якість кінцевого продукту. У Unity доступні кілька варіантів рендерингу, кожен з яких має свої особливості та переваги.

Розглянемо кожен з них:

Built-in Render Pipeline - це стандартний формат, який був доступний у Unity до впровадження URP і HDRP. Він менш гнучкий у налаштуваннях, але досі використовується для простих або менш вимогливих проектів [37].

HDRP (High Definition Render Pipeline) - використовується для високоякісних візуалізацій на ПК і консолях, які потребують значних ресурсів. Підходить для проектів з високою деталізацією графіки [37].

URP (Universal Render Pipeline) – цей формат забезпечує оптимальний баланс між продуктивністю та якістю графіки. URP підтримує широкий спектр платформ, включаючи мобільні пристрої, і надає розширені можливості налаштування освітлення та тіней. Це дозволяє створювати візуально привабливі додатки з низькими системними вимогами [37].

Для роботи було обрано URP систему, оскільки вона є найбільш збалансованою, та в залежності від потужності девайса користувача дозволяє знайти баланс між якістю візуалу та продуктивністю додатку.

Після створення проєкту було виставлено оптимальні конфігурації такі як роздільна датність екрану та висока якість деталізації візуальних елементів, а також завантажені всі необхідні ресурси, такі як пакети графіки, та пакети Firebase для підключення “Firebase Realtime Data Base” та “ Firebase Autentification”, а також налаштовані конфігурації самого пакету Firebase, який отримав прив’язку до акаунту через індивідуальне ID проєкту.

2.2.2 Реалізація базового графічного інтерфейсу для подальшого тестування додатку

Під час розробки додатку одним із ключових етапів є створення базового графічного інтерфейсу (UI), який дозволяє користувачам взаємодіяти з додатком і тестувати його основні функції. На цьому етапі реалізації додаються основні елементи інтерфейсу, які допомагають забезпечити зручний доступ до різних функцій додатку та перевірити їх роботу в реальних умовах.

На даному етапі було додано такі елементи базового графічного інтерфейсу:

- Головне меню - є центральним елементом, що надає доступ до всіх основних функцій додатку. Воно включає наступні компоненти:

- Кнопка “Навчальні групи” - надає доступ до управління навчальними групами.
- Кнопка “Дисципліни” - відкриває сторінку з переліком доступних навчальних дисциплін.
- Кнопка “Розклад” - відкриває розділ з календарем та розкладом занять.
- Кнопка “Вихід” - завершує сеанс користувача і повертає на сторінку входу.

Рис 2.8 Результат реалізації інтерфейсу головного меню

- Форма реєстрації та автентифікації - забезпечує безпеку доступу до додатку, дозволяючи користувачам створювати акаунти і входити в систему:
  - Поле вводу логіну - дозволяє користувачу вводити свій логін або адресу електронної пошти.
  - Поле вводу паролю - захищене поле для введення паролю.
  - Кнопка “Зареєструватися” - відкриває форму для створення нового акаунту.

- Кнопка “Увійти” - виконує автентифікацію користувача і переводить його на головну сторінку додатку.

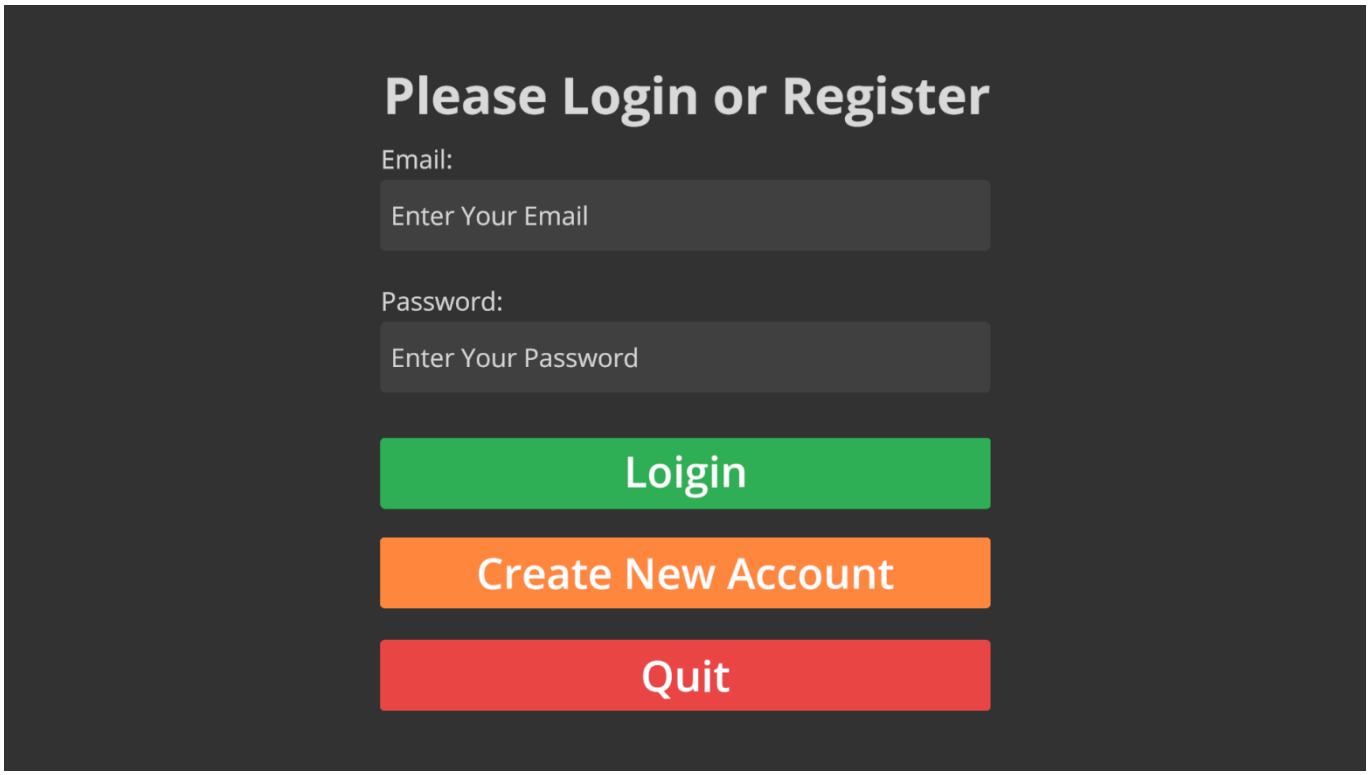

Рис 2.9 Результат реалізації інтерфейсу форми реєстрації та автентифікації

- Вкладка “Календар” дозволяє користувачам планувати свої заняття та заходи:
  - Навігаційні кнопки “Назад” і “Вперед” - дозволяють переходити між місяцями та роками.
  - Поточна дата - відображає поточний місяць і рік.
  - Список днів місяця - дозволяє користувачам обирати конкретний день для додавання уроків чи заходів.
  - Форма додавання уроку - містить поля для вибору дисципліни, навчальної групи, часу уроку і частоти повторення.

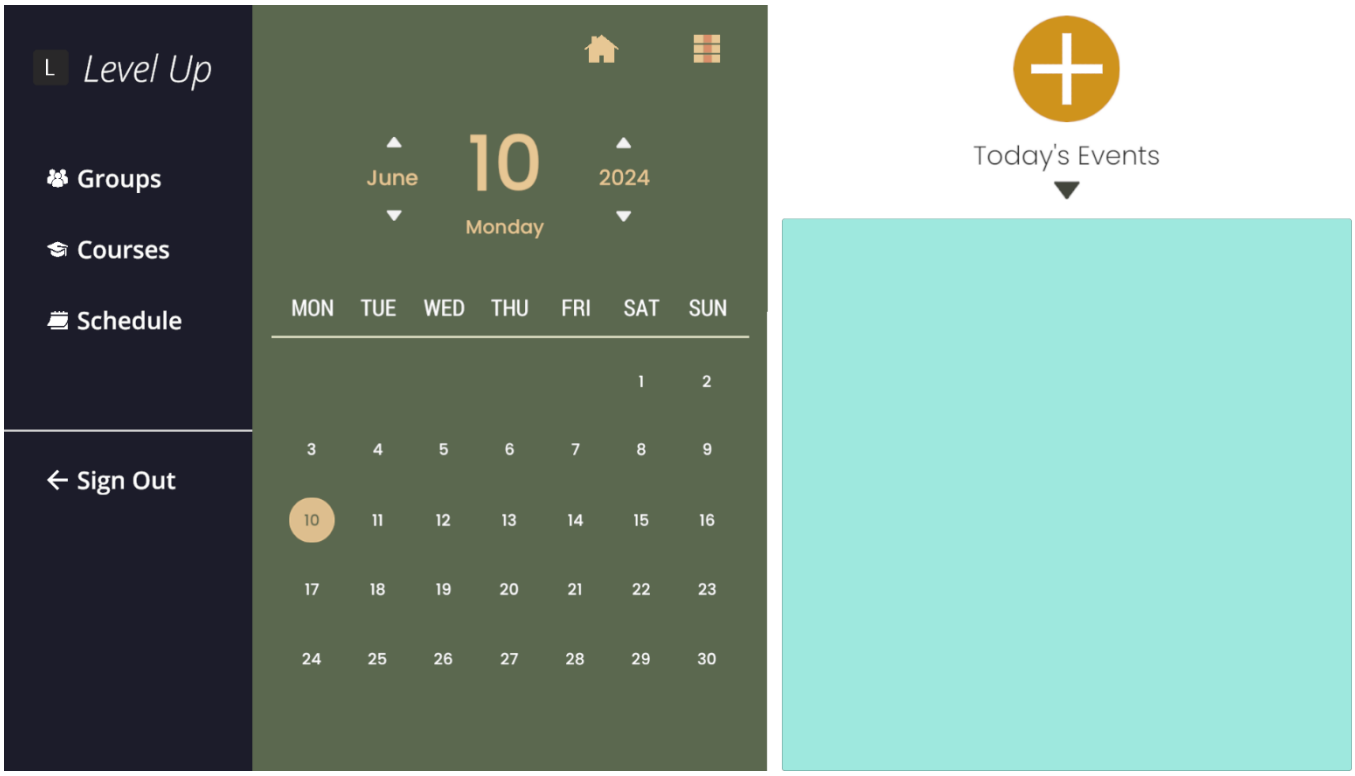

Рис 2.10 Результат реалізації інтерфейсу вкладки “Календар”

- Вкладка “Групи” для управління навчальними групами
  - Список існуючих груп - відображає всі наявні групи.
  - Кнопка “Додати групу” - відкриває форму для створення нової навчальної групи.
  - Кнопка “Редагувати групу” - відкриває форму редагування навчальної групи.
  - Кнопка “Видалити групу” - дозволяє видалити обрану групу.
  - Форма редагування групи - містить поля для зміни назви групи та списку студентів.

Рис 2.11 Результат реалізації інтерфейсу вкладки “Групи”

- Вкладка “Дисципліни” дозволяє керувати доступними навчальними дисциплінами:
  - Список дисциплін - відображає перелік наявних дисциплін з можливістю фільтрації та пошуку
  - Кнопка “Додати дисципліну” - відкриває форму для додавання нової дисципліни.
  - Форма редагування дисципліни - містить поля для зміни назви і вартості дисципліни.
  - Кнопка “Видалити дисципліну” - видаляє обрану дисципліну з бази даних.

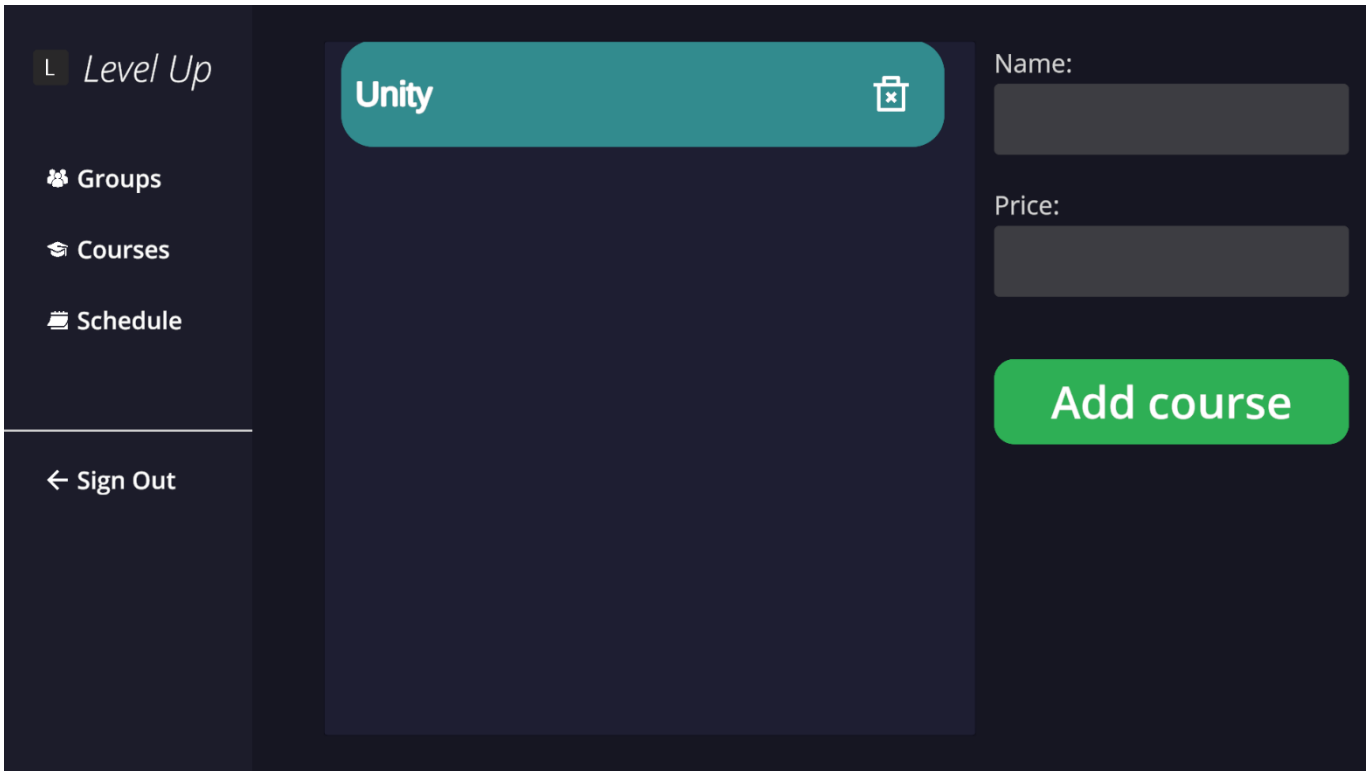

Рис 2.12 Результат реалізації інтерфейсу вкладки “ Дисципліни”

Розроблений графічний інтерфейс дозволяє користувачам взаємодіяти з системою і тестувати її функціонал. Використання інтуїтивно зрозумілих елементів інтерфейсу забезпечує зручний доступ до всіх основних функцій додатку і сприяє покращенню користувацького досвіду. Це також дозволяє виявляти і виправляти помилки на ранніх етапах розробки, що сприяє швидкості розробки.

2.3 Програмування автоматизованої системи супроводу навчального процесу

2.3.1 Дослідження різниці між компонентами та класами

У подальшій розробці додатку, буде використано поняття “класи” та “компоненти”, розуміння різниці між цими поняттями допоможе краще розуміти архітектуру додатку.

Класами називають окремі об'єкти або елементи, які можуть працювати самостійно, без необхідності прив'язки до інших об'єктів або елементів інтерфейсу.

Ці класи виконують конкретні завдання та можуть існувати незалежно від інших частин програми. Вони забезпечують логічну структуру додатку і використовуються для зберігання та обробки даних [38].

Компонентами називають скрипти або модулі, які прив'язуються до елементів інтерфейсу та забезпечують їхню поведінку. Вони відповідають за інтерактивність додатку та забезпечують зв'язок між логікою програми (класами) та інтерфейсом користувача. Вони дозволяють реалізувати взаємодію користувача з додатком, забезпечуючи відгук на його дії та зміну стану інтерфейсу [39].

Розрізнення класів та компонентів допоможе структурувати код, розподілити обов'язки між різними частинами додатку і зробити його більш зрозумілим та підтримуваним. Класи будуть відповідати за зберігання і логічну обробку даних, а компоненти - за їхню інтерактивну реалізацію та взаємодію з користувачем.

2.3.2 Отримання даних з інтерфейсу користувача

Зчитування даних є однією з найважливіших функцій у розробці сучасних додатків. Воно забезпечує можливість отримувати і обробляти інформацію, яку вводять користувачі. За зчитування даних в додатку відповідає клас InputManager.

InputManager - це компонент, який відповідає за обробку введення користувачів з полів InputField у додатку. Його основна роль полягає в тому щоб зчитувати дані з полів “InputField”, які використовуються для введення текстової інформації користувачами. Наприклад, коли користувач вводить своє ім'я, електронну пошту чи інші дані, InputManager отримує ці значення для подальшої обробки в додатку.
Основні завдання InputManager:

- Забезпечити коректне зчитування даних з полів введення та їх передання в інші частини додатку для подальшої обробки.
- Виконувати базову перевірку введених даних на відповідність певним критеріям, наприклад, перевірку коректності формату введеної електронної пошти або номера телефону. Вона допомагає швидко виявити найпростіші помилки.

- Після зчитування та базової перевірки даних InputManager передає їх до інших компонентів чи класів додатку, які відповідальні за подальшу обробку інформації, наприклад за її занесення до бази даних.

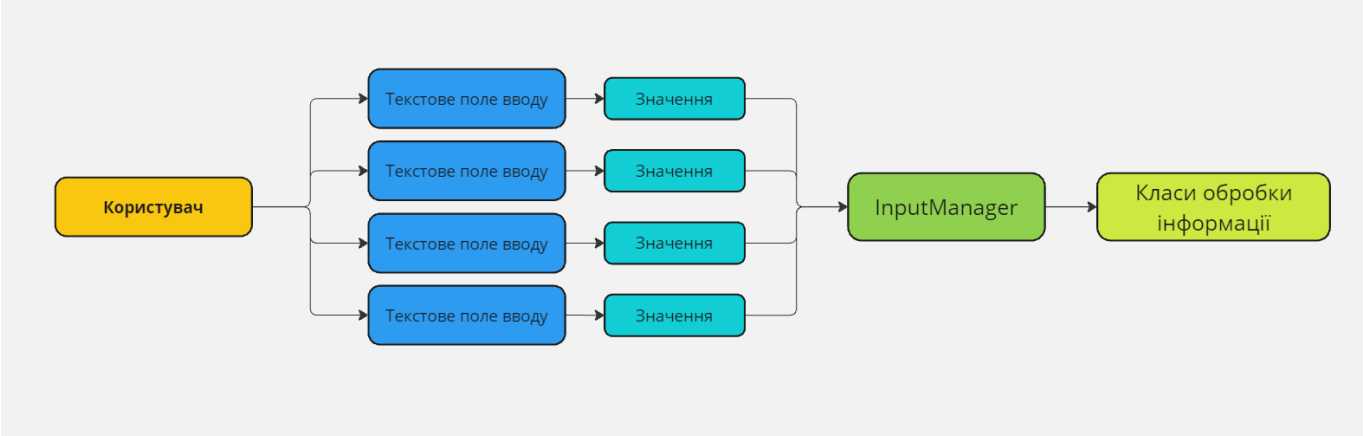

Рис 2.13 Принцип роботи InputManager

2.3.3 Навігація між вікнами додатку

Якісна система переходу між різними вікнами додатку є важливою складовою для забезпечення задоволення користувачів і ефективного використання програмного забезпечення. Вона впливає на досвід користувача, загальний вигляд та функціональність додатку.

Експертність додатку також відображається у якості системи переходів. Погано організований або неефективний перехід може залишити враження про непрофесійний підхід до розробки програмного забезпечення.

Естетика грає важливу роль у відчутті зручності та привабливості додатку. Гладкі та естетичні переходи доповнюють загальний дизайн і створюють приємне візуальне сприйняття.

Клас Window Manager відповідає за управління вікнами в додатку. Він забезпечує можливість відкриття будь-яких вікон у програмі та контролює їхнє відображення та поведінку.

Основні функції класу WindowManager:

- Відкриття вікон - компонент дозволяє відкривати нові вікна в додатку за допомогою вбудованих функцій Unity, які дозволяють змінювати активність об’єктів, тим самим надаючи можливість відмалювати або навпаки приховати певний ігровий об’єкт на екрані.
- Закриття вікон - WindowManager забезпечує можливість закриття вікон.
- Інформація про успішність операцій - клас надає інформацію про те, чи вдалося виконати запит на відкриття вікна за допомогою вбудованих обробників подій мови програмування C#, та викликає відповідні події, які повідомляють про успішність операцій.

