МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

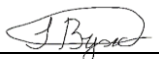
Факультет прикладної математики та інформатики
(повне найменування назва факультету)

дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Дипломна робота

СТВОРЕННЯ ВЕБ-САЙТУ ДЛЯ ПЛАНУВАННЯ ТА ЕФЕКТИВНОГО КОНТРОЮTIME MANAGEMENT

Виконав: студент IV курсу, групи ПМі-43с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)


Вужинський І-Р. А.
(підпис) (прізвище та ініціали)

Керівник Коркуна Н. М.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики _____
Кафедра Дискретного аналізу та інтелектуальних систем _____
Спеціальність 122 «Комп'ютерні науки»_____
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____

"31 "серпня 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ У РОБОТУ СТУДЕНТУ

Вужинському Івану-Роману Андрійовичу____
(прізвище, ім'я, по батькові)

1 .Тема роботи: Веб-сайт для планування та ефективного контролю time management

керівник роботи: асистент кафедри дискретного аналізу та інтелектуальних систем

Коркуна Наталія Михайлівна ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **" 13" вересня 2022 року № 15**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані до роботи Документація по мові програмування C# та .NET 7, , IDE Visual Studio 2022 та Visual Studio Code, .NET 7, React 18 та його документація, бібліотека MaterialUi, інтернет ресурси, джерела про теорію графів.

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) Аналіз існуючих продуктів на ринку, аналіз і вибір технології, формулювання вимог та функціоналу сайту, розробка веб-додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Ілюстрації для ознайомлення з структурою проекту, ілюстрації налаштування проекту, ілюстрації різних сторінок веб-сайту та приклади його роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження теоретичних відомостей	20.10.2022	
2	Створення початкових вимог	27.12.2022	
3	Дослідження та вибір технологій реалізації	09.01.2023	
4	Створення першого прототипу API	20.04.2023	
5	Підключення бази даних	22.04.2023	
6	Створення сторінок дошки, класів, користувача та груп	02.05.2023	
7	Додавання технології drag-and-drop	07.05.2023	
8	Підключення авторизації	08.05.2023	
9	Вдосконалення інтерфейсу користувача	15.05.2023	
10	Тестування додатку	26.05.2023	
11	Оформлення дипломної роботи	31.05.2023	
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			

Студент _ _ **Вужинський І.-Р. А..**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Коркуна Н.М.**
(підпис) (прізвище та ініціали)

Реферат

Метою дипломної роботи було створення додатку, що полегшить планування та допоможе ефективно розподіляти час студентам.

Під час виконання дипломної роботи я дослідив мікросервісну архітектуру та її відмінності та переваги над монолітною. Також я ознайомився з методологіями розробки продуктів та планування проектів, таких як Agile, Kanban, Waterfall та інші.

Для написання Application Programming Interface я використовував ASP.NET Core, а для побудови бази даних – MS SQL Server та таку Object-Relational Mapping, як Entity Framework Core. Авторизація користувачів здійснюється за допомогою jwt-токенів, а паролі шехуються за допомогою алгоритму SHA256. Для реалізації Single Page Application, я використав бібліотеку React для Java Script, а для створення та доступу до загального стану додатку – Redux. За допомогою MaterialUI була створена стилізація.

Дослідивши наукові праці Jeffrey Saltz та Robert Heckman, я дійшов висновку про ефективність використання методології Kanban для навчання студентів, тож було прийнято рішення створити програмну реалізацію.

Авторизувавшись, користувач має змогу створювати записи на канбан-дошці, визначати їх пріоритетність та дедлайни, додавати користувачів до груп, відслідковувати поточну успішність і т.д. Кожен запис містить короткий опис, отримані бали, клас (предмет) та інші необхідні дані.

ЗМІСТ

ВСТУП

У сучасному швидкому світі, де час є одним з найцінніших ресурсів, ефективне управління часом стає надзвичайно важливим аспектом для досягнення успіху та досягнення поставлених цілей. Особливо це актуально для студентів, які поєднують навчання з іншими обов'язками. Вміння ефективно планувати та контролювати використання часу має вирішальне значення для досягнення академічного успіху та збалансованого життя.

З цією метою, розробка веб-застосунку, спрямованого на планування та ефективний контроль часу студентів, є важливим та необхідним кроком у сучасному цифровому світі, де навчання відбувається все більше в онлайн-форматі. Такий веб-застосунок може стати надійним та потужним інструментом для керування часом, надаючи студентам зручні та ефективні інструменти для планування, організації та відстеження їхніх навчальних завдань та проектів.