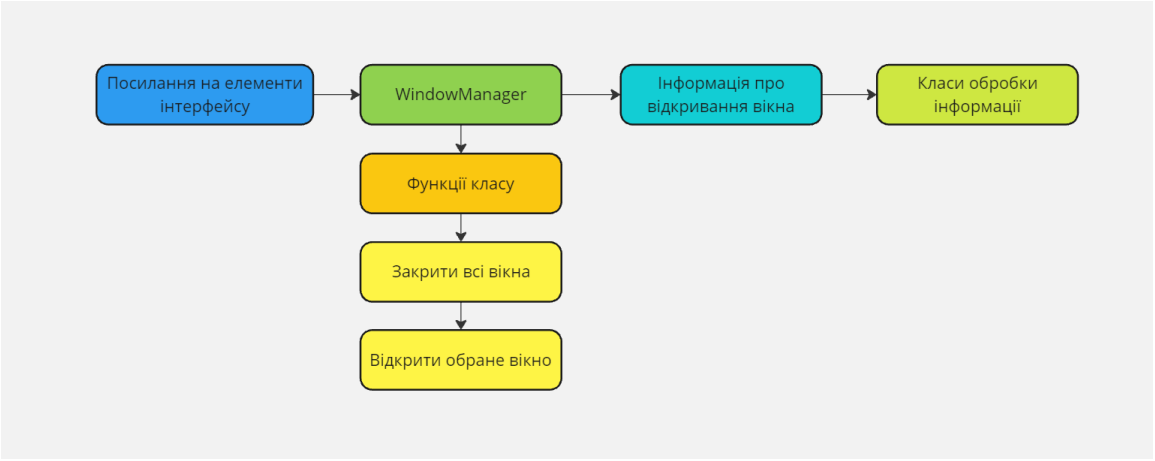

Рис 2.14 Принцип роботи WindowManager

2.3.4 Взаємодія із базою даних

Клас, який взаємодіє з базою даних, є одним із найважливіших будь-якого програмного додатка, який потребує збереження, оновлення та отримання даних. Його значення полягає в ефективній роботі з інформацією, що зберігається в базі даних, забезпеченні надійності та безпеки даних.

Перш за все, клас, який взаємодіє з базою даних, відповідає за зберігання інформації, необхідної для функціонування програми. Це можуть бути дані про користувачів, товари, послуги, історія транзакцій, налаштування додатка тощо.

Завдяки цьому класу, програма може звертатися до бази даних, щоб отримувати потрібні дані у будь-який момент часу та в будь-якій частині додатка.

Клас FirebaseManager виступає як важливий посередник між програмним додатком та Firebase - платформою, що надає послуги хмарного зберігання та обробки даних. Його основна роль полягає в забезпеченні взаємодії між додатком та базою даних Firebase, управлінні запитами та обробці даних.

Основні функції класу FirebaseManager:

- Реєстрація нових користувачів - клас FirebaseManager відповідає за процес реєстрації нових користувачів у системі. Він приймає введені користувачем дані з InputManager, перевіряє їх додатково на валідність та додає нових користувачів до бази даних Firebase.
- Автентифікація користувачів - клас FirebaseManager забезпечує можливість автентифікації користувачів у додатку за допомогою різних методів аутентифікації, таких як електронна пошта та пароль.
- Додавання та оновлення даних - цей клас дозволяє додавати нові записи до бази даних Firebase, такі як заняття, групи, студенти, дисципліни тощо. Він також забезпечує можливість оновлення існуючих записів з метою зміни або доповнення інформації.

- Зчитування даних з бази даних - FirebaseManager здійснює зчитування даних з бази даних Firebase та передає їх до додатку для подальшого використання. Це може включати зчитування списків уроків, груп, студентів, дисциплін тощо для відображення їх у візуальному інтерфейсі додатку.
- Обробка помилок та винятків - клас Firebase Manager відповідає за обробку помилок та винятків, які можуть виникати під час взаємодії з базою даних Firebase, та надає зручний інтерфейс для обробки цих ситуацій у додатку.
- Видалення даних з бази даних - Firebase Manager надає можливість видаляти існуючі записи з бази даних Firebase. Це може включати видалення уроків, груп, студентів, дисциплін тощо.

- Надсилання подій про успішність або помилковість операцій - клас Firebase Manager може надсилати події або повідомлення про успішне або невдале виконання операцій, таких як додавання, оновлення чи видалення даних. Це дозволяє додатку реагувати на результати виконання операцій та вживати відповідних заходів.
- Оновлення даних в реальному часі - FirebaseManager може надавати можливість оновлення даних в реальному часі, що дозволяє автоматично оновлювати відображення даних у додатку при їх зміні в базі даних Firebase без необхідності в ручному оновленні сторінок або компонентів.

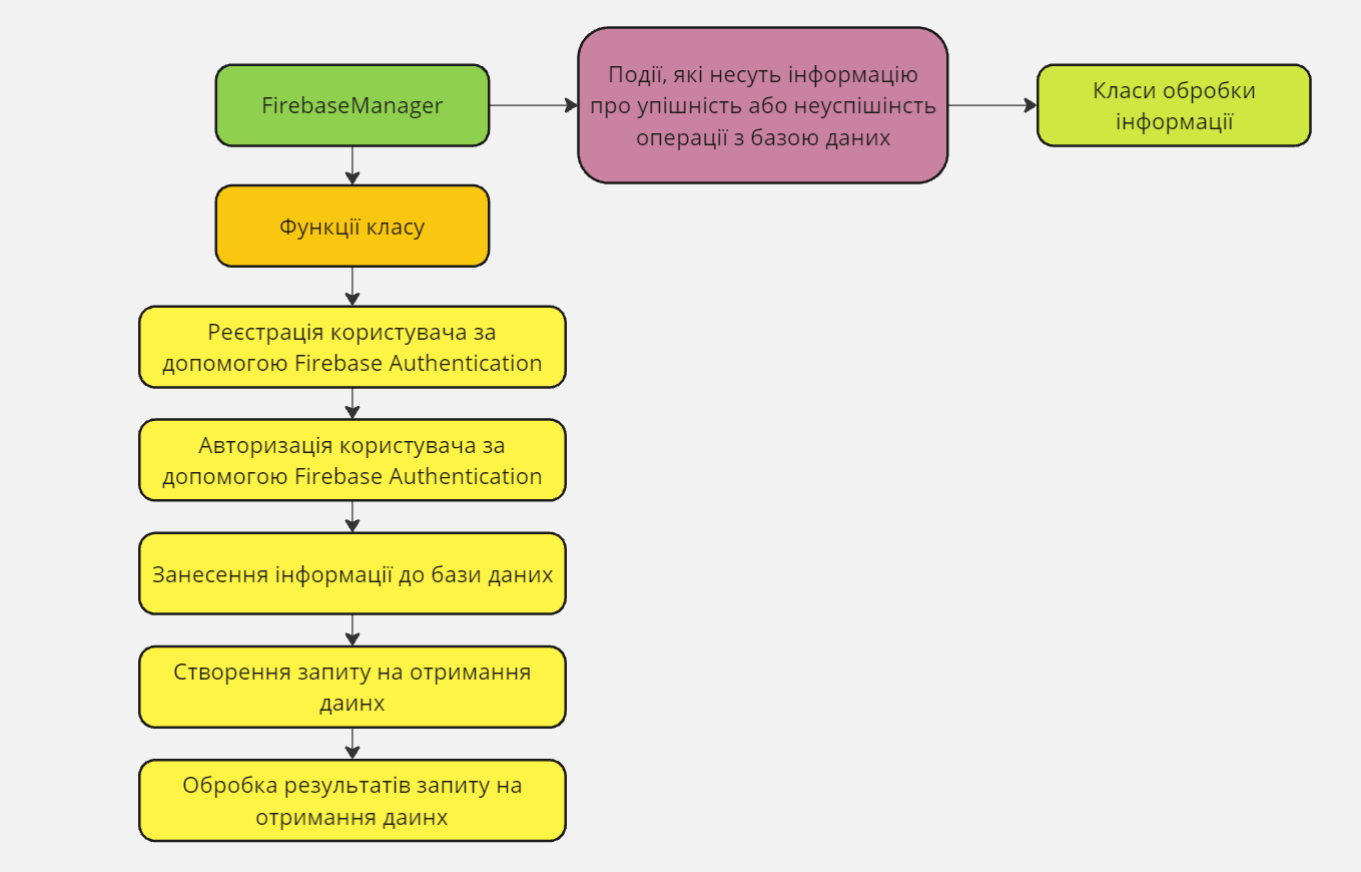

Рис 2.15 Принцип роботи FirebaseManager

2.3.4 Основні компоненти додатку

При розробці сучасного додатку важливим є створення чіткої і структурованої архітектури, де кожен компонент має свою конкретну роль і відповідальність. Це не

лише спрощує розробку, але й забезпечує зручність підтримки та масштабованість додатку. Одним із ключових аспектів є взаємодія цих компонентів з Firebase Manager, який забезпечує доступ до бази даних та інших сервісів Firebase. Тож маємо потребу у окремих компонентах додатку, кожен з яких виконує свою роль та слугує взаємозв'язком певної вкладки меню із Firebase Manager.

Основні компоненти додатку, які слугують взаємозв'язком певно вкладки із Firebase Manager, включають:

- Компонент управління дисциплінами CourseMnager - відповідає за додавання нових навчальних дисциплін, включаючи їх назви та інші параметри. Цей компонент звертається до Firebase Manager для зберігання та отримання даних про дисципліни.
- Компонент управління групами GroupManager - дозволяє створювати, редагувати та видаляти навчальні групи, додавати та видаляти студентів. Використовує Firebase Manager для роботи з даними про групи та студентів.
- Компонент управління уроками ScheduleMnager - відповідає за створення, редагування та видалення уроків. Взаємодіє з Firebase Manager для збереження та отримання даних про уроки, їх розклад і повторення.
- Компонент календаря FlatCalendar - забезпечує перегляд та управління розкладом уроків. Він взаємодіє з Firebase Manager для отримання та збереження даних про розклад, налаштування повторень і часу проведення уроків.

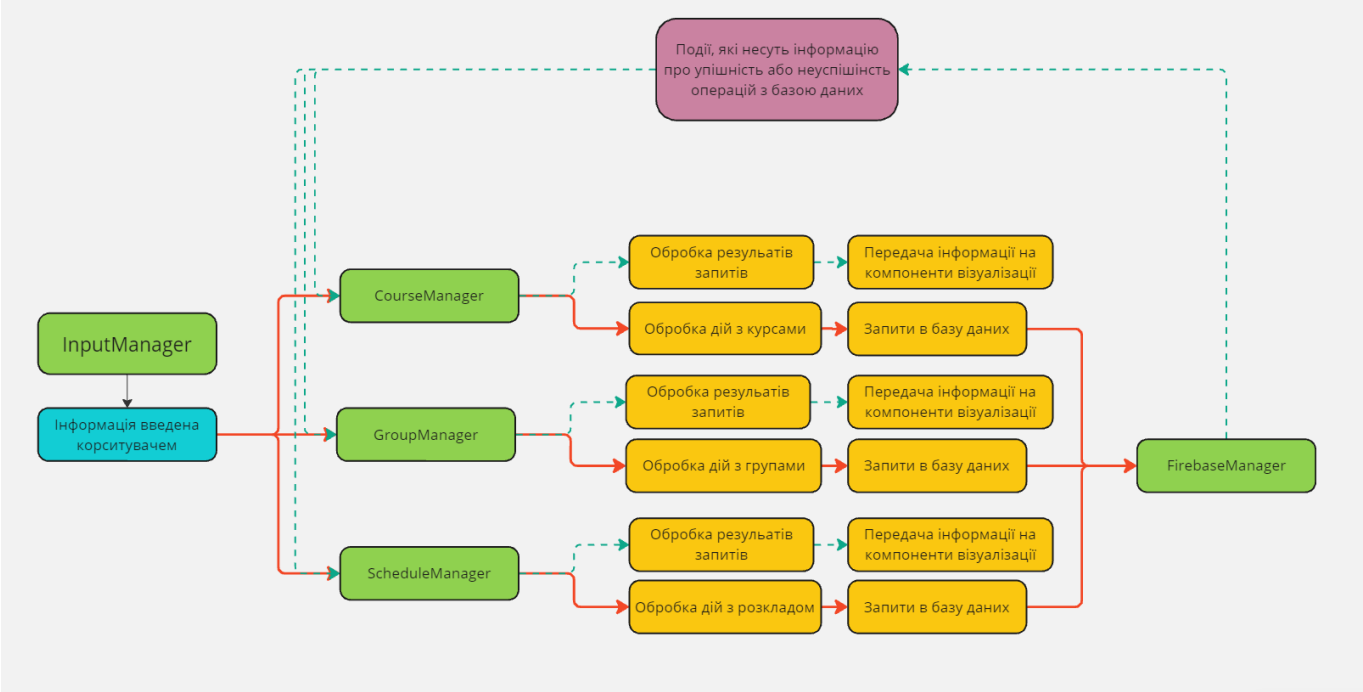

Рис 2.16 Принцип роботи основних компонентів додатку

Кожен з цих компонентів виконує свою конкретну роль і забезпечує взаємодію окремих вкладок з Firebase Manager, що дозволяє додатку залишатися гнучким і ефективним. Це є основою для створення надійного, зручного та функціонального додатку, який може ефективно обслуговувати своїх користувачів.

2.3.5 Комоненти для візуалізації інформації із бази даних

Для створення сучасного, функціонального додатку, важливо забезпечити якісну візуалізацію інформації, яка зберігається в базі даних. Це дозволяє користувачам зручно переглядати, взаємодіяти з даними та приймати обґрунтовані рішення. Розберемо всі візуальні компоненти необхідні для відображення інформації з бази даних та їхні функції.

Візуальні компоненти служать інтерфейсом між користувачем і даними, що зберігаються в базі даних. Вони створюються за наказом основних компонентів, таких як наприклад LessonManager отримують інформацію від них інформацію, і відображають її на екрані. Це дозволяє користувачам зручно працювати з даними,

переглядати їх у зручному форматі та здійснювати різні операції, такі як редагування або видалення.

Розглянемо всі візуальні компоненти в розроблюваному додатку:

- SimpleDialogWindow - компонент який отримує інформацію від компонента, який його створює та відображає на інтерфейсі просте діалогове вікно із помилками або порадами.
- SingleGroupeVisual компонент, який динамічно створюється за наказом GroupManager отримує інформацію від GroupManager, та візуалізує її на екрані, в даному випадку це просто візуалізація блоку із назвою та кнопками керування групою.
- SingleLessonVisual компонент, який динамічно створюється за наказом ScheduleManager отримує інформацію від ScheduleManager, та візуалізує її на екрані, в даному випадку це візуалізація блоку уроку із його детальною інформацією.
- SingleGridItemVisual загальний компонент, який можна використати коли потрібно просто згенерувати візуальний елемент із назвою та функціоналом видалення, до прикладу в даній роботі компонент чудово підходить для візуалізації навчальної дисципліни та ще кількох візуальних елементів.

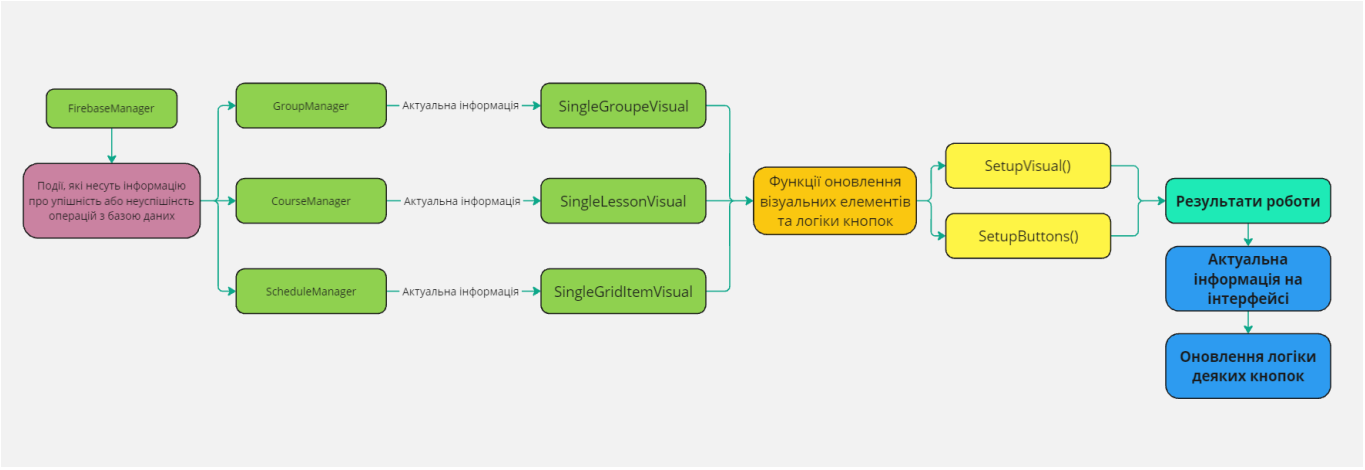

Рис 2.17 Принцип роботи компонентів візуалізації інформації

Використання візуальних компонентів для відображення інформації з бази даних є критично важливим для створення зручного і функціонального додатку. Вони забезпечують наочне представлення даних, підвищують зручність користування і ефективність взаємодії з додатком. Інтеграція з FirebaseManager дозволяє забезпечити актуальність та своєчасність даних, що відображаються, та спрощує керування ними.

2.3.6 Утиліти для автоматизації повторюваних дій

Утиліти (utility classes) у програмуванні, є класами, які забезпечують виконання загальних рутинних операцій за допомогою статичних методів. Вони надають зручний інтерфейс для вирішення повторюваних завдань, які можна використовувати в різних частинах програми без необхідності створення нових об'єктів [40].

Під час розробки виникало два види основної рутинної роботи. Перший – це очищення інтерфейсу від застарілої інформації після оновлення даних на інтерфейсі, яке виконувалося у кожному компоненті, який генерував візуальні блоки інтерфейсу згідно з інформацією отриманої з бази даних. Другий - це постійні складнощі роботи з датами, так як є постійна потреба конвертувати дати в стрічки і навпаки, а також форматування стрічок із часом занять та інших.

Для автоматизації вищеописаних завдань було розроблено два класи, які мають статичні методи доступні у будь-якому місці коду, які вирішують ці проблеми.
Було використано такі класи:
Cleaner – клас, який дозволяє очистити будь яку частину інтерфейсу із застарілою інформацією за допомогою виклику всього однієї простої функції.

```
6   public class Cleaner
7   {
        6 references
8       public static void ClearContainer(Transform container)
9       {
10          foreach (Transform item in container)
11          {
12              GameObject.Destroy(item.gameObject);
13          }
14      }
15
        1 reference
16      public static void ClearInputField(TMP_InputField inputField)
17      {
18          inputField.text = "";
19      }
20  }
```

Рис 2.17 Системний код класу Cleaner

TimeManager – клас який вміє перетворювати стрічкову інформацію на дату і навпаки, а також може форматувати тест годин, до загального стандартного вигляду, який передається на блоки візуальних компонентів.

```
6   public class TimeManager
7   {
        6 references
8       public static DateTime GetDayByString(string year_month_day)
9       {
10          string[] dateParts = year_month_day.Split('-');
11          if (int.TryParse(dateParts[0], out int year) &&
12              int.TryParse(dateParts[1], out int month) &&
13              int.TryParse(dateParts[2], out int day))
14          {
15              DateTime dateTime = new DateTime(year, month, day);
16              return dateTime;
17          }
18          else
19          {
20              Debug.LogError("Invalid date string");
21              return DateTime.Now;
22          }
23      }
24  }
```

Рис 2.18 Системний код класу TimeManager

2.4 Тестування та результати роботи розробленого додатку

2.4.1 Тестування системи реєстрації та автентифікації

Реєстрація та автентифікація – це перші процеси з якими зіштовхується користувач, який відкрив додаток. Коректна робота цих процесів є критично важливою для подальшого використання додатку юзером.

Коректність роботи систем реєстрації та автентифікації була перевірена за допомогою створення нового акаунта та входу до свого акаунта та головного меню додатка.

Для початку було відкрито меню реєстрації проведено два тести для перевірки чи валідує данні додаток. Спочатку була некоректно введена адреса електронної пошти, потім хибне затвердження паролю.