Завдяки цьому веб-застосунку, студенти зможуть легко створювати свій особистий розклад занять, встановлювати терміни виконання завдань та дедлайни для проектів. Вони зможуть додавати деталі, опис та пріоритет до кожного завдання, щоб краще його організувати. Окрім того, вони зможуть встановлювати нагадування, щоб не пропустити важливі події або дедлайни.

Окрім того, веб-застосунок буде мати функціонал спільної роботи, що дозволить студентам створювати завдання для всієї групи одразу.

Загалом, розробка такого веб-застосунку для студентів відповідає реальним потребам сучасної освіти та допоможе їм стати більш організованими, продуктивними та успішними. Використання цього інструменту сприятиме поліпшенню управління часом студентів,

забезпечить їм більшу самодисципліну та дозволить зосередитися на досягненні власних академічних та особистих цілей.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Методології управління проектами

Є декілька методологій управління часом та проектами загалом: Kanban, Waterfall, Agile, Scrum, Lean, PRINCE2 та Spiral. Детальніше:

- Waterfall (Каскадна модель) - це традиційна методологія, де проект поділяється на послідовні етапи, які виконуються послідовно від початку до кінця. Кожен етап вимагає завершення перед початком наступного. Цей підхід підходить для проектів з чітко визначеними вимогами та стабільними умовами.
- Agile є гнучким підходом до управління проектами, який базується на ітераційному та інкрементальному методу розробки. Agile акцентується на самоорганізації команди, інтерактивному плануванні та постійному вдосконаленні продукту.
- Scrum - це ітераційний підхід, де проект поділяється на короткі ітерації, називані спринтами. Команда працює над конкретним набором завдань протягом кожного спринту, після чого відбувається огляд та планування наступного спринту.
- Lean-підхід ставить акцент на ефективність та мінімізацію витрат. Він спрямований на усунення витрат та зайвого навантаження, щоб забезпечити швидку та якісну доставку продукту.
- PRINCE2 (Projects in Controlled Environments) - це структурована методологія управління проектами, розроблена для великих проектів. Вона надає ретельно визначені ролі, процедури та шаблони для ефективного управління проектом.

- Spiral поєднує ітераційний підхід з ризик-орієнтованим управлінням. Проект прогресує через послідовні ітерації, з кожною ітерацією враховуючи й вирішуючи ризики.

Однією з таких методологій є підхід «канбан». В його основі лежить канбан-дошка, яка є інструментом для візуалізації робочого процесу та керування завданнями. Вона дозволяє організовувати завдання, контролювати їх статус і пріоритети, а також полегшує спільну роботу команди.

Основна ідея канбан-дошки полягає в розділенні робочого процесу на фази або колонки, кожна з яких відповідає певному стану завдання, наприклад, "В очікуванні", "У процесі виконання" і "Завершено". Кожне завдання представляється карткою або стікером, який переміщується по колонках на дошці відповідно до його статусу.

Для ефективного застосування канбан-дошки:

1. Визначення фаз процесу. Загальноприйнятими є три згадані вище, хоча їх кількість та суть можуть варіюватись залежно від налаштованих процесів команди.
2. Візуалізація завдань. Кожне із завдань повинно бути записане на окремій картці або стікері та містити усю необхідну інформацію.
3. Організація пріоритетів. Картки зі завданнями мають бути розташованими відповідно до їх пріоритету. Ви можете використовувати кольорові мітки або номери для позначення пріоритетів.
4. Регулярне оновлення завдань. Коли починаєте працювати над завданням, перемістіть його картку до відповідної колонки. Як тільки завдання виконане, перемістіть його наступну колонку.

Підтримуйте дошку в актуальному стані, постійно оновлюючи її.

5. Використання засобів позначення. Додайте на дошку додаткові елементи для позначення терміновості, статусу чи призначення завдань. Наприклад, використовуйте кольорові прапорці або позначки для виділення особливих вимог або дедлайнів.
6. Регулярні рецензії. Проводьте регулярні рецензії вашої канбан-дошки для перегляду прогресу, ідентифікації можливих затримок або проблем і перерозподілу завдань, якщо потрібно.
7. Встановлення лімітів. Введіть обмеження на кількість завдань, які можуть бути у кожній колонці або призначені кожні людині одночасно. Це допоможе уникнути перевантаження і забезпечити раціональний розподіл роботи.