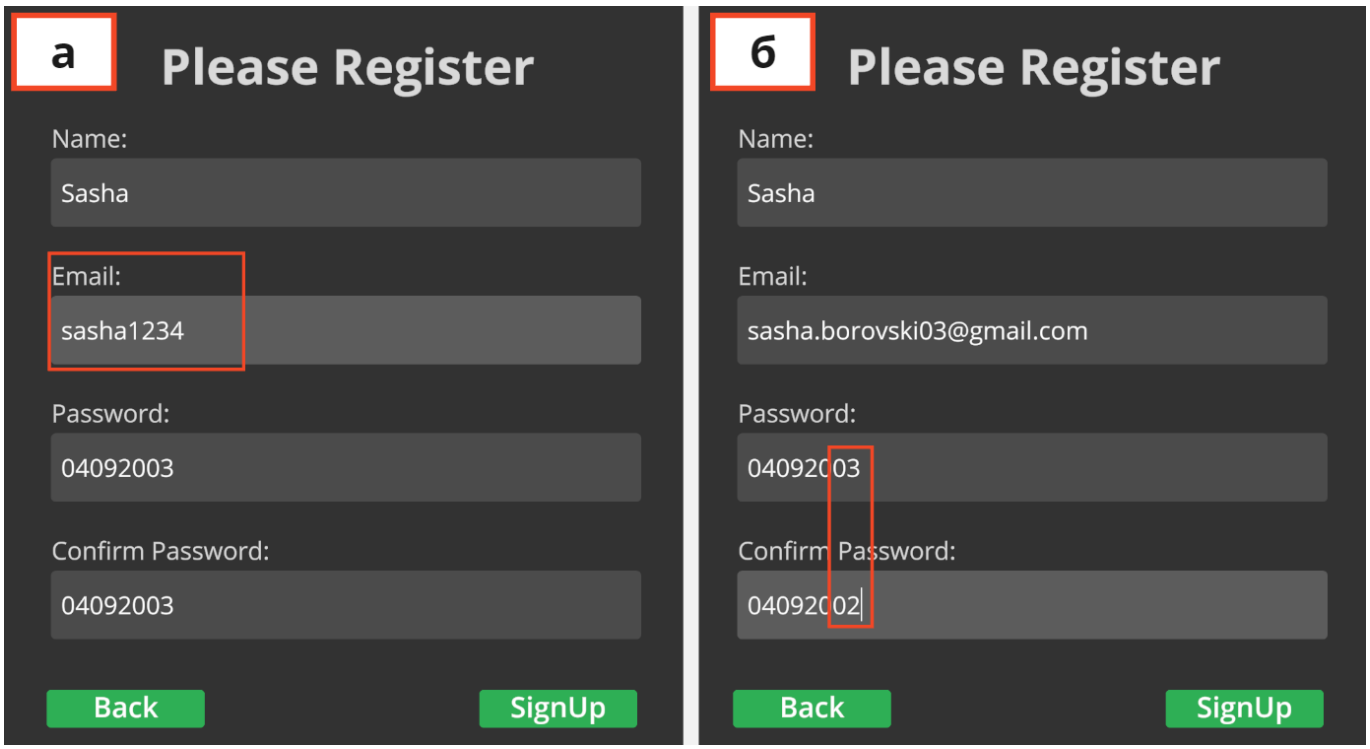

Рис. 2.19 Спроба введення хибних даних: а) введення хибної електронної адреси; б) введення хибного підтвердження паролю

Після спроби реєстрації у обох випадках було отримано вікно з помилкою, яка сигналізувала про введення хибної інформації.

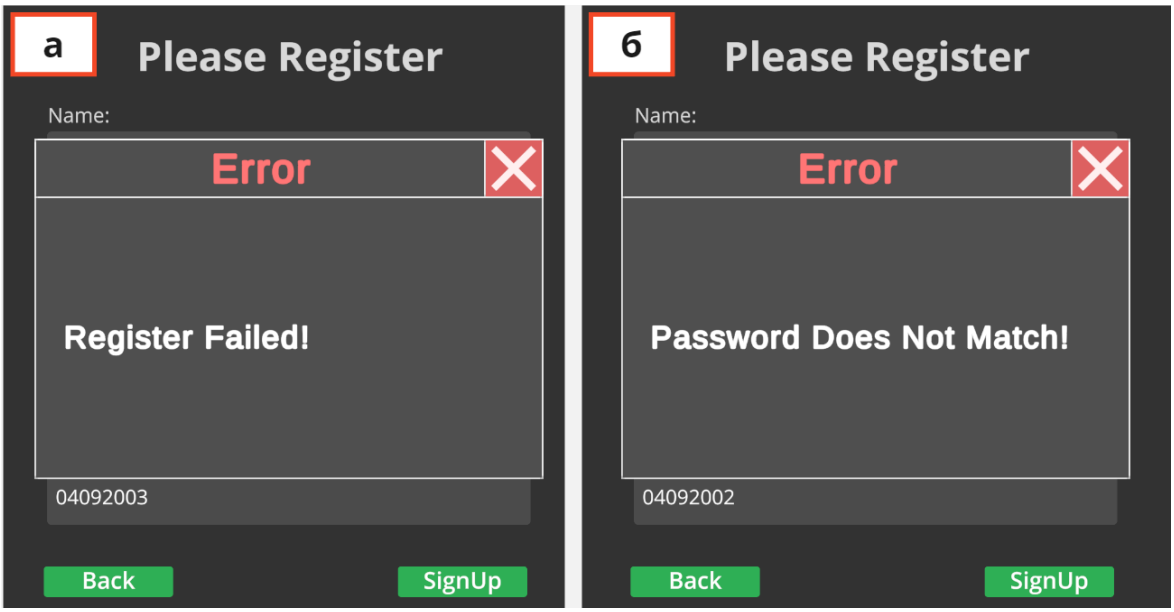

Рис. 2.20 Результати спроби реєстрації із невалідними даними: а) введення невалідного email; б) введення хибного паролю для підтвердження

Після введення валідної інформації було здійснено перенаправлення до меню автентифікації, а в базу даних було додано новий акаунт з унікальним ID та даними, які було введено при реєстрації.

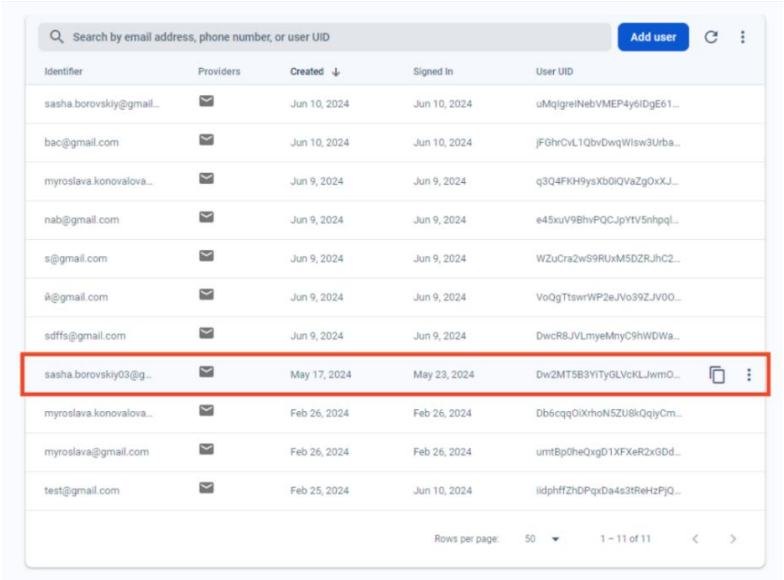

Рис. 2.21 Відображення створеного акаунту в базі даних

Наступним кроком була спроба автентифікації за допомогою створеного акаунту. У поля авторизації було введено відповідні данні і натиснута кнопка авторизації.

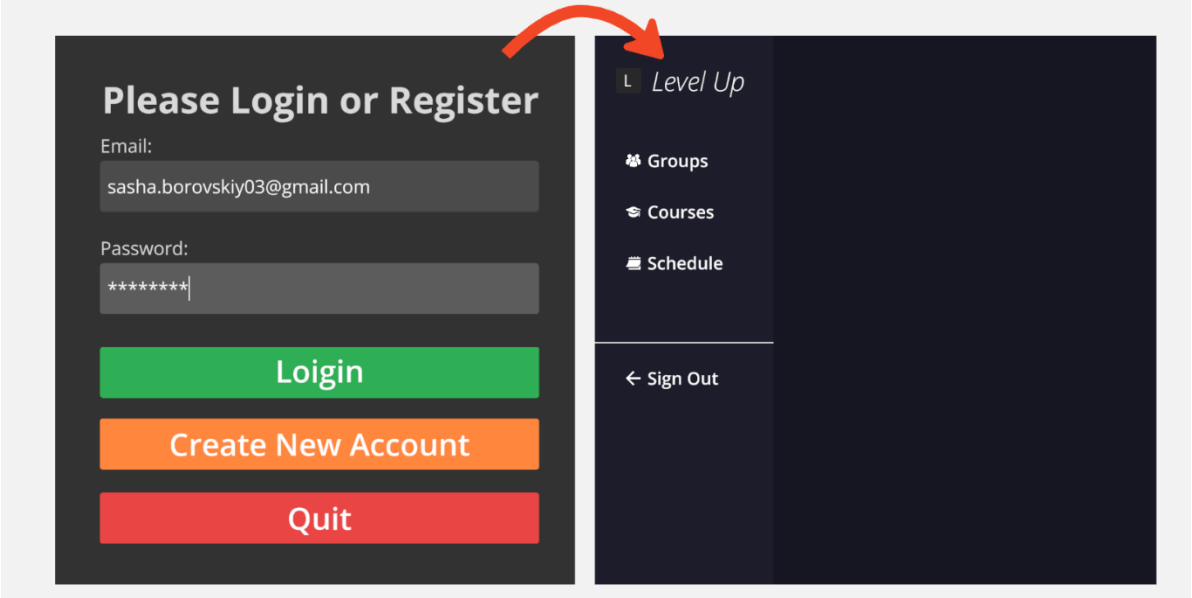

Рис. 2.22 Демонстрація успішного процесу автентифікації

2.4.2 Тестування системи контролю навчальних груп

Щоб запланувати будь-яке заняття у розкладі, спочатку потрібно обов’язково створити хоча б одну навчальну групу для якої і буде призначатися це заняття. Тому наступним кроком стало тестування системи контролю навчальних груп.

Тестування системи груп було розпочате із перевірки на валідацію. При залишені полів групи пустими, або при відсутності в групі студентів щоразу було виявлено попередження.

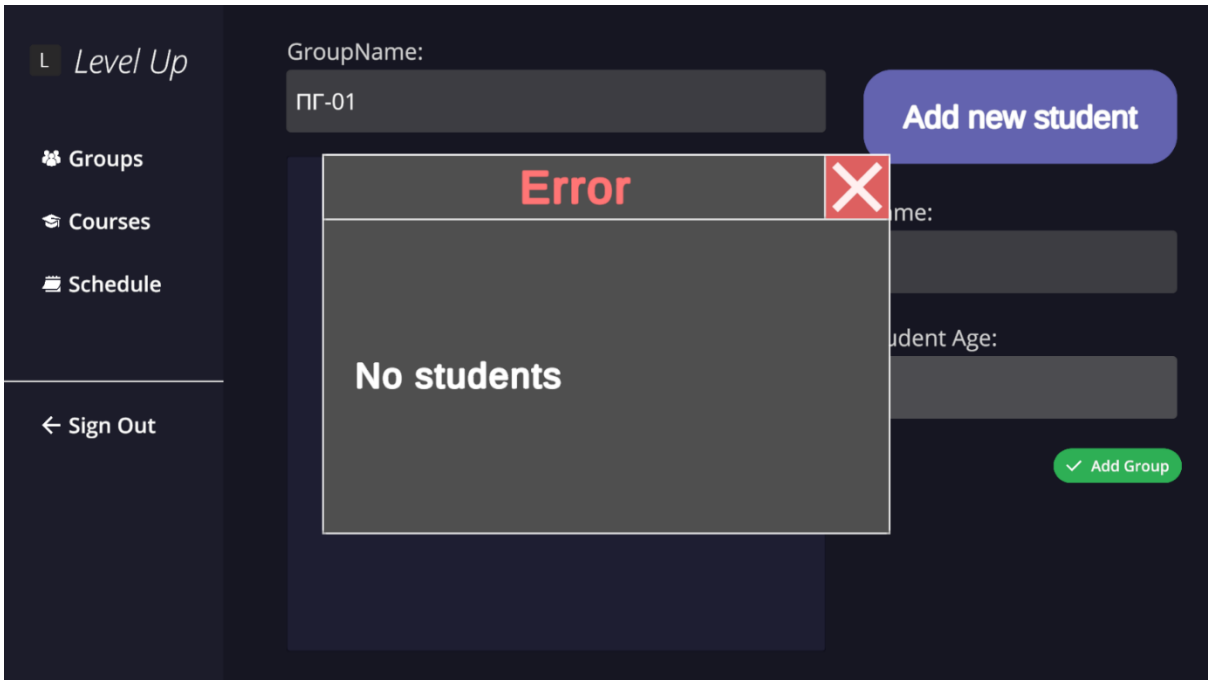

Рис. 2.23 Приклад попередження при відсутності студентів в групі

Оскільки було виявлено, що перевірки на валідність працюють успішно, далі було перевірено можливість додавання студентів. Всі студенти успішно додалися, для тестової групи було додано 4 студенти.

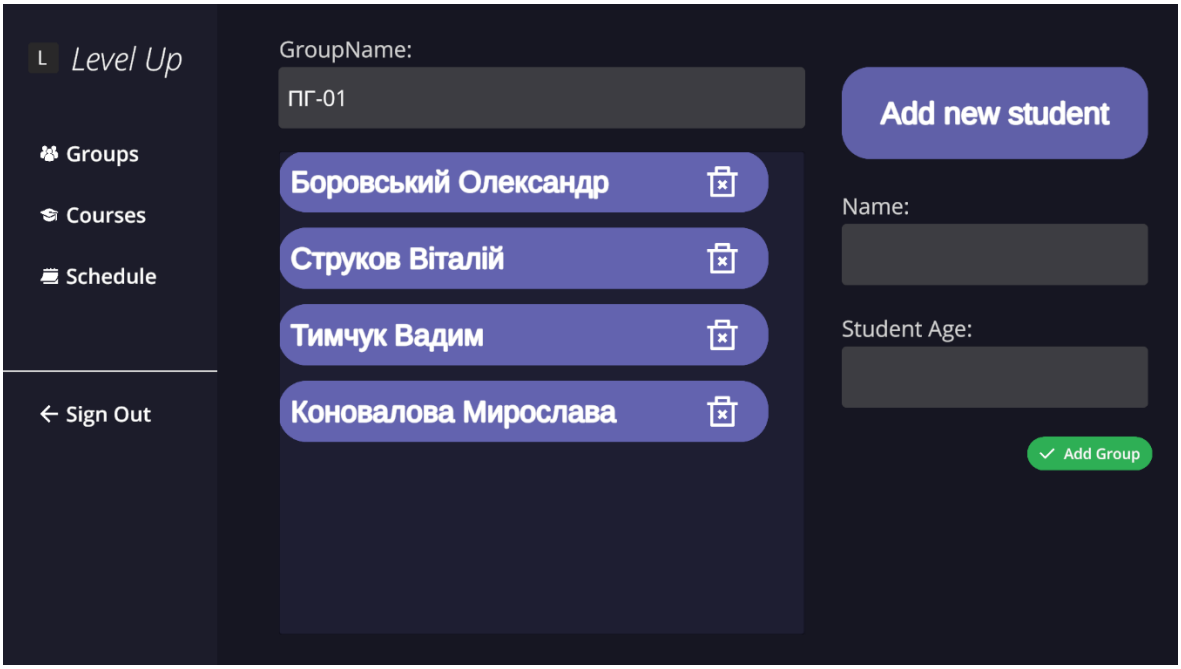

Рис. 2.24 Результати додавання студентів до групи

Після додавання студентів була виконана спроба створити групу. В результаті відбулося перенаправлення до вкладки груп. В списку груп з’явився елемент відповідний до створеної нами групи

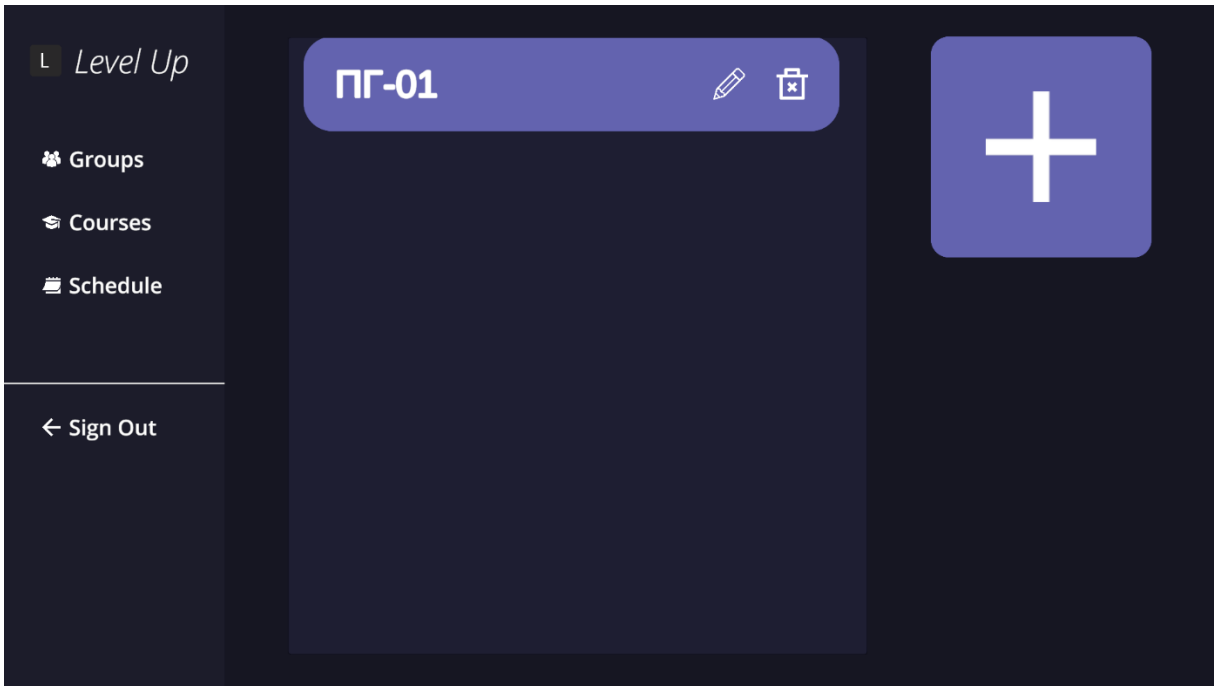

Рис. 2.25 Результат додавання групи на інтрефейсі

Окрім оновлення візуального інтрефейсу було виявлено оновлення в базі даних. У вкладці користувачів, з’явився новий список користувача із створеною нами групою, та доданими в неї студентами.

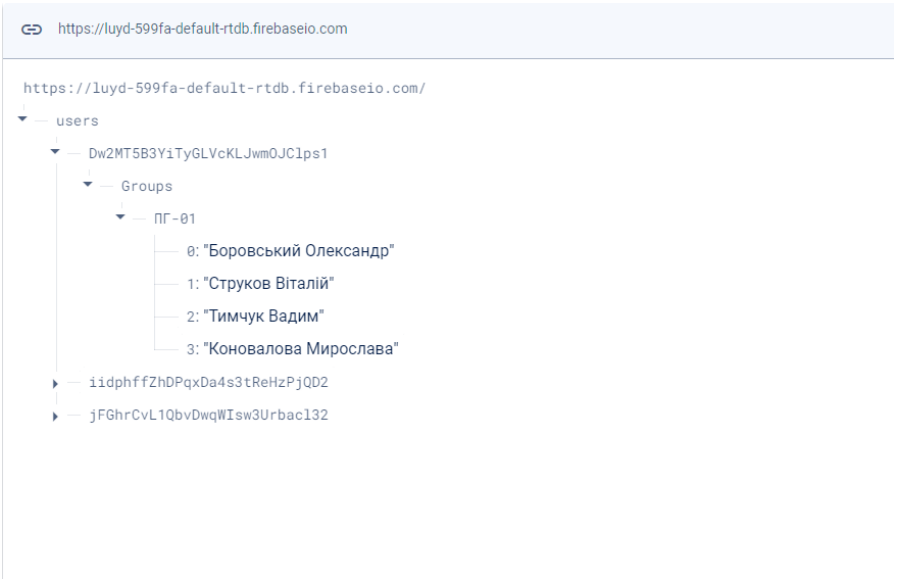

Рис. 2.26 Результат додавання групи в базу даних

Після додавання групи було виконано перевірку на можливість її редагування. Із групи був видалений один студент та його місце був доданий новий студент, після чого бузи збережені зміни. Це дозволило виконати тестування одразу двох механік: додавання та видалення студентів із групи.

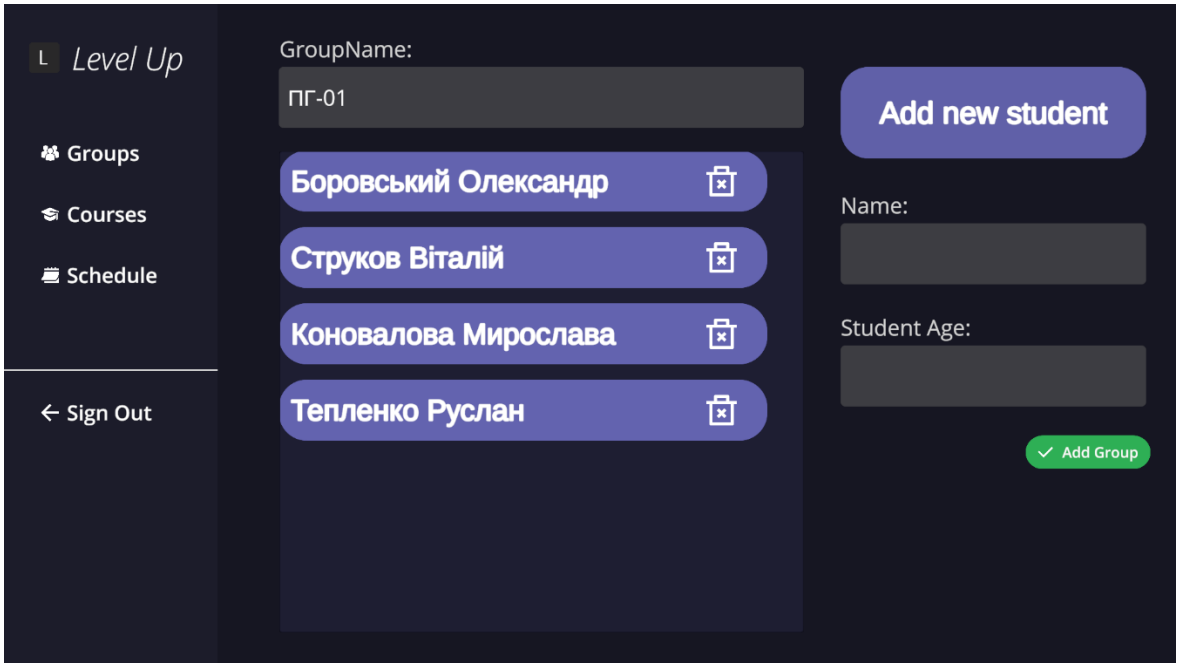

Рис. 2.27 Результат заміни студента групи на інтерфейсі

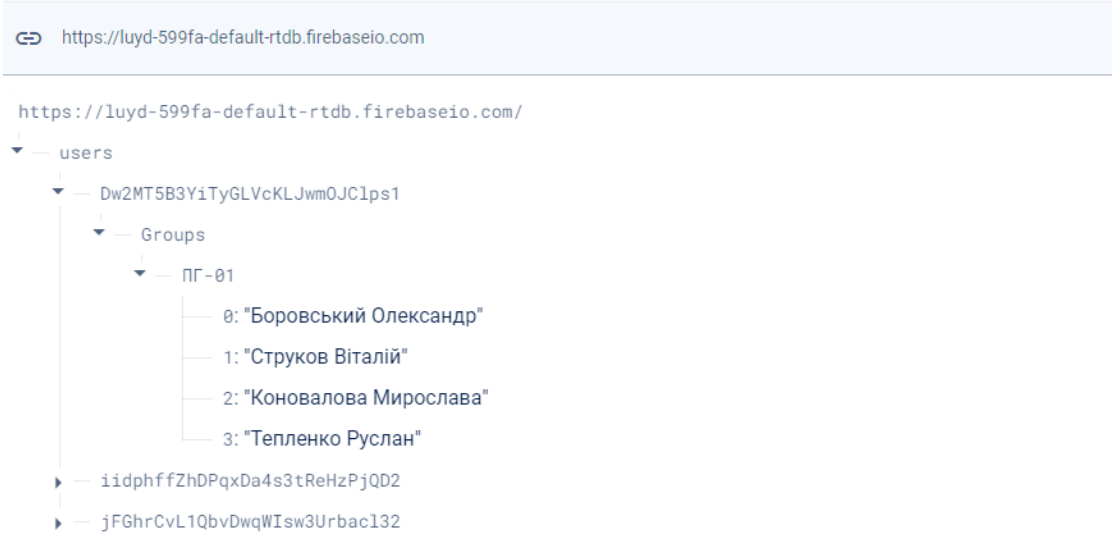

Рис. 2.28 Результат заміни студента групи на в базі даних

Результати заміни студентів було успішно оновлено як на інтерфейсі користувача, так і в базі даних.

Останньою частиною тестування системи груп стало тестування можливості видалення групи. Для даного тесту була додана та збережена ще одна група із двома студентами.

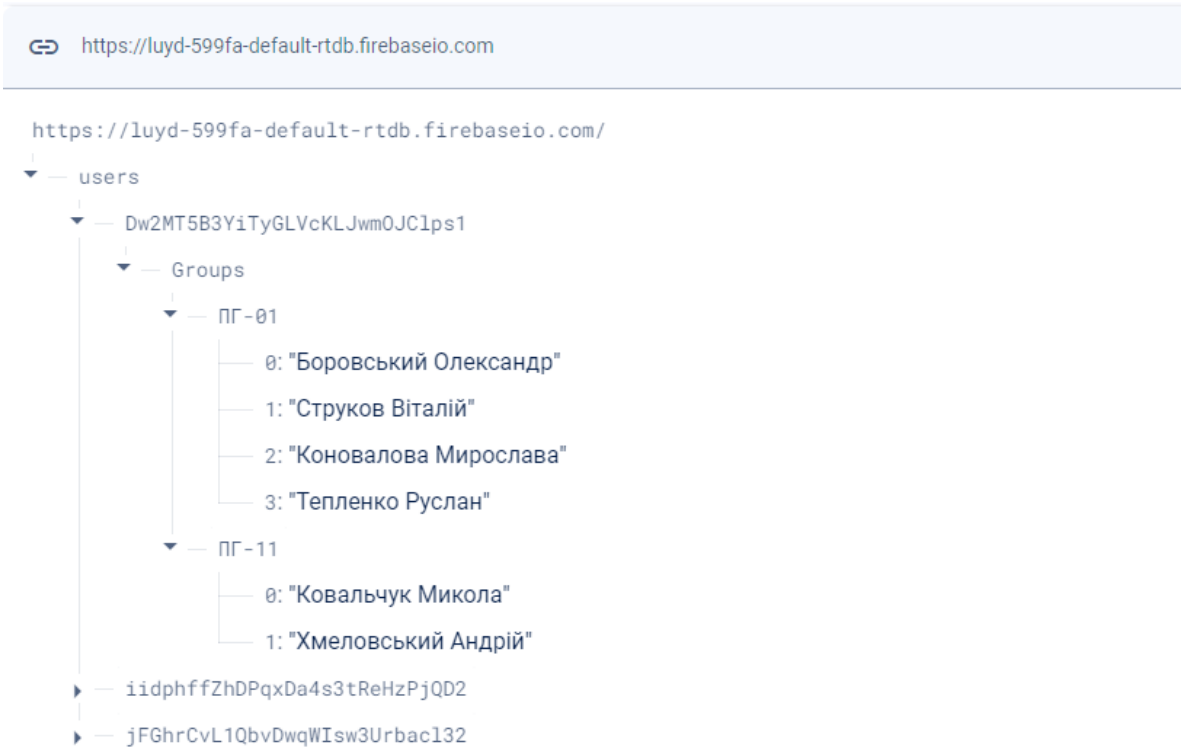

Рис. 2.29 Результат додавання нової групи в базу даних

Після додавання нової групи як і в попередньому тесті відбулося успішне оновлення інформації як на інтерфейсі так і в базі даних. Після натискання кнопку видалення також відбулося успішне оновлення інформації як на інтерфейсі так і в базі даних, та навчальна група була успішно видалена.

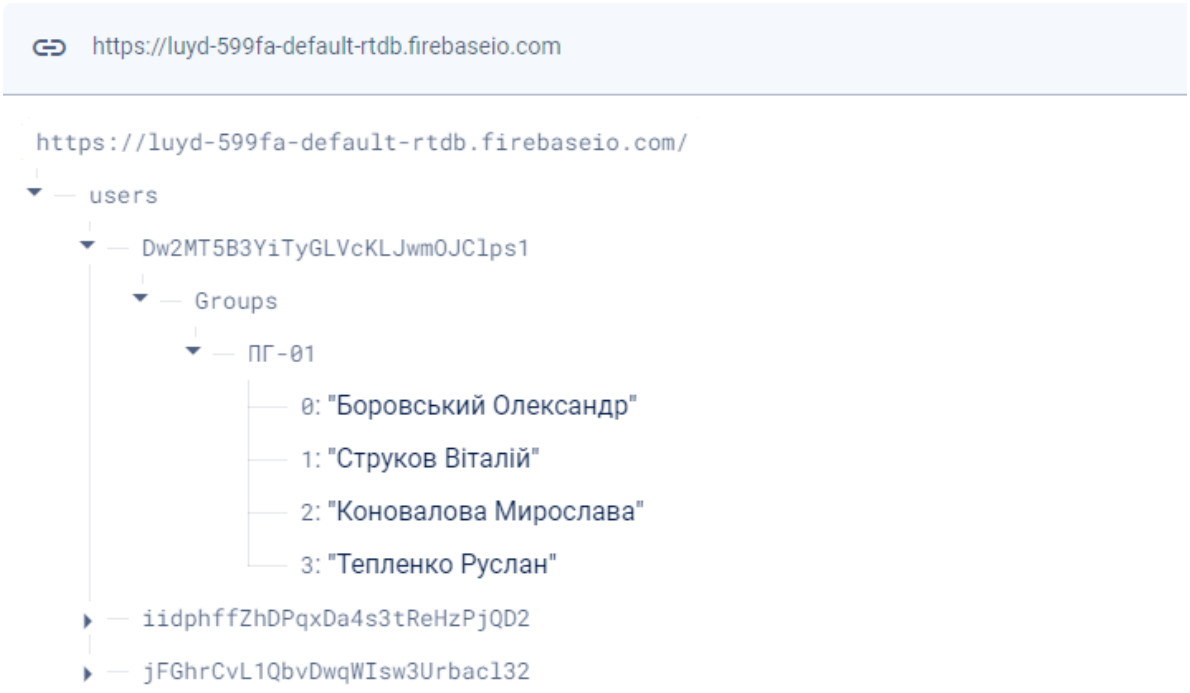

Рис. 2.30 Результат видалення групи з бази даних

2.4.3 Тестування системи контролю навчальних дисциплін

Для планування заняття у розкладі окрім створення навчальної групи, спочатку потрібно обов’язково створити хоча б одну навчальну дисципліну, яка буде прив’язуватися до відповідної групи під час створення занять в розкладі. Тому наступним етапом стало тестування системи контролю навчальних дисциплін.

Тестування системи дисциплін було розпочате із перевірки на валідацію. При залишені полів пустими. Перевірка була пройдена успішно всі результати працюють відмінно та аналогічно до того як працювала валідація в системі створення груп.

Після тестування валідації було виконано спроби додати дві навчальні дисципліни. Додавання пройшло успішно, та інформація була успішно оновлена як на інтерфейсі так і базі даних, де також можна спостерігати додавання нової підвкладки, яка відповідає за навчальні дисципліни.

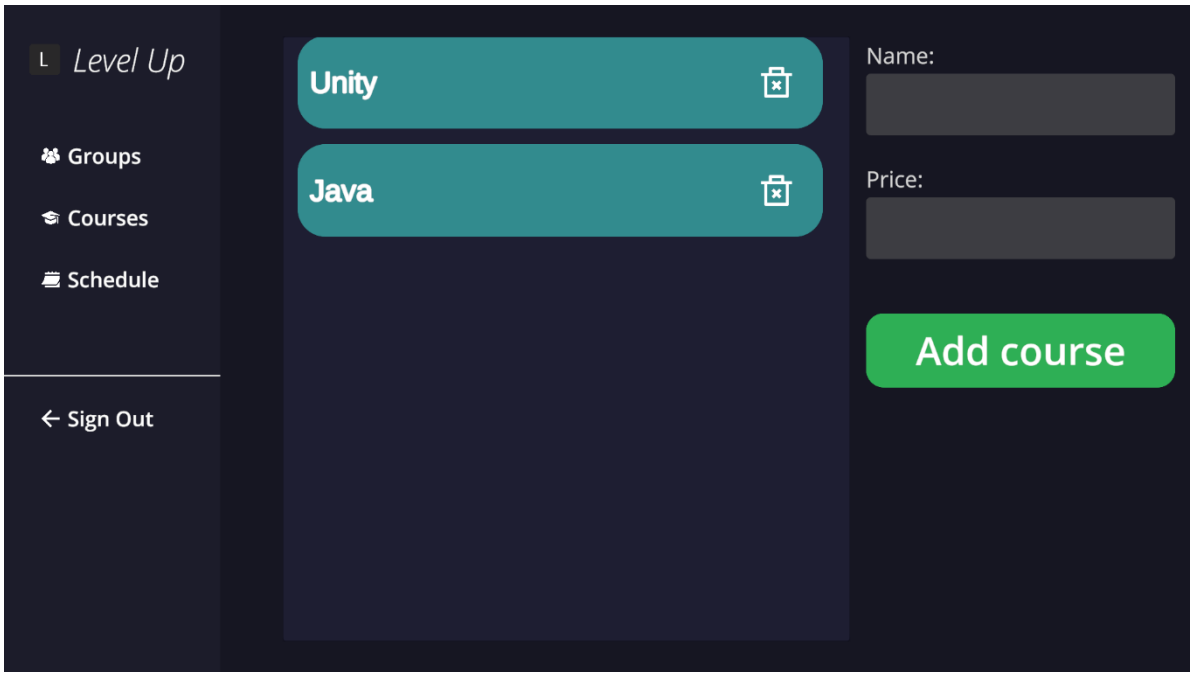

Рис. 2.31 Результат додавання дисциплін на інтерфейсі

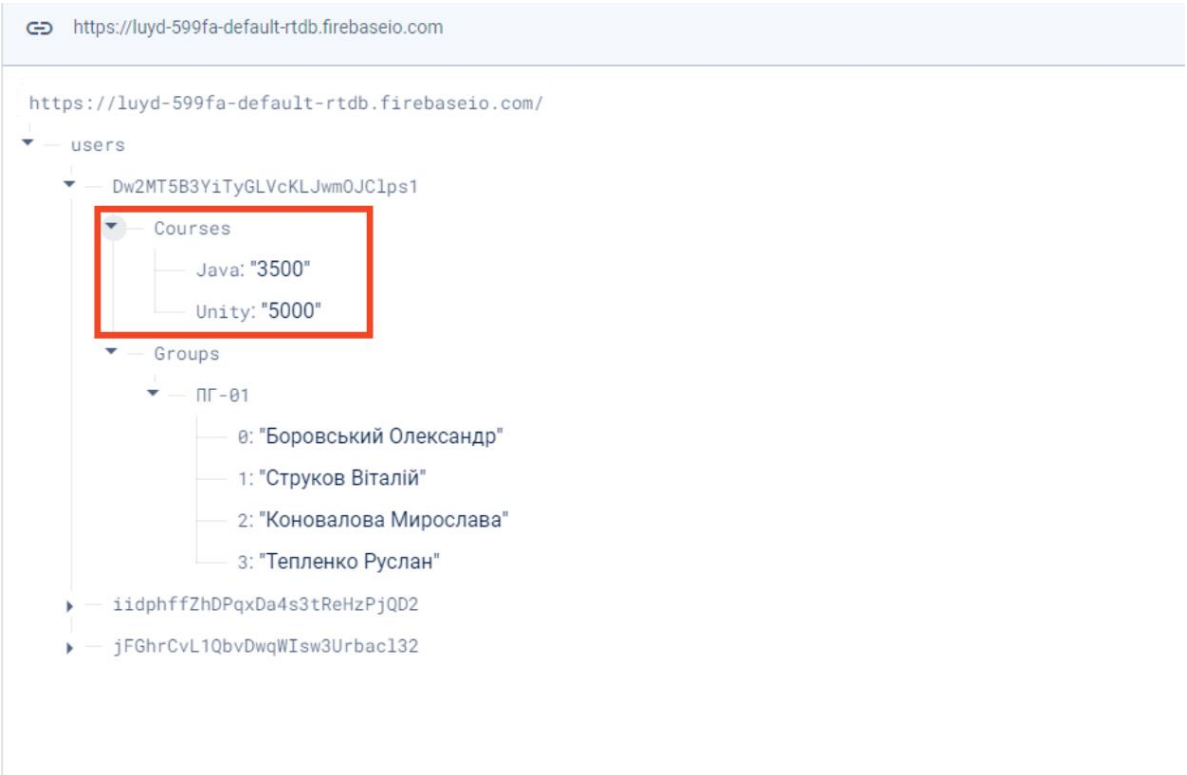

Рис. 2.32 Результат додавання дисциплін в базу даних

Для тестування функціоналу видалення групи було додано ще дві групи, та одразу ж виконано спробу видалення однієї із них. Всі операції пройшли успішно. Інформація була оновлена як на інтерфейсі так і в базі даних.

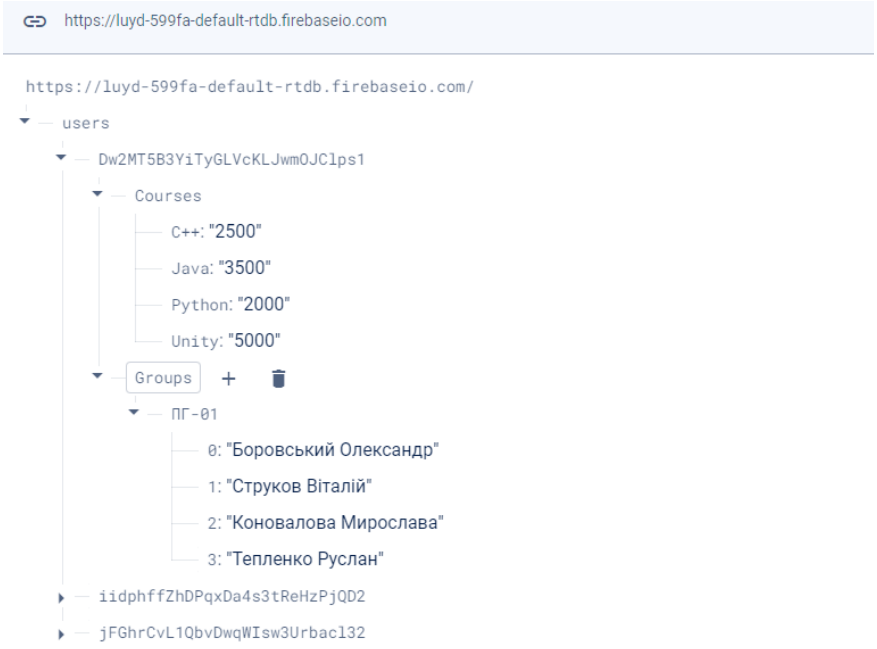

Рис 2.33 Результат додавання дисциплін в базу даних

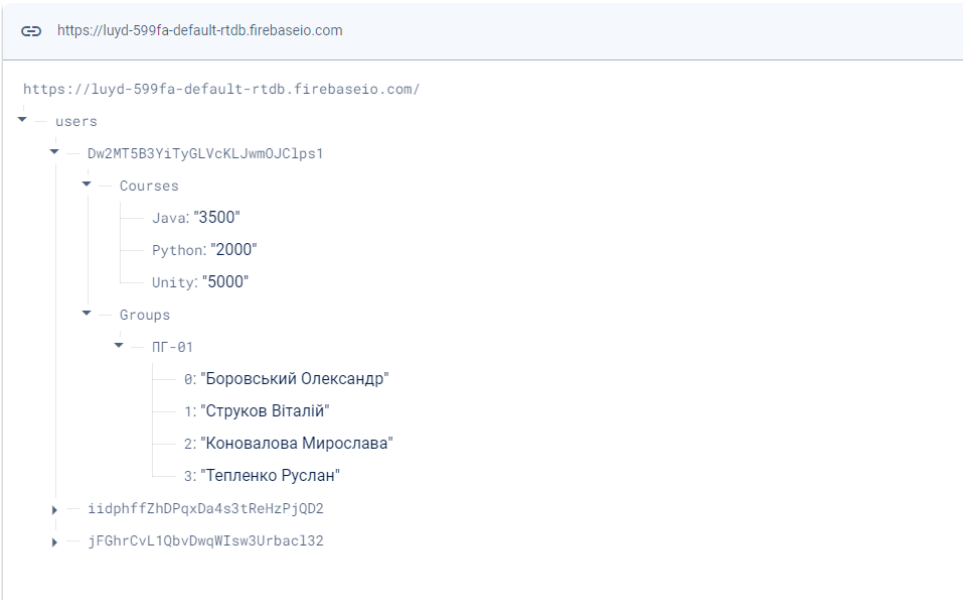

Рис 2.34 Результат видалення дисципліни із бази даних

2.4.4 Тестування системи розкладу та додавання занять

Після відкривання вкладки з розкладом, було отримано доступ до календаря, на якому автоматично обралася сьогоднішня дата. Календар задовольняє всі описані вимоги. Він має можливість обирати різні дні, а також пролистувати місяці та роки як вперед так і назад в необмеженій кількості.