1.2 Платформа .NET

Для розробки вище згаданого застосунку я використовую технології .NET 7 для створення API (Application Programming Interface) та React для створення інтерфейсу користувача. Розповім детальніше про вибір технологій:

.NET надає широкий спектр багаторазових компонентів та бібліотек, які полегшують розробку бекенду. Наприклад, ASP.NET забезпечує потужні інструменти для розробки веб-додатків, Entity Framework - для роботи з базами даних. Також .NET відомий своєю хорошою продуктивністю та масштабованістю. Він надає широкі можливості для оптимізації та розпаралелювання коду, що дозволяє створювати швидкі та ефективні бекенд-додатки.

1.3 Архітектура

Я обрав мікросервісну архітектуру, адже вона має декілька переваг:

Перш за все, мікросервіси пропонують потужну модуляризацію, уникнення небажаних залежностей та взаємодію лише через явні інтерфейси, таким чином запобігаючи проблемам, що виникають при використанні монолітної архітектури з часом.

З цього витікає, що невеликі мікросервіси додатково полегшують заміну окремих модулів. Якщо виникає потреба, мікросервіс може бути повністю переписаний.

Початок роботи з архітектурою на основі мікросервісів є легким і надає переваги одразу, навіть при роботі зі старими системами: замість додавання до старого заплутаного коду нового заплутаного коду, систему можна покращити за допомогою мікросервісу. З таким підходом необов'язково повністю замінювати старий код. Крім того, мікросервіс не прив'язаний до технологічного стеку системи старого типу, а може бути розроблений за допомогою сучасних підходів.

Мікросервіси пришвидшують введення продукту на ринок. Як вже зазначалося, мікросервіси можуть бути введені в експлуатацію поодиноко. Якщо команди, що працюють над великою системою, відповідають за один або кілька мікросервісів, і якщо для реалізації нового функціоналу потрібні зміни лише в цих мікросервісах, кожна команда може розробляти та вводити свою частину в експлуатацію без витрати часу на координацію з іншими командами. Це дозволяє багатьом командам працювати над численними функціями паралельно та введення більшої кількості змін в експлуатацію за менший час, ніж це було б можливо з монолітним розгортанням.

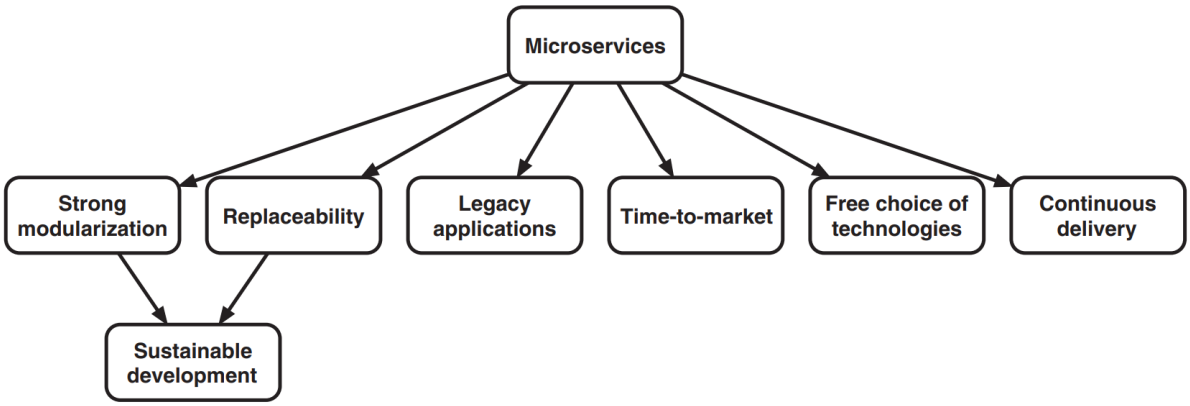


Рис. 1 – основні переваги мікросервісної архітектури

Створення архітектури проекту базується на підході, що розділяє операції команд та запитів у системі на дві різні моделі - CQRS (Command Query Responsibility Segregation). Концепція CQRS виникла з бажання збільшити прозорість, масштабованість та продуктивність системи шляхом розрізання логіки обробки команд і запитів.

Основна ідея CQRS полягає в тому, що команди (запити на зміну стану системи) та запити (запити на отримання стану системи) обробляються окремо. Він використовує дві різні моделі для цих операцій:

- Модель команд (Command Model): Ця модель використовується для обробки команд, які змінюють стан системи. Команди передаються до моделі команд, яка містить бізнес-логіку і валідацію для збереження змін у стані системи. Ця модель зазвичай підтримує операції створення, оновлення та видалення даних.
- Модель запитів (Query Model): Ця модель використовується для обробки запитів, які повертають дані з системи. Вона може бути спеціально оптимізованою для швидкого і ефективного виконання запитів та отримання необхідних даних. Модель запитів не змінює стану системи, а лише надає доступ до поточного стану.