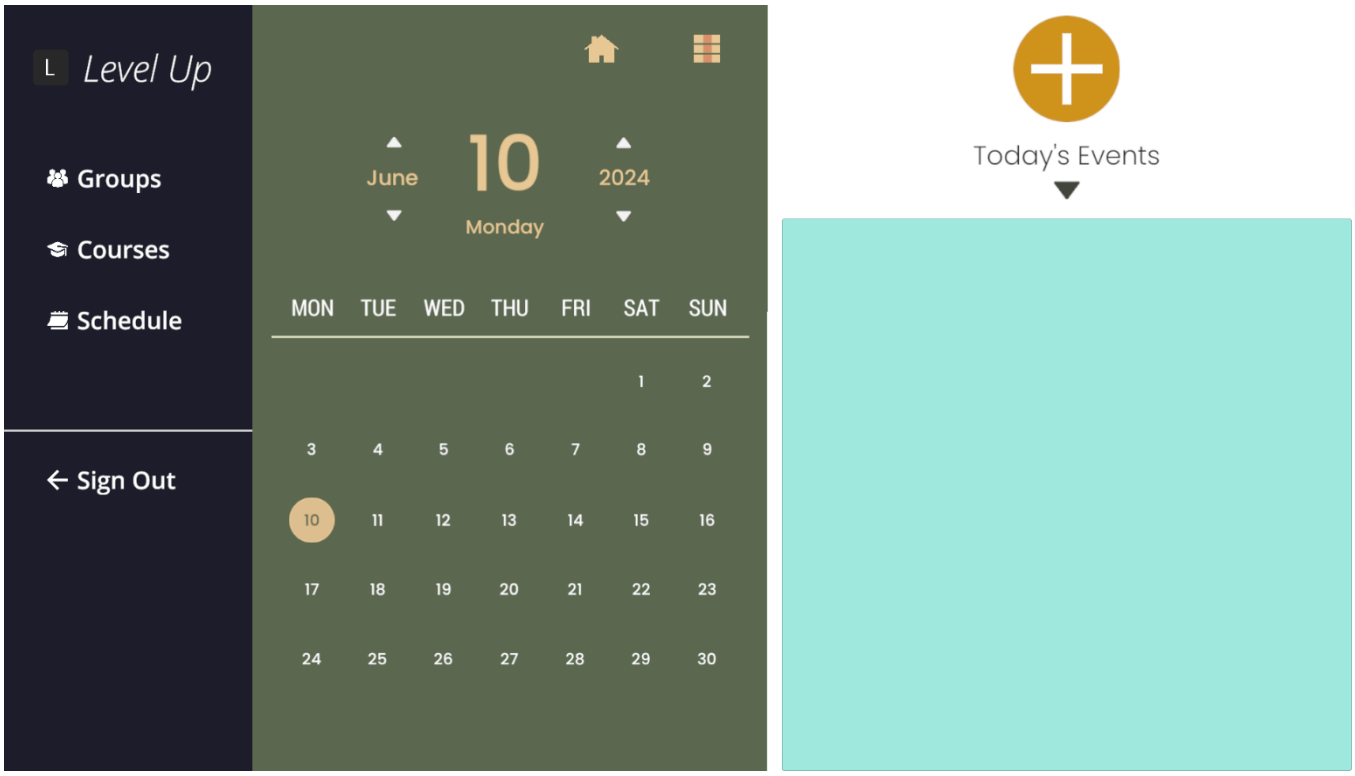

Рис 2.35 Інтерфейс календаря та кнопки його навігації

Після успішного тестування навігації календарем, наступним кроком тестування було здійснення перевірки системи призначення занять. Призначення занять має значно менше проблем із валідацією оскільки замість ручного введення інформації, в даному випадку було отримано візуальні елементи, де замість вписування інформації, користувач вибирає із серед вже існуючих або створених ним варіантів вибору. Такий підхід до призначення занять значно зменшив ризик помилок користувача, та зробив планування ще більш дружелюбним до користувача.

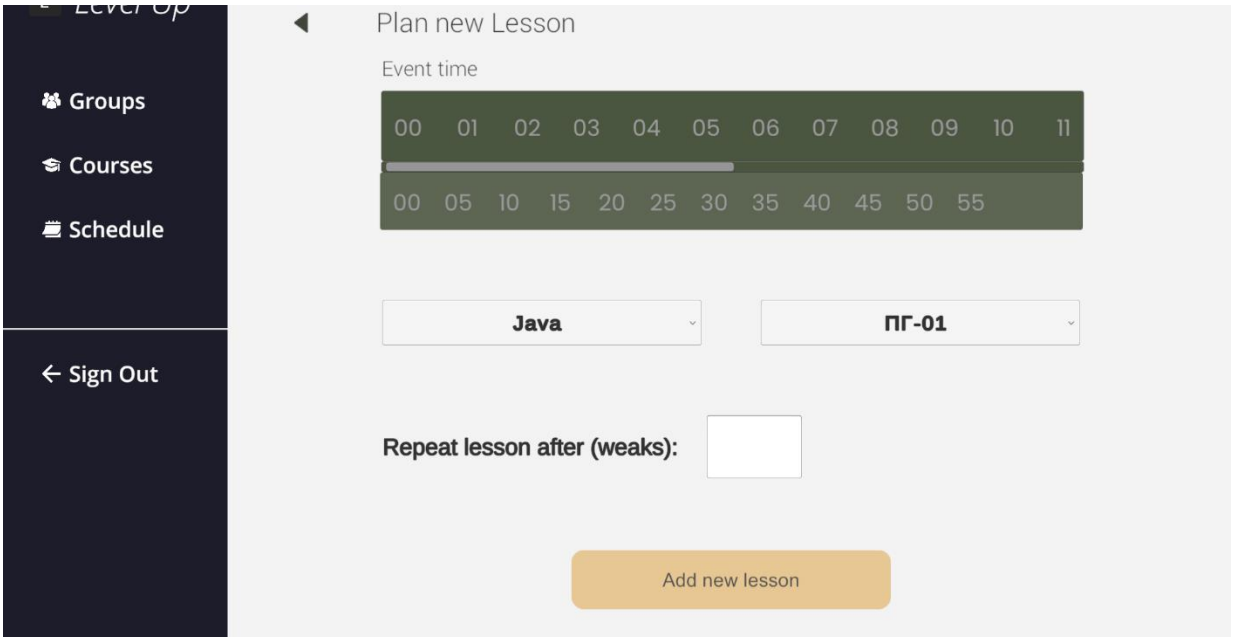

Рисунок 2.36 Панель призначення заняття

Заняття планується на день, який було обрано в календарі. Для тесту було додано заняття, дата заняття - 10.06.2024, час заняття – 10:25, дисципліна – “Python”, група – ПГ-01, повторення заняття – кожен один тиждень.

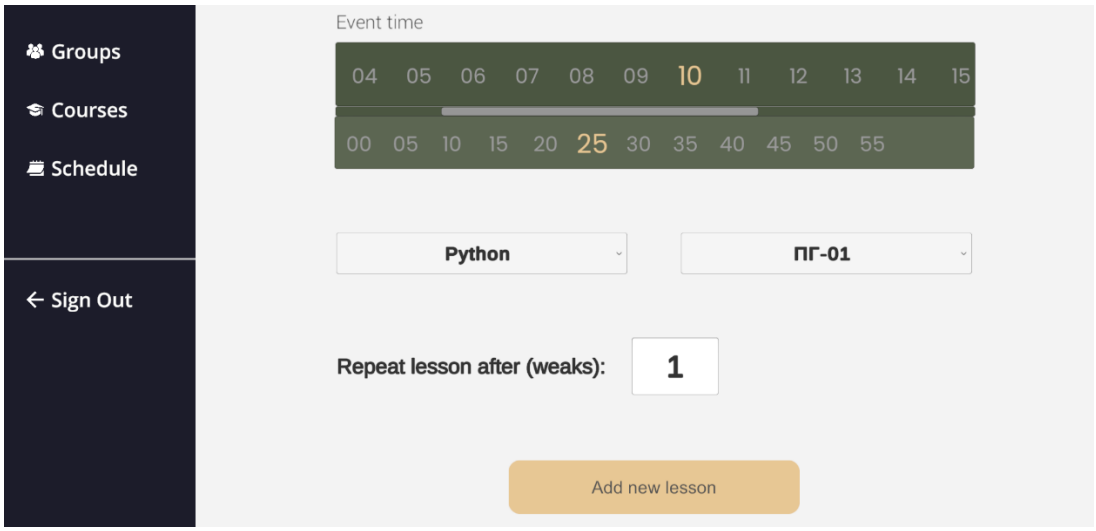

Рисунок 2.37 Обрані параметри заняття

Після додавання заняття інтерфейс був успішно оновлений і починаючи із понеділка – 10.06, кожен наступний понеділок отримав запланований заняття

у своєму списку подій, понеділки до зазначеної дати, та інші дні тижня не мають жодних запланованих занять. Інформація у базі даних була також успішно оновлена, після чого у обраного користувача можна побачити вкладку із заняттями та інформацією про їх планування.

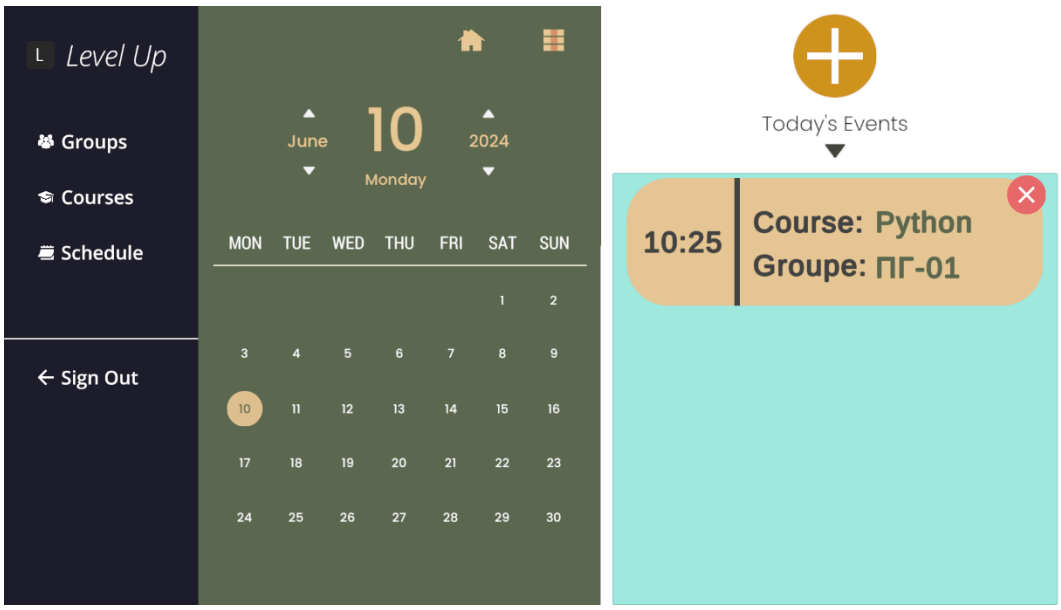

Рисунок 2.38 Результат додавання заняття на обрану дату

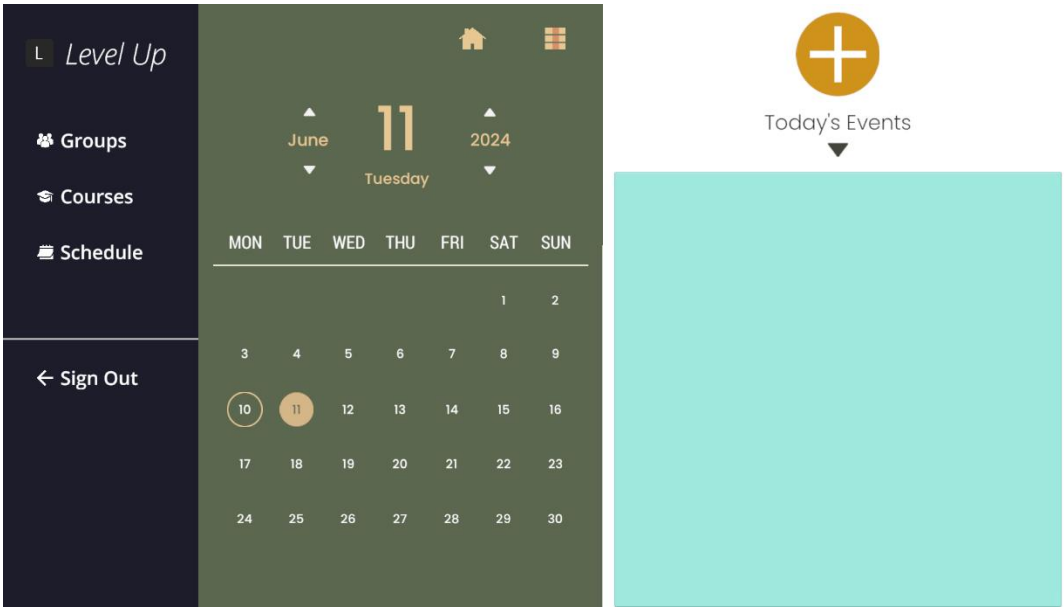

Рисунок 2.39 Результат обирання іншого дня тижня

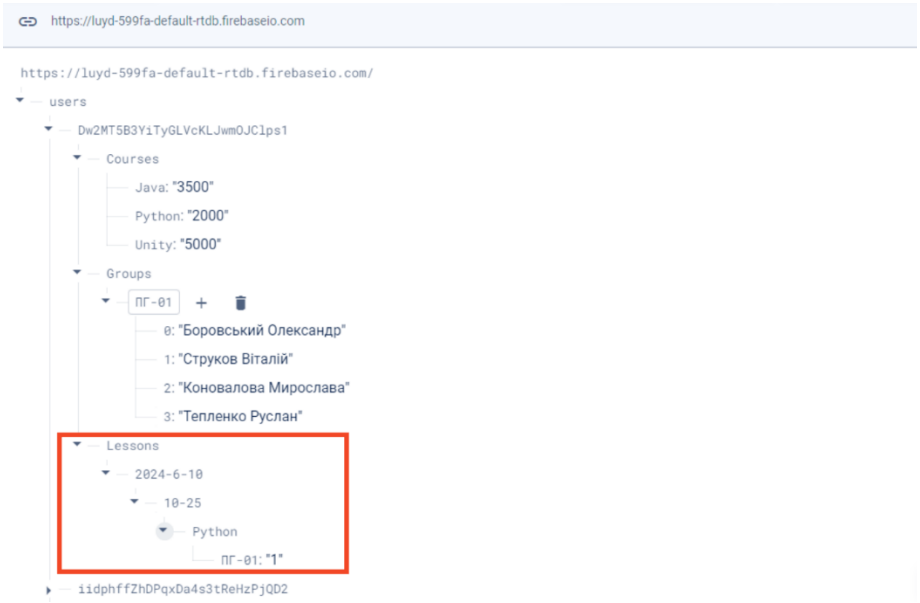

Рисунок 2.40 Результат оновлення інформації в базі даних

Наступним етапом тестування даної системи стало тестування можливості динамічного повторення уроків. Потрібно перевірити можливість керування частотою повторення занять. Для цього було додано ще одне заняття із інтервалом повторення 2 тижні.

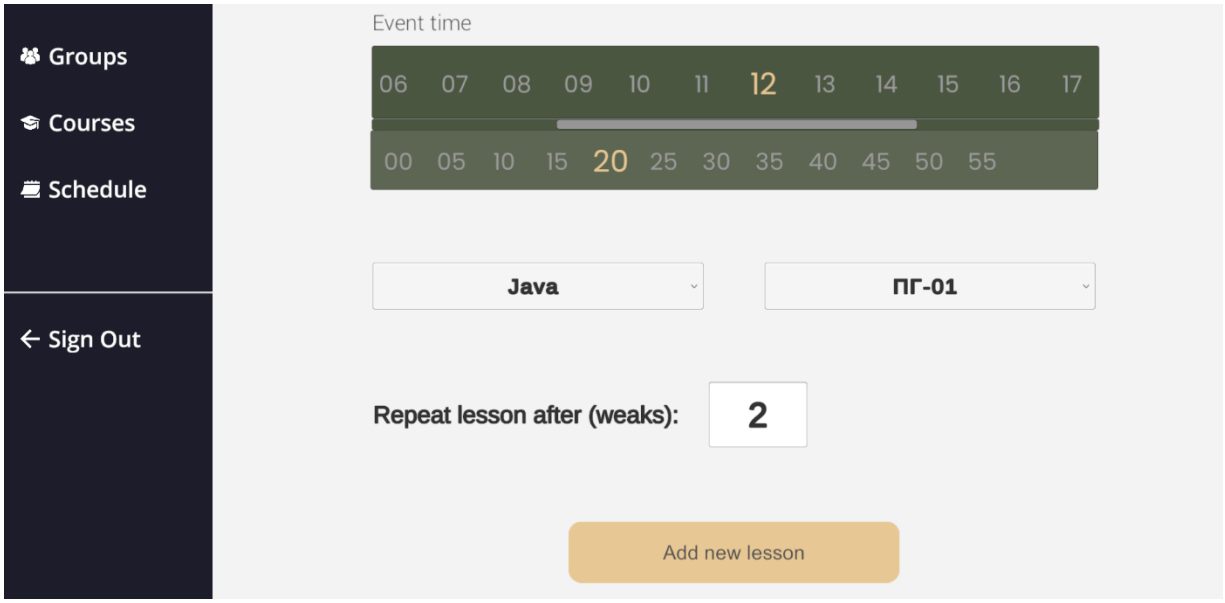

Рисунок 2.41 Параметри ствення нового заняття

Після додавання нового заняття одразу оновилася інформація на ітерфейсі та інформація у базі даних. Після чого у обраний день – п’ятницю можна було спостерігати заняття у списку запланованих подій.

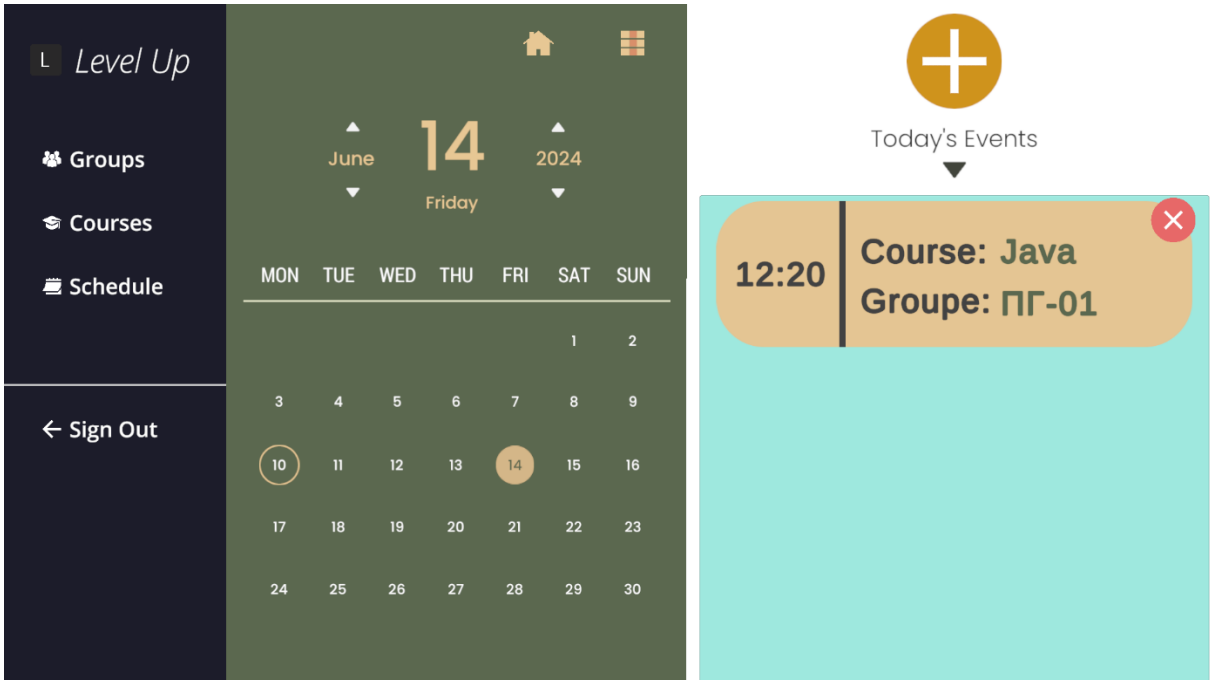

Рисунок 2.42 Результат додавання нового заняття на інтерфейсі

Після додавання заняття було перевірено періодичність повторення. Результат виявився відмінним, починаючи з п’ятниці - 14.06, кожен другий тиждень у п’ятницю можна спостерігати заплановане заняття. Як результат можна зробити висновок, що функціонал динамічного повторення занять працює відмінно.

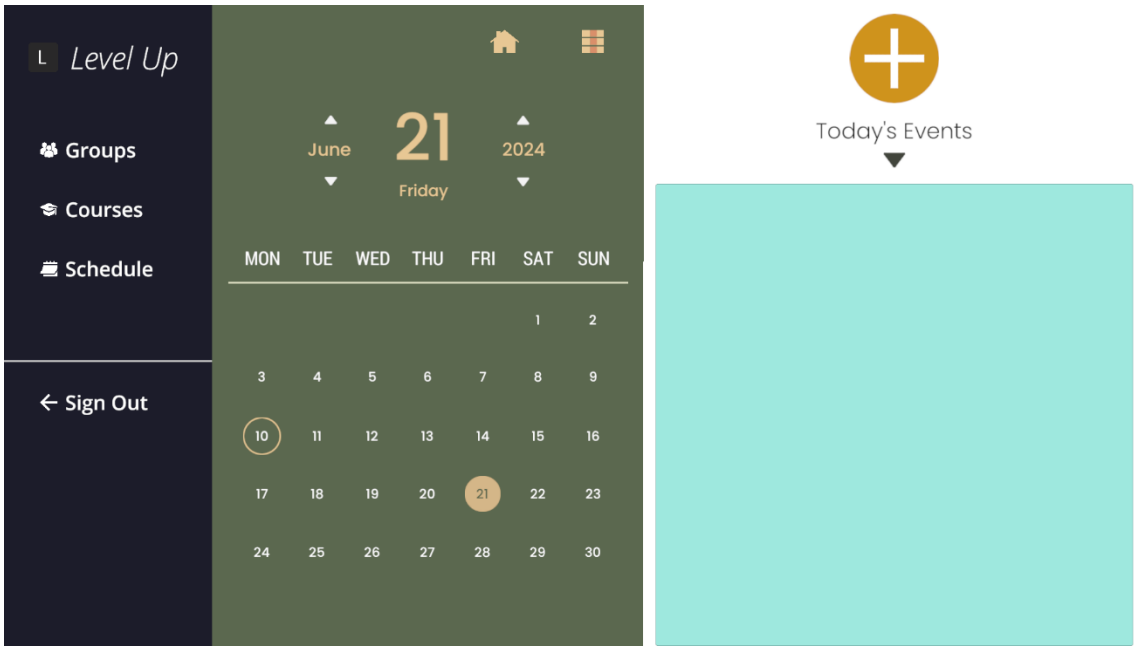

Рисунок 2.43 Перший тиждень після планування заняття

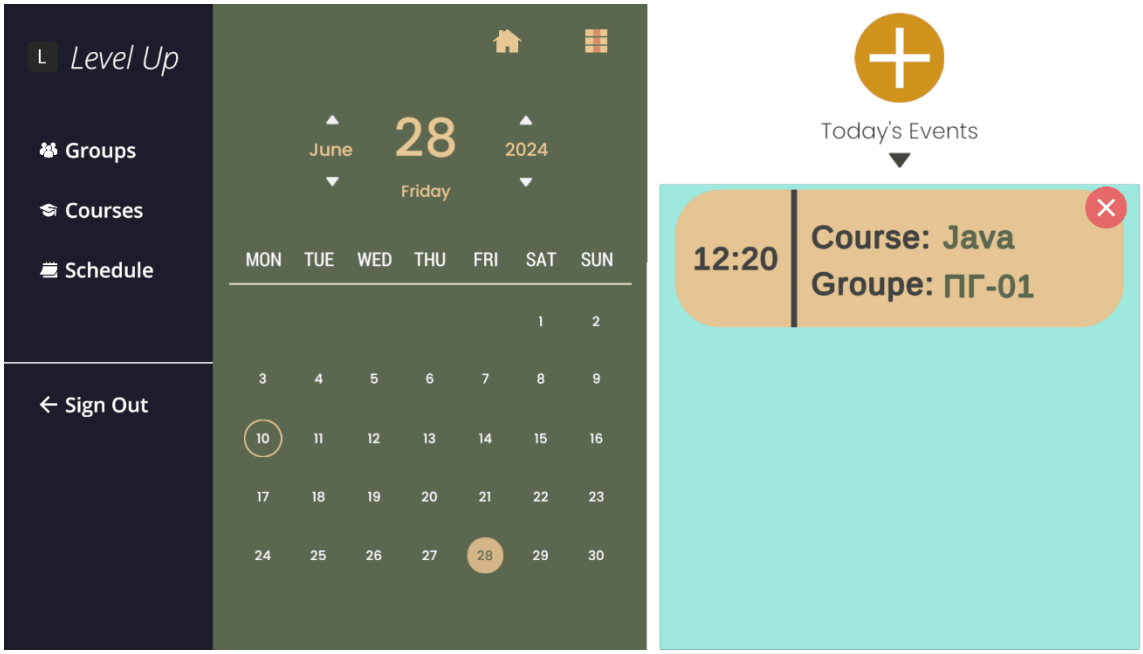

Рисунок 2.44 Другий тиждень після планування заняття

Останньою вимогою до розкладу, була можливість видалення уроків. Необхідний функціонал було простестовано за допомогою видалення першого створеного заняття. Результат роботи є задовільним, відбулося видалення заняття із інтерфейсу та бази даних, у інші дні, на які планувалося заняття воно також видалилося.

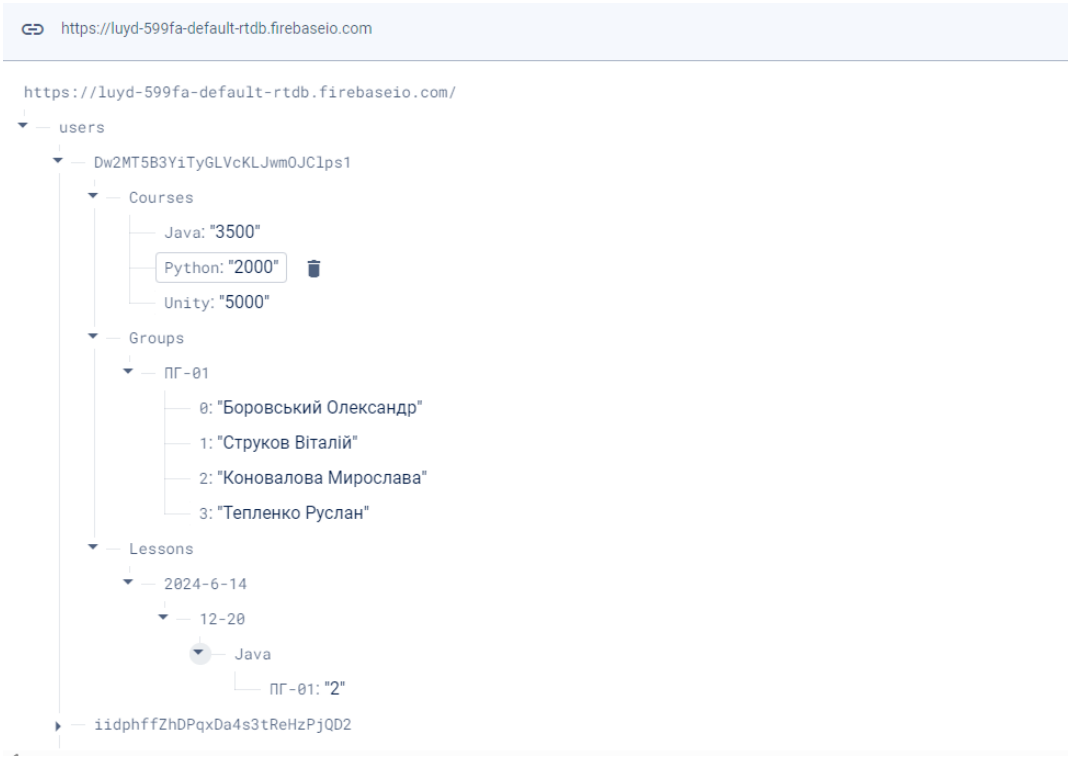

Рисунок 2.45 Результат видалення заняття у базі даних

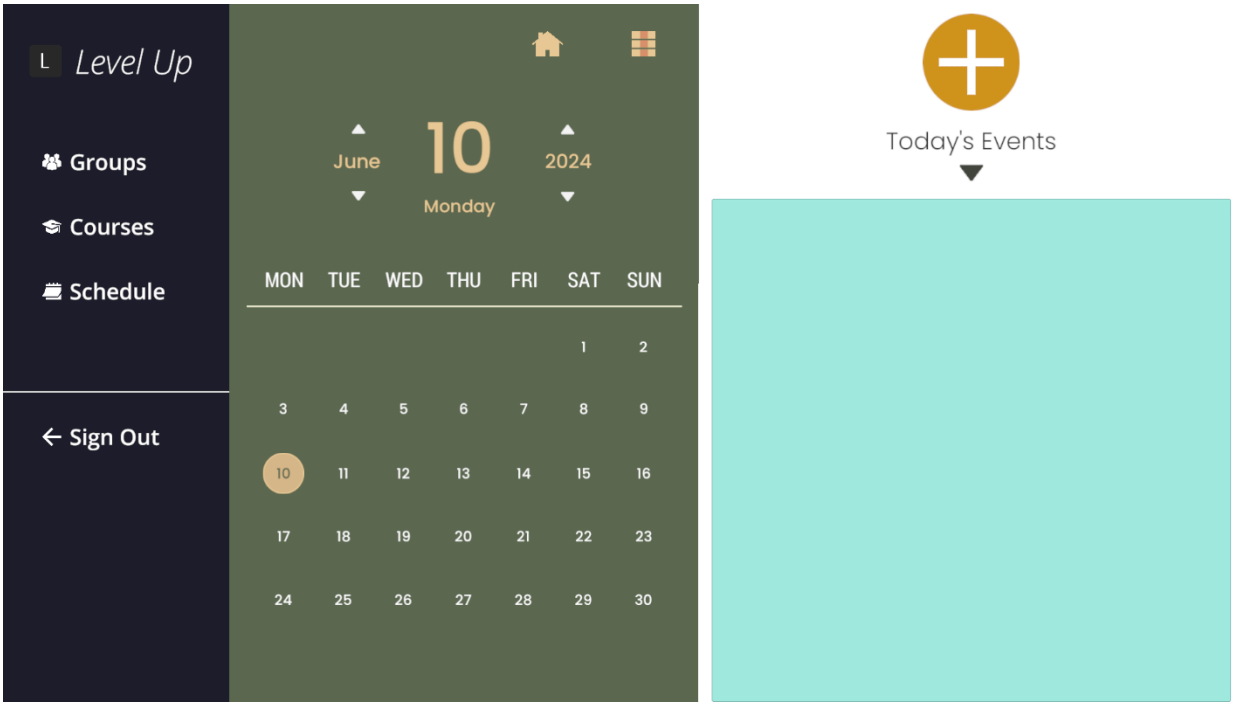

Рисунок 2.46 Результат видалення заняття на інтерфейсі

2.4.4 Тестування продуктивності та безпеки додатку

Окрім, тестування функціоналу також було виконано тестування додатку на відповідність згідно з нефункціональними вимогами такими як продуктивність та безпека.

Кінцевий білд додатку тестувався на трьох девайсах різної потужності та при різному підключенні до інтернету. У всіх випадках час відклику програми при оновлені інтрфейсу не перевищив значення 0.5 секунд, а час авторизації не перевищив 2 секунд, що відповідає поставленим вимогам і цілком задовольняє їх, та навіть перевищує.

Безпека додатку була реалізована завдяки можливостям Firebase, а саме завдяки Firebase realtime databe rules. Firebase realtime databe rules – це правила бази даних, які визначають дозволи зчитування та оновлення інформації в базі даних. Правила прописуються простим задаванням умов в спеціальний текстовий файл, для обраної бази було реалізовано набір правил, які дозволяють додавати та зчитувати інформацію із бази даних лише авторизованим користувачам, а прописується це все лише двома простими командами.

Рисунок 2.47 Правила доступу бази даних

Завдяки правилам бази даних було отримано необхідний рівень безпеки, після задання правил, оновлювати або зчитувати інформацію з бази даних зможуть лише авторизовані користувачі, та лише за своїм унікальним ідентифікаційним номером.

ВИСНОВКИ РОЗДІЛ 2

В другому розділі роботи було проведено вибір актуальних технологій для розроблюваної системи. Обрані технології для реалізації візуальної та логічної частини додатку. Після обрання технологій було створено інтферйс користувача відвідно до вимог, проведено аналіз класів системи та досліджено принцип та особливості їх роботи. Після аналізу класів було проведено тестування розробленого додатку. Під час тестування було виявлено, що додаток працює правильно.

ВИСНОВКИ

Під час виконання дипломної роботи було досліджено технології організації освітніх процесів. Було проаналізовано сучасні платформи для супроводу освітнього процесу. Було виявлено ряд недоліків у вже існуючих LMS додатках. Основною проблемою було виявлено відсутність систем розкладу і переліку всіх завдань, які було б зібрано в одному місці, що в свою чергу негативно впливало до досвід користувачів та змушувало користувачів моніторити кожну дисципліну окремо. Інші види додатків, які могли закрити питання наявності розкладу також мали ряд недоліків такі як завищена абстрактність, що призводить до потреби додаткового навчання та негативно впливає на відсоток автоматизації процесу. Найпростіші із досліджених додатків надають максимум користі за мінімальну кількість зусиль користувача, але мають дуже обмежений функціонал. Що в свою чергу так само негативно вливало на досвід користувача.

Після аналізу ринку було виявлено ряд вище перерахованих проблем, які повністю не міг покрити жодна із розглянутих платформ. Тому було ухвалено рішення розробити такий додаток, який зможе покрити всі перераховані проблемні області, які було виявлено під час аналізу.

Після аналізу було досліджено всі сучасні технології на ринку, які могли б допомогти максимально швидко та якісно реалізувати додаток, який зможе перекрити всі досліджені проблемні зони. Серед широкого спектру було обрано ряд найкращих технологій, які можуть відмінно співпрацювати між собою.

Після обрання технологій, було чітко сформульовано функціональні та нефункціональні вимоги до розроблюваного додатку. Що в свою чергу дозволило почати реалізаційну частину роботи.

Згідно з сформульованими вимогами було розроблено автоматизовану систему у вигляді додатку, для викладачів, який може автоматизувати значну частину організації навчального процесу. Під час розробки було реалізовано всі функціональні вимоги до додатку. А також виконано тестування на захищеність, та

швидкодію згідно з нефункціональними вимогами. Додаток пройшов всі етапи розробки та гарно себе показав під час фінального тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронні журнали. Переваги та недоліки. Режим доступу: https://vseosvita.ua/blogs/elektronni-zhurnay-perevahy-ta-nedoliky-94093.html
2. Система планування й організація навчального процесу у ВНЗ. Режим доступу: https://shorturl.at/YK9Th
3. Zoom. Режим доступу: https://zoom.us/
4. Google meet. Режим доступу: https://meet.google.com/
5. Moodle. Режим доступу: https://moodle.org/?lang=uk
6. Canvas. Режим доступу: https://www.instructure.com/canvas
7. Blackboard. Режим доступу: https://www.blackboard.com/
8. Система управління навчанням. Режим доступу: https://ahaslides.com/uk/blog/learning-management-system/
9. Що таке Google Classroom. Режим доступу: https://osvitoria.media/news/google-classroom-instruktsiya-yak-samostijno-stvoryuvaty-onlajn-kursy/
10. What is Canvas? Режим доступу: https://community.canvaslms.com/t5/Canvas-Basics-Guide/What-is-Canvas/ta-p/45
11. Set Up Your Canvas Course in 30 minutes or Less. Режим доступу: https://community.canvaslms.com/t5/Contingency-Resources/Set-Up-Your-Canvas-Course-in-30-minutes-or-Less/ba-p/258437
12. Schoology: все, що вам потрібно знати. Режим доступу: https://polaridad.es/uk/ventajas-y-desventajas-de-schoology/
13. How do I integrate Schoology? Режим доступу: https://support.wevideo.com/hc/en-us/articles/360001230954-How-do-I-integrate-Schoology
14. Notion. Режим достпу: https://www.notion.so/
15. Огляд Notion: корисні функції. Режим доступу: https://icoola.ua/blog/programa-notion-na-iphone/
16. Розклад КПІ ім. Ігоря Сікорського. Режим доступу: https://schedule.kpi.ua/
17. Системні вимоги. Режим доступу: https://studfile.net/preview/4452595/page:9/

18. Що таке функціональні вимоги: приклади, визначення, повний посібник. Режим доступу: https://visuresolutions.com/uk/blog/functional-requirements/
19. Нефункціональні вимоги: приклади, типи, підходи. Режим доступу: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/
20. Refactoring.Guru. Режим доступу: Refactoring and Design Patterns
21. Як вивчити мову програмування C# та стати .NET розробником. Режим доступу: https://edu.cbsystematics.com/ua/blog/learn-csharp-become-dnet
22. Oracle official. Режим доступу: https://www.oracle.com/ua/database/what-is-database/
23. Що таке firebase? https://avada-media.ua/ua/services/firebase/
24. Автентифікація Firebase. Режим доступу: Firebase Authentication (google.com)
25. Що таке nosql бази даних? Режим доступу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-bazi-dannih-nosql/
26. Firebase realtime database. Режим доступу: Firebase Realtime Database (google.com)
27. Visual studio. Режим доступу: https://visualstudio.microsoft.com/
28. Навіщо потрібна Visual Studio 2022: основні переваги та можливості. Режим доступу: https://shchuka.zapisi.cx.ua/ukraincyam/navishho-potribna-visual-studio-2022-osnovni-perevagi-ta-mozhlivosti.html
29. Unity official page. Режим доступу: https://unity.com/
30. Ігровий двигун Юнті та на що він здатний. Режим доступу: https://itproger.com/ua/news/igrovoy-dvizhok-unity-razbiraemsya-chto-k-chem
31. Топ-5 фреймворків для Front-end розробки. Режим доступу: https://wezom.academy/ua/top-5-frejmvorkov-dlja-front-end-razrabotki/
32. Unity documentation - User interface. Режим доступу: Unity - Manual: User interface (UI) (unity3d.com)
33. Бібліотеки в програмуванні: потужний інструмент для створення чистого коду. Режим доступу: https://foxminded.ua/shcho-take-biblioteka-v-prohramuvanni/

34. Unity documentation - TextMeshPro. Режим доступу: Unity - Manual: TextMeshPro (unity3d.com)
35. FlatCalendar. Режим доступу: https://assetstore.unity.com/packages/tools/gui/flat-calendar-v2-59431
36. Modern UI pack. Режим доступу: https://assetstore.unity.com/packages/tools/gui/modern-ui-pack-201717
37. What are “Render Pipelines” in Unity? Режим доступу: https://myjl-besnard.medium.com/what-are-render-pipelines-in-unity-bb9bcd02cf98
38. Класи в програмуванні. Режим доступу: https://foxminded.ua/klasy-v-prohramuvanni/
39. Використання компонентів. Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/UsingComponents.html
40. Java Helper vs. Utility Classes. Режим доступу: https://www.baeldung.com/java-helper-vs-utility-classes

ДОДАТОК А

Програмний код

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System;

public class FieldInputManager : MonoBehaviour
{

    public static FieldInputManager Instance {  get; private set; }

    [SerializeField] private TMP_InputField emailLoginField;
    [SerializeField] private TMP_InputField passwordLoginField;
    [SerializeField] private TMP_InputField usernameRegisterField;
    [SerializeField] private TMP_InputField emailRegisterField;
    [SerializeField] private TMP_InputField passwordRegisterField;
    [SerializeField] private TMP_InputField passwordRegisterVerifyField;

    private void Awake()
    {
        if (Instance != null)
        {
            Debug.LogError("There are more than FieldInputManager");
        }
        Instance = this;
    }

    public string GetEmailLoginText()
    {
        return emailLoginField.text;
    }

    public string GetPasswordLoginField()
    {
        return passwordLoginField.text;
    }

    public string GetUsernameRegisterText()
    {
        return usernameRegisterField.text;
    }

    public string GetEmailRegisterField()
    {
        return emailRegisterField.text;
    }

    public string GetPasswordRegisterField()
    {
        return passwordRegisterField.text;
    }

    public string GetRegisterVerifyPasswordField()
    {
        return passwordRegisterVerifyField.text;
    }

    public void ClearLoginFeilds()
```

```
    {
        emailLoginField.text = "";
        passwordLoginField.text = "";
    }
    public void ClearRegisterFeilds()
    {
        usernameRegisterField.text = "";
        emailRegisterField.text = "";
        passwordRegisterField.text = "";
        passwordRegisterVerifyField.text = "";
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class WindowManager : MonoBehaviour
{

    public event EventHandler OnGroupWindowOpened;
    public event EventHandler OnCourseWindowOpened;
    public event EventHandler OnLessonsWindowOpened;

    public static WindowManager Instance { get; private set; }

    [SerializeField] private Canvas mainCanvas;

    [Header("Menues")]
    [SerializeField] private GameObject authenticationMenu;
    [SerializeField] private GameObject profileMenu;

    [Header("Authentication")]
    [SerializeField] private GameObject loginWindow;
    [SerializeField] private GameObject registerWindow;
    [SerializeField] private GameObject forgotPasswordWindow;

    [Header("Groups")]
    [SerializeField] private GameObject groupsMainWindow;
    [SerializeField] private GameObject addNewGroupWindow;

    [Header("Courses")]
    [SerializeField] private GameObject coursesMainWindow;

    [Header("Lessons")]
    [SerializeField] private GameObject lessonsMainWindow;
    [SerializeField] private GameObject addNewLessonWindow;
    [SerializeField] private SimpleDialogWindow simpleDialogWindowPrefab;

    private GameObject[] authenticationWindows;
    private GameObject[] groupsWindows;
    private GameObject[] coursesWindows;
    private GameObject[] lessonsWindows;

    private void Awake()
    {
        if (Instance != null)
        {
            Debug.LogError("There are more than one WindowManager");
        }
        Instance = this;
    }

    private void Start()
```

```
    {
        authenticationWindows = new GameObject[] { loginWindow, registerWindow,
forgotPasswordWindow };
        groupsWindows = new GameObject[] { groupsMainWindow, addNewGroupWindow };
        lessonsWindows = new GameObject[] { lessonsMainWindow, addNewLessonWindow };
        coursesWindows = new GameObject[] { coursesMainWindow };
    }

    private void CloseAllWindows(GameObject[] windows)
    {
        foreach (GameObject window in windows)
        {
            window.SetActive(false);
        }
    }

    public void OpenLoginWindow()
    {
        CloseAllWindows(authenticationWindows);
        loginWindow.SetActive(true);
    }

    public void CloseEveryAppWindow()
    {
        CloseAllWindows(authenticationWindows);
        CloseAllWindows(groupsWindows);
        CloseAllWindows(coursesWindows);
        CloseAllWindows(lessonsWindows);
    }

    public void OpenRegisterWindow()
    {
        CloseAllWindows(authenticationWindows);
        registerWindow.SetActive(true);
    }
    public void OpenForgotPasswordWindow()
    {
        CloseAllWindows(authenticationWindows);
        forgotPasswordWindow.SetActive(true);
    }
    public void OpenProfileMenu()
    {
        CloseAuthenticationMenu();
        profileMenu.SetActive(true);
    }

    public void OpenAuthenticationMenu()
    {
        CloseAllWindows(authenticationWindows);
        authenticationMenu.SetActive(true);
        OpenLoginWindow();
    }

    public void CloseAuthenticationMenu()
    {
        CloseAllWindows(authenticationWindows);
        authenticationMenu.SetActive(false);
    }

    public void OpenGroupsWindow()
    {
        CloseEveryAppWindow();
        groupsMainWindow.SetActive(true);
        OnGroupWindowOpened.Invoke(this, EventArgs.Empty);
```

```
    }
    public void OpenAddNewGroupWindow()
    {
        CloseAllWindows(groupsWindows);
        addNewGroupWindow.SetActive(true);
    }

    public void OpenCoursesMainWindow()
    {
        CloseEveryAppWindow();
        coursesMainWindow.SetActive(true);
        OnCourseWindowOpened?.Invoke(this, EventArgs.Empty);
    }

    public void OpenLessonsMainWindow()
    {
        CloseEveryAppWindow();
        lessonsMainWindow.SetActive(true);
        OnLessonsWindowOpened?.Invoke(this, EventArgs.Empty);

    }

    public void OpenAddNewLessonWindow()
    {
        if (ScheduleManager.Instance.HasGroupAndCourse())
        {
            addNewLessonWindow.SetActive(true);
        }
        else
        {
            CreateSimpleDialogWindow("Error", "Nou groups or courses yet");
        }


    }
    public void CloseAddNewLessonWindow()
    {
        addNewLessonWindow.SetActive(false);
    }


    public void CreateSimpleDialogWindow(string windowText, string messageText)
    {
        GameObject spawnedWinow = Instantiate(simpleDialogWindowPrefab.gameObject,
mainCanvas.transform);
        spawnedWinow.GetComponent<SimpleDialogWindow>().SetupWindow(windowText,
messageText);
    }

    public void QuitApplication()
    {
        Application.Quit();
    }

    public void CloseProfileMenu()
    {
        CloseEveryAppWindow();
        profileMenu.SetActive(false);
        OpenAuthenticationMenu();
        OpenLoginWindow();
    }
}

using UnityEngine;
using UnityEngine.UI;
```

```
using Firebase;
using Firebase.Auth;
using Firebase.Extensions;
using System;
using System.Threading.Tasks;
using TMPro;
using System.Collections;
using Firebase.Database;
using System.Collections.Generic;
using UnityEngine.Experimental.AI;

public class FirebaseManager : MonoBehaviour
{
    public event EventHandler OnUselrLogined;
    public event EventHandler<OnUserGeroupsListLoaded_EventArgs> OnUserGroupsListLoaded;
    public event EventHandler<OnGroupDataLoaded_EventArgs> OnGroupDataLoaded;
    public event EventHandler<OnCoursesDataLoaded_EventArgs> OnCoursesDataLoaded;
    public event EventHandler<OnLessonsDataLoaded_EventArgs> OnLessonsDataLoaded;


    public class OnLessonsDataLoaded_EventArgs : EventArgs
    {
        public List<Lesson> userLessonsList;
    }

    public class OnUserGeroupsListLoaded_EventArgs : EventArgs
    {
        public List<StudentGroup> groupsList;
    }
    public class OnGroupDataLoaded_EventArgs : EventArgs
    {
        public StudentGroup studentGroup;
    }
    public class OnCoursesDataLoaded_EventArgs : EventArgs
    {
        public List<string> coursesList;
    }

    private FieldInputManager fieldInputManager;

    public DependencyStatus dependencyStatus;
    public FirebaseAuth auth;
    public FirebaseUser user;
    public DatabaseReference DBreference;

    private void Start()
    {
        fieldInputManager = FieldInputManager.Instance;
    }

    private void Awake()
    {
        //Check that all of the necessary dependencies for Firebase are present on the
system
        FirebaseApp.CheckAndFixDependenciesAsync().ContinueWith(task =>
        {
            dependencyStatus = task.Result;
            if (dependencyStatus == DependencyStatus.Available)
            {
                //If they are avalible Initialize Firebase
                InitializeFirebase();
            }
            else
            {
```

```
                Debug.LogError("Could not resolve all Firebase dependencies: " +
dependencyStatus);
            }
        });
    }

    private void InitializeFirebase()
    {
        Debug.Log("Setting up Firebase Auth");
        //Set the authentication instance object
        auth = FirebaseAuth.DefaultInstance;
        DBreference = FirebaseDatabase.DefaultInstance.RootReference;
    }

    public void LoginButton()
    {
        //Call the login coroutine passing the email and password
        StartCoroutine(Login(fieldInputManager.GetEmailLoginText(),
fieldInputManager.GetPasswordLoginField()));
    }

    //Function for the register button
    public void RegisterButton()
    {
        //Call the register coroutine passing the email, password, and username
        StartCoroutine(Register(fieldInputManager.GetEmailRegisterField(),
                                fieldInputManager.GetPasswordRegisterField(),
                                fieldInputManager.GetUsernameRegisterText()));
    }

    public void SignOutButton()
    {
        auth.SignOut();
        WindowManager.Instance.CloseProfileMenu();
        fieldInputManager.ClearRegisterFeilds();
        fieldInputManager.ClearLoginFeilds();
    }
    private IEnumerator Login(string _email, string _password)
    {
        //Call the Firebase auth signin function passing the email and password
        Task<AuthResult> LoginTask = auth.SignInWithEmailAndPasswordAsync(_email,
_password);
        //Wait until the task completes
        yield return new WaitUntil(predicate: () => LoginTask.IsCompleted);

        if (LoginTask.Exception != null)
        {
            //If there are errors handle them
            Debug.LogWarning(message: $"Failed to register task with
{LoginTask.Exception}");
            FirebaseException firebaseEx = LoginTask.Exception.GetBaseException() as
FirebaseException;
            AuthError errorCode = (AuthError)firebaseEx.ErrorCode;

            string message = "Login Failed!";
            switch (errorCode)
            {
                case AuthError.MissingEmail:
                    message = "Missing Email";
                    break;
                case AuthError.MissingPassword:
                    message = "Missing Password";
                    break;
                case AuthError.WrongPassword:
                    message = "Wrong Password";
```

```
                    break;
                case AuthError.InvalidEmail:
                    message = "Invalid Email";
                    break;
                case AuthError.UserNotFound:
                    message = "Account does not exist";
                    break;
            }
            WindowManager.Instance.CreateSimpleDialogWindow("Error", message);
        }
        else
        {
            //User is now logged in
            //Now get the result
            user = LoginTask.Result.User;
            OnUselrLogined?.Invoke(this, EventArgs.Empty);
            Debug.LogFormat("User signed in successfully: {0} ({1})", user.DisplayName,
user.Email);

            StartCoroutine(LoadUserData());

            float loadingTime = 2;

            yield return new WaitForSeconds(loadingTime);

            WindowManager.Instance.OpenProfileMenu();



            fieldInputManager.ClearRegisterFeilds();
            fieldInputManager.ClearLoginFeilds();

        }
    }

    private IEnumerator Register(string _email, string _password, string _username)
    {
        if (_username == "")
        {
            //If the username field is blank show a warning
            WindowManager.Instance.CreateSimpleDialogWindow("Error", "Missing Username");
        }
        else if (fieldInputManager.GetPasswordRegisterField() !=
fieldInputManager.GetRegisterVerifyPasswordField())
        {
            //If the password does not match show a warning
            WindowManager.Instance.CreateSimpleDialogWindow("Error", "Password Does Not
Match!");
        }
        else
        {
            //Call the Firebase auth signin function passing the email and password
            Task<AuthResult> RegisterTask =
auth.CreateUserWithEmailAndPasswordAsync(_email, _password);
            //Wait until the task completes
            yield return new WaitUntil(predicate: () => RegisterTask.IsCompleted);

            if (RegisterTask.Exception != null)
            {
                //If there are errors handle them
                Debug.LogWarning(message: $"Failed to register task with
{RegisterTask.Exception}");
                FirebaseException firebaseEx = RegisterTask.Exception.GetBaseException() as
FirebaseException;
                AuthError errorCode = (AuthError)firebaseEx.ErrorCode;
```

```
                string message = "Register Failed!";
                switch (errorCode)
                {
                    case AuthError.MissingEmail:
                        message = "Missing Email";
                        break;
                    case AuthError.MissingPassword:
                        message = "Missing Password";
                        break;
                    case AuthError.WeakPassword:
                        message = "Weak Password";
                        break;
                    case AuthError.EmailAlreadyInUse:
                        message = "Email Already In Use";
                        break;
                }
                WindowManager.Instance.CreateSimpleDialogWindow("Error", message);
            }
            else
            {
                //User has now been created
                //Now get the result
                user = RegisterTask.Result.User;

                if (user != null)
                {
                    //Create a user profile and set the username
                    UserProfile profile = new UserProfile { DisplayName = _username };

                    //Call the Firebase auth update user profile function passing the
profile with the username
                    Task ProfileTask = user.UpdateUserProfileAsync(profile);
                    //Wait until the task completes
                    yield return new WaitUntil(predicate: () => ProfileTask.IsCompleted);

                    if (ProfileTask.Exception != null)
                    {
                        //If there are errors handle them
                        Debug.LogWarning(message: $"Failed to register task with
{ProfileTask.Exception}");
                        FirebaseException firebaseEx =
ProfileTask.Exception.GetBaseException() as FirebaseException;
                        AuthError errorCode = (AuthError)firebaseEx.ErrorCode;
                        WindowManager.Instance.CreateSimpleDialogWindow("Error", "Username
Set Failed!");
                    }
                    else
                    {
                        //Username is now set
                        //Now return to login screen

                        WindowManager.Instance.OpenLoginWindow();


                        fieldInputManager.ClearRegisterFeilds();
                        fieldInputManager.ClearLoginFeilds();
                    }
                }
            }
        }

    private IEnumerator UpdateUsernameAuth(string _username)
    {
```

```
        //Create a user profile and set the username
        UserProfile profile = new UserProfile { DisplayName = _username };

        //Call the Firebase auth update user profile function passing the profile with the
username
        Task ProfileTask = user.UpdateUserProfileAsync(profile);
        //Wait until the task completes
        yield return new WaitUntil(predicate: () => ProfileTask.IsCompleted);

        if (ProfileTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with
{ProfileTask.Exception}");
        }
        else
        {
            //Auth username is now updated
        }
    }

    private IEnumerator UpdateUsernameDatabase(string _username)
    {
        //Set the currently logged in user username in the database
        Task DBTask =
DBreference.Child("users").Child(user.UserId).Child("username").SetValueAsync(_username);

        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else
        {
            //Database username is now updated
        }
    }



    private IEnumerator LoadUserData()
    {
        //Get the currently logged in user data
        Task<DataSnapshot> DBTask =
DBreference.Child("users").Child(user.UserId).GetValueAsync();

        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else if (DBTask.Result.Value == null)
        {
            //No data exists yet

        }
        else
        {
            //Data has been retrieved
            DataSnapshot snapshot = DBTask.Result;

        }
    }
```

```
    private void AddNewGroupDB(string groupName, List<Student> groupStudentsList)
    {
        for (int i = 0; i < groupStudentsList.Count; i++)
        {
            StartCoroutine(AddNewStudentToGroupDB(groupName, i,
groupStudentsList[i].Name));
        }
    }

    private IEnumerator AddNewStudentToGroupDB(string groupName, int studentIndex, string
studentName)
    {

        Task DBTask =
DBreference.Child("users").Child(user.UserId).Child("Groups").Child(groupName).Child(studen
tIndex.ToString()).SetValueAsync(studentName);

        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else
        {
            //StudentAdded;
        }
    }

    public void AddNewGroup(string groupName, List<Student> groupStudentsList)
    {
        AddNewGroupDB(groupName, groupStudentsList);

    }



    private IEnumerator LoadUserGroupsList()
    {
        //Get the currently logged in user data
        Task<DataSnapshot> DBTask =
DBreference.Child("users").Child(user.UserId).Child("Groups").GetValueAsync();
        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else if (DBTask.Result.Value == null)
        {
            //No data exists yet
        }
        else
        {
            //Data has been retrieved
            DataSnapshot snapshot = DBTask.Result;

            List<StudentGroup> studentGroupsList = new List<StudentGroup>();

            foreach (var snapshotChild in snapshot.Children)
            {
                studentGroupsList.Add(new StudentGroup(snapshotChild.Key));
            }
```

```
            OnUserGroupsListLoaded?.Invoke(this, new OnUserGeroupsListLoaded_EventArgs {
groupsList =  studentGroupsList});
        }
    }

    public IEnumerator LoadUserGroupStudentsList(string groupName)
    {
        Task<DataSnapshot> DBTask =
DBreference.Child("users").Child(user.UserId).Child("Groups").Child(groupName).GetValueAsyn
c();
        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else if (DBTask.Result.Value == null)
        {
            //No data
        }
        else
        {

            DataSnapshot snapshot = DBTask.Result;
            List<Student> studentsList = new List<Student>();
            foreach (var snapshotChild in snapshot.Children)
            {
                studentsList.Add(new Student(snapshotChild.Value.ToString(), "0"));
            }

            OnGroupDataLoaded.Invoke(this, new OnGroupDataLoaded_EventArgs
            {
                studentGroup = new StudentGroup(groupName, studentsList)
            });
        }


    }

    public void DeleteGroup(string groupName)
    {

DBreference.Child("users").Child(user.UserId).Child("Groups").Child(groupName).RemoveValueA
sync();
    }

    public void DeleteLesson(string lessonDate, string lessonTime)
    {

DBreference.Child("users").Child(user.UserId).Child("Lessons").Child(lessonDate).Child(less
onTime).RemoveValueAsync();
    }

    public void StartLoadingUserGroupsList()
    {
        StartCoroutine(LoadUserGroupsList());
    }

    public void StartLoadGropData(string groupName)
    {
        StartCoroutine(LoadUserGroupStudentsList(groupName));
    }

    private IEnumerator AddDBCourse(string courseName, int coursePrice)
    {
```

```
        Task DBTask =
DBreference.Child("users").Child(user.UserId).Child("Courses").Child(courseName).SetValueAs
ync(coursePrice.ToString());

        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else
        {
            //Course Added;

        }
    }

    private IEnumerator AddDBLesson(string lessonDate, string lessonTime, string
lessonCourse, string lessonGroup, int repeatAfterWeekCount)
    {

        Task DBTask =
DBreference.Child("users").Child(user.UserId).Child("Lessons").Child(lessonDate).

Child(lessonTime).Child(lessonCourse).Child(lessonGroup).SetValueAsync(repeatAfterWeekCount
.ToString());

        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else
        {
            //LessonAdded;
        }
    }

    public void AddNewLesson(string lessonDate, string lessonTime, string lessonCourse,
string lessonGroup, int repeatAfterWeekCount)
    {
        StartCoroutine(AddDBLesson(lessonDate, lessonTime, lessonCourse, lessonGroup,
repeatAfterWeekCount));
    }

    public void AddNewCourse(string courseName, int coursePrice)
    {
        StartCoroutine(AddDBCourse(courseName, coursePrice));

    }



    public void DeleteCourse(string courseName)
    {

DBreference.Child("users").Child(user.UserId).Child("Courses").Child(courseName).RemoveValu
eAsync();
    }

    private IEnumerator LoadUserCoursesList()
    {
        //Get the currently logged in user data
```

```
        Task<DataSnapshot> DBTask =
DBreference.Child("users").Child(user.UserId).Child("Courses").GetValueAsync();
        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else if (DBTask.Result.Value == null)
        {
            //No data exists yet
        }
        else
        {
            //Data has been retrieved
            DataSnapshot snapshot = DBTask.Result;

            List<string> coursesList = new List<string>();

            foreach (var snapshotChild in snapshot.Children)
            {
                coursesList.Add(snapshotChild.Key);
            }

            OnCoursesDataLoaded?.Invoke(this, new OnCoursesDataLoaded_EventArgs {
coursesList = coursesList });
        }
    }

    private IEnumerator LoadLessonsList()
    {
        //Get the currently logged in user data
        Task<DataSnapshot> DBTask =
DBreference.Child("users").Child(user.UserId).Child("Lessons").GetValueAsync();
        yield return new WaitUntil(predicate: () => DBTask.IsCompleted);

        if (DBTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {DBTask.Exception}");
        }
        else if (DBTask.Result.Value == null)
        {
            //No data exists yet
        }
        else
        {
            //Data has been retrieved
            DataSnapshot snapshot = DBTask.Result;

            List<Lesson> lessonsList = new List<Lesson>();

            foreach (var firstChild in snapshot.Children)
            {
                string lessonDate = firstChild.Key;

                foreach (var secondChild in firstChild.Children)
                {
                    string lessonTime = secondChild.Key;

                    foreach (var thirdChild in secondChild.Children)
                    {
                        string lessonCourse = thirdChild.Key;

                        foreach (var fourthChild in thirdChild.Children)
                        {
```

```
                            string lessonGroup = fourthChild.Key;
                            string repeatWeekCount = fourthChild.Value.ToString();

                            Lesson newLesson = new Lesson(lessonDate, lessonTime,
lessonCourse, lessonGroup, repeatWeekCount); //TODO зробити вектор даних
                            lessonsList.Add(newLesson);
                        }
                    }
                }

            }

            OnLessonsDataLoaded?.Invoke(this, new OnLessonsDataLoaded_EventArgs {
userLessonsList = lessonsList });
        }
    }

    public void StartLoadingLessonsList()
    {
        StartCoroutine(LoadLessonsList());
    }


    public void StartLoadingCoursesList()
    {
        StartCoroutine(LoadUserCoursesList());
    }



}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class CourseManager : MonoBehaviour
{
    [SerializeField] private FirebaseManager firebaseManager;
    [SerializeField] private WindowManager windowManager;

    [SerializeField] private GameObject courseSingleVisualPrefab;
    [SerializeField] private Transform coursesGrid;

    [SerializeField] private TMP_InputField courseNameInputField;
    [SerializeField] private TMP_InputField coursePriceInputField;

    private List<string> courseNamesList = new List<string>();




    private void Start()
    {
        windowManager.OnCourseWindowOpened += WindowManager_OnCourseWindowOpened;
        firebaseManager.OnCoursesDataLoaded += FirebaseManager_OnCoursesDataLoaded;
    }


    private void WindowManager_OnCourseWindowOpened(object sender, System.EventArgs e)
    {
        Cleaner.ClearContainer(coursesGrid);
        LoadUserGroups();
    }
```

```
    private void FirebaseManager_OnCoursesDataLoaded(object sender,
FirebaseManager.OnCoursesDataLoaded_EventArgs e)
    {
        courseNamesList = e.coursesList;
        CreateAllCoursesVisuals();
    }


    public void AddNewCourse()
    {
        string courseName = courseNameInputField.text;
        string coursePrice = coursePriceInputField.text;
        if (!InputFieldsIsEmpty())
        {
            firebaseManager.AddNewCourse(courseName, int.Parse(coursePrice));
            courseNamesList.Add(courseName);
            SpawnCourseSingleVisual(courseName);
            ClearInputFields();
        }
    }

    private void SpawnCourseSingleVisual(string courseName)
    {
        GameObject spawnedCourseItem = Instantiate(courseSingleVisualPrefab, coursesGrid);
        SingleGridItemVisual singleGridCourseVisual =
spawnedCourseItem.GetComponent<SingleGridItemVisual>();
        singleGridCourseVisual.SetName(courseName);
        singleGridCourseVisual.GetDeleteButton().onClick.AddListener(() =>
DeleteCourse(courseName, singleGridCourseVisual));
    }

    private void DeleteCourse(string courseName, SingleGridItemVisual visual)
    {
        firebaseManager.DeleteCourse(courseName);
        courseNamesList.Remove(courseName);
        Destroy(visual.gameObject);
    }

    private void LoadUserGroups()
    {
        firebaseManager.StartLoadingCoursesList();
    }

    private void CreateAllCoursesVisuals()
    {
        Cleaner.ClearContainer(coursesGrid);
        foreach (string name in courseNamesList)
        {
            SpawnCourseSingleVisual(name);
        }
    }

    private void ClearInputFields()
    {
        courseNameInputField.text = "";
        coursePriceInputField.text = "";
    }

    private bool InputFieldsIsEmpty()
    {
        return string.IsNullOrEmpty(courseNameInputField.text) ||
string.IsNullOrEmpty(courseNameInputField.text);
    }
```

```
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;


public class GroupsManager : MonoBehaviour
{
    public static GroupsManager Instance { get; private set; }

    [SerializeField] private FirebaseManager firebaseManager;

    [SerializeField] private Transform groupGrid;
    [SerializeField] private SingleGroupeVisual singleGroupVisualPrefab;

    [SerializeField] private Transform studentsGrid;
    [SerializeField] private SingleGridItemVisual studentSingleVisualPrefab;

    [Header("Inputs")]
    [SerializeField] private TMP_InputField groupNameInputField;
    [SerializeField] private TMP_InputField studentNameInputField;
    [SerializeField] private TMP_InputField studentAgeInputField;

    private List<SingleGroupeVisual> currentUserGroupsVisualList = new
List<SingleGroupeVisual>();
    private List<Student> newGroupStudentsList = new List<Student>();

    private List<StudentGroup> curentUserGroupsList;

    private WindowManager windowManager;

    private void Awake()
    {
        windowManager = WindowManager.Instance;
        curentUserGroupsList = new List<StudentGroup>();

        if (Instance != null)
        {
            Debug.LogError("There are more than one GroupsManager");
        }
        Instance = this;
    }

    private void Start()
    {
        windowManager.OnGroupWindowOpened += WindowManager_OnGroupWindowOpened;
        firebaseManager.OnUserGroupsListLoaded += FirebaseManager_OnUserGroupsListLoaded;
        firebaseManager.OnGroupDataLoaded += FirebaseManager_OnGroupDataLoaded;
    }

    private void FirebaseManager_OnGroupDataLoaded(object sender,
FirebaseManager.OnGroupDataLoaded_EventArgs e)
    {
        LoadGroupEditVisuals(e.studentGroup);
        groupNameInputField.text = e.studentGroup.Name;
        DeleteGroup(e.studentGroup);
    }

    private void WindowManager_OnGroupWindowOpened(object sender, System.EventArgs e)
    {
        Cleaner.ClearContainer(groupGrid);
```

```
        LoadStudentGroupsList();
    }

    private void FirebaseManager_OnUserGroupsListLoaded(object sender,
FirebaseManager.OnUserGeroupsListLoaded_EventArgs e)
    {
        curentUserGroupsList = e.groupsList;
        LoadGroupsVisuals();
    }


    private void LoadStudentGroupsList()
    {

        firebaseManager.StartLoadingUserGroupsList();
    }

    private void LoadGroupData(string groupName)
    {
        firebaseManager.StartLoadGropData(groupName);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.LeftControl))
        {
            foreach (StudentGroup item in curentUserGroupsList)
            {
                print(item.Name);
            }
        }
    }

    public void AddNewGroup()
    {
        string newGroupName = groupNameInputField.text;
        if (!GroupInputFieldsIsEmpty())
        {
            if (newGroupStudentsList.Count > 0)
            {
                StudentGroup newStudentGroup = new StudentGroup(newGroupName,
newGroupStudentsList);

                SpawnGroupVisual(newStudentGroup);
                // Add to db
                firebaseManager.AddNewGroup(newGroupName, newGroupStudentsList);
                curentUserGroupsList.Add(new StudentGroup(newGroupName,
newGroupStudentsList));
                newGroupStudentsList.Clear();
                Cleaner.ClearContainer(studentsGrid);
                WindowManager.Instance.OpenGroupsWindow();
                Cleaner.ClearInputField(groupNameInputField);
            }
            else
            {
                //TODO
                //Student list is Empty
                print("St 0");
                WindowManager.Instance.CreateSimpleDialogWindow("Error", "No students");
            }
        }
        else
        {
            //TODO
            //Some fields are empty
```

```
            print("Group name is empty");
            WindowManager.Instance.CreateSimpleDialogWindow("Error", "Group name is
empty");
        }
    }

    public void SpawnGroupVisual(StudentGroup studentGroup)
    {

        GameObject spawnedStudentGroupObject =
Instantiate(singleGroupVisualPrefab.gameObject, groupGrid);
        SingleGroupeVisual spawnedStudentGroupVisual =
spawnedStudentGroupObject.GetComponent<SingleGroupeVisual>();
        currentUserGroupsVisualList.Add(spawnedStudentGroupVisual);
        spawnedStudentGroupVisual.SetGroupNameText(studentGroup.Name);
        spawnedStudentGroupVisual.GetDeleteGroupButton().onClick.AddListener(() =>
DeleteGroup(studentGroup));
        spawnedStudentGroupVisual.GetEditGroupButton().onClick.AddListener(() =>
EditGroup(studentGroup.Name));
    }

    public void AddNewStudent()
    {
        string newStudentName = studentNameInputField.text;
        string newStudentAge = studentAgeInputField.text;

        if (!StudentInputFieldsIsEmpty())
        {

            SpawnStudentVisual(newStudentName);
        }
        else
        {
            print("Student name is empty");
        }
    }

    public void SpawnStudentVisual(string studentName)
    {
        Student newStudent = new Student(studentName, "0");
        GameObject spawnedStudentObject = Instantiate(studentSingleVisualPrefab.gameObject,
studentsGrid);
        SingleGridItemVisual spawnedStudentVisual =
spawnedStudentObject.GetComponent<SingleGridItemVisual>();
        newGroupStudentsList.Add(newStudent);
        spawnedStudentVisual.SetName(studentName);
        spawnedStudentVisual.GetDeleteButton().onClick.AddListener(() =>
DeleteStudent(newStudent, spawnedStudentVisual));
        ClearStudentsInputsFields();
    }

    public void EditGroup(string groupName)
    {
        LoadGroupData(groupName);
        windowManager.OpenAddNewGroupWindow();
    }

    public void DeleteGroup(StudentGroup studentGroup)
    {
        if (TryDeleteGroup(studentGroup.Name))
        {
            foreach (SingleGroupeVisual visual in currentUserGroupsVisualList)
            {
                if (visual.GetGroupNameText() == studentGroup.Name)
                {
```

```
                Destroy(visual.gameObject);

            }
        }
    }
}

public void DeleteStudent(Student student, SingleGridItemVisual studentSingleVisual)
{
    newGroupStudentsList.Remove(student);
    Destroy(studentSingleVisual.gameObject);

}


private void ClearStudentsInputsFields()
{
    studentNameInputField.text = "";
    studentAgeInputField.text = "";
}

private bool StudentInputFieldsIsEmpty()
{
    return string.IsNullOrEmpty(studentNameInputField.text) &&
string.IsNullOrEmpty(studentAgeInputField.text);
}

private bool GroupInputFieldsIsEmpty()
{
    return string.IsNullOrEmpty(groupNameInputField.text);
}

private void ClearContainer(Transform transforms)
{
    foreach (Transform item in transforms)
    {
        Destroy(item.gameObject);
    }
}



public void LoadGroupsVisuals()
{
    currentUserGroupsVisualList.Clear();
    ClearContainer(groupGrid.transform);
    foreach (StudentGroup studentGroup in curentUserGroupsList)
    {
        SpawnGroupVisual(studentGroup);
    }
}

public void LoadGroupEditVisuals(StudentGroup studentGroup)
{

    ClearContainer(studentsGrid.transform);

    foreach (Student student in studentGroup.groupStudentList)
    {

        SpawnStudentVisual(student.Name);
    }
}
public List<StudentGroup> GetUserGroupsList()
{
```

```
        return curentUserGroupsList;
    }

    public bool TryDeleteGroup(string groupName)
    {
        StudentGroup group = GetGroupByName(groupName);
        if (group != null)
        {
            curentUserGroupsList.Remove(group);
            firebaseManager.DeleteGroup(groupName);
            return true;
        }
        return false;
    }

    public StudentGroup GetGroupByName(string groupName)
    {
        foreach (StudentGroup group in curentUserGroupsList)
        {
            if (group.Name == groupName)
            {
                return group;
            }
        }

        return null;
    }


}

public class Student
{
    public string Name { get; private set; }
    public string Age { get; private set; }


    public Student(string name, string age)
    {
        Name = name;
        Age = age;
    }
}

public class StudentGroup
{

    public string Name { get; private set; }
    public List<Student> groupStudentList { get; private set; }

    public StudentGroup(string name, List<Student> groupStudentList)
    {
        Name = name;
        this.groupStudentList = groupStudentList;
    }

    public StudentGroup(string name)
    {
        Name = name;
    }

    public bool TryRemoveStudent(string studentName)
    {
        foreach(Student student in groupStudentList)
        {
```

```
            if (student.Name == studentName)
            {
                groupStudentList.Remove(student);
                return true;
            }
        }
        return false;
    }

}


using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class ScheduleManager : MonoBehaviour
{
    public static ScheduleManager Instance { get; private set; }

    [SerializeField] private FlatCalendar2 flatCalendar2;
    [SerializeField] private FirebaseManager firebaseManager;

    [SerializeField] private TMP_Dropdown groupDropdown;
    [SerializeField] private TMP_Dropdown coursesDropdown;
    [SerializeField] private TMP_InputField repeatAfterWeek_InputField;

    [SerializeField] private Transform lessonsGrid;
    [SerializeField] private SingleLessonVisual lessonVisualPrefab;

    private const int daysInWeek = 7;
    private const int minutesSymbolCount = 2;

    private string newLessonDate;
    private string newLessonTime;
    private string newLessonCourseName;
    private string newLessonGroupeName;
    private string newRepeatAfterWeekValue;

    private WindowManager windowManager;

    private List<StudentGroup> groupsList = new List<StudentGroup>();
    private List<string> coursesList = new List<string>();

    private void Awake()
    {
        if (Instance != null)
        {
            Debug.LogError("There are more than one ScheduleManager");
        }
        Instance = this;
    }

    private void Start()
    {
        windowManager = WindowManager.Instance;
        windowManager.OnLessonsWindowOpened += WindowManager_OnLessonsWindowOpened;
        firebaseManager.OnCoursesDataLoaded += FirebaseManager_OnCoursesDataLoaded;
        firebaseManager.OnUserGroupsListLoaded += FirebaseManager_OnUserGroupsListLoaded;
        firebaseManager.OnLessonsDataLoaded += FirebaseManager_OnLessonsDataLoaded;
        flatCalendar2.OnSelectedDayChanged += FlatCalendar2_OnSelectedDayChanged;
    }
```

```
    private void FlatCalendar2_OnSelectedDayChanged(object sender, EventArgs e)
    {
        Cleaner.ClearContainer(lessonsGrid);
        firebaseManager.StartLoadingLessonsList();
    }
    private void FirebaseManager_OnLessonsDataLoaded(object sender,
FirebaseManager.OnLessonsDataLoaded_EventArgs e)
    {
        UpdateDayLessonsVisual(e.userLessonsList);
    }

    private void UpdateDayLessonsVisual(List<Lesson> lessonsList)
    {
        foreach (Lesson lesson in lessonsList)
        {
            DateTime selectedDate =
TimeManager.GetDayByString(flatCalendar2.GetSelectedDate());
            DateTime lessonDate = TimeManager.GetDayByString(lesson.date);
            if (selectedDate >= lessonDate)
            {
                DateTime firstLessonDay = TimeManager.GetDayByString(lesson.date);
                DateTime selectedDay =
TimeManager.GetDayByString(flatCalendar2.GetSelectedDate());
                int daysAfterFirstLesson = (selectedDay - firstLessonDay).Days;

                bool todayIsLessonDay = (daysAfterFirstLesson %
(int.Parse(lesson.repeatWeekCount) * daysInWeek)) == 0;

                if (todayIsLessonDay)
                {
                    SpawnLessonVisual(lesson);
                }
            }
        }
    }

    private void FirebaseManager_OnUserGroupsListLoaded(object sender,
FirebaseManager.OnUserGeroupsListLoaded_EventArgs e)
    {
        groupsList = e.groupsList;
        UpdateGroupDropdown();
    }

    private void FirebaseManager_OnCoursesDataLoaded(object sender,
FirebaseManager.OnCoursesDataLoaded_EventArgs e)
    {
        coursesList = e.coursesList;
        UpdateCoursesDropdown();
    }

    private void WindowManager_OnLessonsWindowOpened(object sender, System.EventArgs e)
    {
        groupsList = new List<StudentGroup>();
        coursesList = new List<string>();
        Cleaner.ClearContainer(lessonsGrid);
        firebaseManager.StartLoadingUserGroupsList();
        firebaseManager.StartLoadingCoursesList();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Y))
        {
```

```
            DateTime date1 = DateTime.Today;
            DateTime date2 = TimeManager.GetDayByString("2024-05-26");
            print((date2 - date1).Days);
        }
    }

    private void InitializeNewGroupData()
    {
        newLessonDate = flatCalendar2.GetSelectedDate();
        newLessonTime = flatCalendar2.GetSelectedHours() + "-" +
flatCalendar2.GetSelectedMinutes();
        newLessonCourseName = coursesDropdown.options[coursesDropdown.value].text;
        newLessonGroupeName = groupDropdown.options[groupDropdown.value].text;
        newRepeatAfterWeekValue = repeatAfterWeek_InputField.text;
        /*
                print(newLessonDate);
                print(newLessonTime);
                print(newLessonCourseName);
                print(newLessonGroupeName);
                print(newRepeatAfterWeekValue);*/
    }

    private void UpdateGroupDropdown()
    {
        List<string> groupNamesList = new List<string>();

        foreach (StudentGroup grouop in groupsList)
        {
            groupNamesList.Add(grouop.Name);
        }

        UpdateDropdownInfo(groupDropdown, groupNamesList);
    }

    private void UpdateCoursesDropdown()
    {
        UpdateDropdownInfo(coursesDropdown, coursesList);
    }

    private void UpdateDropdownInfo(TMP_Dropdown dropdown, List<string> options)
    {
        dropdown.ClearOptions();
        dropdown.AddOptions(options);
    }

    private bool DateNotInThePast(string date)
    {
        DateTime todayDate = DateTime.Today;
        DateTime lessonDate = TimeManager.GetDayByString(date);

        return lessonDate >= todayDate;
    }

    private void SpawnLessonVisual(string time, string groupName, string courseName)
    {
        GameObject spawnedLessonVisualObject = Instantiate(lessonVisualPrefab.gameObject,
lessonsGrid);
        SingleLessonVisual spawnedLessonVisual =
spawnedLessonVisualObject.GetComponent<SingleLessonVisual>();
        spawnedLessonVisual.SetupVisual(FixTimeString(time), groupName, courseName);
    }

    private void SpawnLessonVisual(Lesson lesson)
    {
```

```
        GameObject spawnedLessonVisualObject = Instantiate(lessonVisualPrefab.gameObject,
lessonsGrid);
        SingleLessonVisual spawnedLessonVisual =
spawnedLessonVisualObject.GetComponent<SingleLessonVisual>();
        spawnedLessonVisual.SetupVisual(FixTimeString(lesson.time), lesson.group,
lesson.course);
        spawnedLessonVisual.GetDeleteButton().onClick.AddListener(() =>
            DeleteLesson(spawnedLessonVisualObject, lesson));
    }

    public void DeleteLesson(GameObject lessonVisual, Lesson lesson)
    {
        Destroy(lessonVisual);
        firebaseManager.DeleteLesson(lesson.date, lesson.time);
    }

    private string FixTimeString(string time)
    {
        //string doubleDotTime = time.Replace('-', ':');
        string[] timeParts = time.Split('-');

        int firstPartIndex = 0;
        int secondPartIndex = 1;

        if (timeParts[secondPartIndex].Length < minutesSymbolCount)
        {
            timeParts[secondPartIndex] += "0";
        }
        string finalTimeString = timeParts[firstPartIndex] + ":" +
timeParts[secondPartIndex];

        return finalTimeString;
    }



    public void AddNewLesson()
    {
        InitializeNewGroupData();

        if (DateNotInThePast(newLessonDate))
        {
            print("Yes, you can :) ");
            firebaseManager.AddNewLesson(newLessonDate, newLessonTime, newLessonCourseName,
                                         newLessonGroupeName,
int.Parse(newRepeatAfterWeekValue));
            SpawnLessonVisual(newLessonTime, newLessonGroupeName, newLessonCourseName);
        }
        else
        {
            print("No, you can't :( ");
        }
    }

    public bool HasGroupAndCourse()
    {
        return coursesList.Count > 0 && groupsList.Count > 0;
    }


}

public class Lesson
{
    public string date { get; private set; }
```

```
    public string time { get; private set; }
    public string course { get; private set; }
    public string group { get; private set; }
    public string repeatWeekCount { get; private set; }

    public Lesson(string date, string time, string course, string group, string
repeatWeekCount)
    {
        this.date = date;
        this.time = time;
        this.course = course;
        this.group = group;
        this.repeatWeekCount = repeatWeekCount;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class SimpleDialogWindow : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI windowNameTmp;
    [SerializeField] private TextMeshProUGUI messageTmp;

    public void SetupWindow(string windowText, string messageText)
    {
        windowNameTmp.text = windowText;
        messageTmp.text = messageText;
    }

    private void OnDisable()
    {
        Destroy(gameObject);
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;


public class SingleGridItemVisual : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI sudentName;
    [SerializeField] private Button deleteButton;

    public void SetName(string name)
    {
        sudentName.text = name;
    }
    public string GetName()
    {
        return sudentName.text;
    }

    public Button GetDeleteButton()
    {
        return deleteButton;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class SingleGroupeVisual : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI groupNameText;
    [SerializeField] private Button editGroupButton;
    [SerializeField] private Button deleteGroupButton;


    public void SetGroupNameText(string text)
    {
        groupNameText.text = text;
    }

    public string GetGroupNameText()
    {
        return groupNameText.text;
    }

    public Button GetEditGroupButton()
    {
        return editGroupButton;
    }
    public Button GetDeleteGroupButton()
    {
        return deleteGroupButton;
    }
}


using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class SingleLessonVisual : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI timeText;
    [SerializeField] private TextMeshProUGUI groupNameText;
    [SerializeField] private TextMeshProUGUI courseNameText;
    [SerializeField] private Button deleteLessonButton;

    public void SetupVisual(string time, string groupName, string courseName)
    {
        timeText.text = time;
        groupNameText.text = groupName;
        courseNameText.text = courseName;
    }

    public Button GetDeleteButton()
    {
        return deleteLessonButton;
    }
}
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TimeManager
{
```

```
    public static DateTime GetDayByString(string year_month_day)
    {
        string[] dateParts = year_month_day.Split('-');
        if (int.TryParse(dateParts[0], out int year) &&
            int.TryParse(dateParts[1], out int month) &&
            int.TryParse(dateParts[2], out int day))
        {
            DateTime dateTime = new DateTime(year, month, day);
            return dateTime;
        }
        else
        {
            Debug.LogError("Invalid date string");
            return DateTime.Now;
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class Cleaner
{
    public static void ClearContainer(Transform container)
    {
        foreach (Transform item in container)
        {
            GameObject.Destroy(item.gameObject);
        }
    }

    public static void ClearInputField(TMP_InputField inputField)
    {
        inputField.text = "";
    }
}
```