Використання CQRS має декілька переваг:

1. Чітке розділення відповідальності: CQRS дозволяє чітко розділити логіку обробки команд і запитів, що полегшує розуміння та підтримку коду.
2. Оптимізація продуктивності: За допомогою окремих моделей для команд і запитів можна оптимізувати їх для різних потреб. Наприклад, модель запитів може бути швидкою та оптимізованою для частого доступу до даних.
3. Швидке впровадження змін: Зміни в логіці команд не впливають на логіку запитів і навпаки. Це дозволяє швидко впроваджувати зміни в одній частині системи без впливу на інші.
4. Масштабованість: CQRS дозволяє масштабувати команди та запити незалежно один від одного. Це дає можливість оптимізувати ресурси для різних типів операцій.

За основу взаємодії самих мікросервісів я обрав патерн Mediator. Патерн проектування Mediator є корисним, коли кількість об'єктів стає настільки великою, що стає складно підтримувати посилання на ці об'єкти. Mediator, в сутності, є об'єктом, який інкапсулює спосіб взаємодії одного

або декількох об'єктів між собою. Патерн проектування Mediator контролює, як ці об'єкти спілкуються між собою та допомагає зменшити кількість залежностей, які вам потрібно керувати. У нього об'єкти не спілкуються між собою безпосередньо, а через посередника. Коли об'єкту потрібно спілкуватися з іншим об'єктом або набором об'єктів, він передає повідомлення посереднику. Посередник потім передає повідомлення кожному об'єкту-отримувачу у формі, зрозумілій для нього. Шляхом усунення прямої комунікації між об'єктами, патерн проектування Mediator сприяє слабкому зв'язуванню. Іншою перевагою використання патерна проектування Mediator є покращення читабельності і підтримуваності коду.

1.4 База даних

Для взаємодії з базою даних я обрав підхід code first та Entity Framework Core в якості O/RM. Entity Framework Core – це кросплатформенний об’єктно-реляційний маппер для розробки програмних додатків, що був розроблений зосереджуючись на модернізації, спрощенні та полегшенню використання функціональності порівняно з його попередниками. EF Core забезпечує зручний спосіб взаємодії з базами даних, дозволяючи розробникам працювати з даними у форматі об'єктів та використовувати LINQ (Language Integrated Query) для запитів до даних. Вона підтримує різноманітні типи баз даних, включаючи SQL Server, MySQL, PostgreSQL, SQLite та інші. EF Core також надає можливості міграції баз даних, що дозволяє автоматично змінювати схему бази даних на основі змін у моделі даних, і має вбудовану підтримку асинхронної роботи, що сприяє покращенню продуктивності та швидкодії додатків.

Також я використовую підхід "Code First", що дозволяє перетворити класи в моделі домену, тому нам необхідно дуже уважно проектувати наші

класи моделей. Решта роботи виконується Entity Framework. Це перевага "Code First" підходу, де класи моделі стають моделями даних.

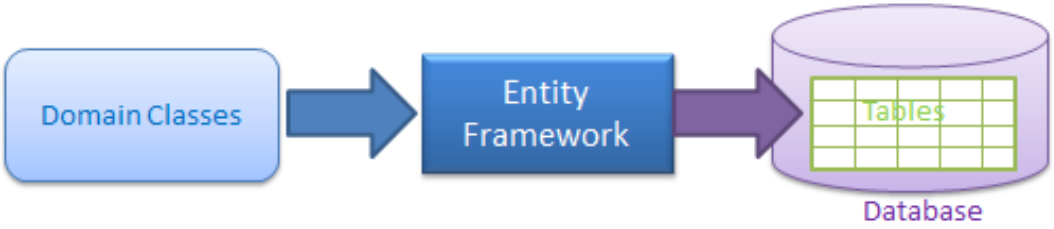


Рис. 2 – принцип роботи code first

Як базу даних я обрав SQL Server, що складається з колекції таблиць, які зберігають певний набір структурованих даних. Таблиця містить набір рядків, також відомих як записи або кортежі, та стовпців, відомих також як атрибути. Кожен стовпець у таблиці призначений для зберігання певного типу інформації, наприклад, дат, імен, сум в доларах та чисел. У межах кожної бази даних є одна або багато груп власності об'єктів, які називаються схемами. У кожній схемі є об'єкти бази даних, такі як таблиці, представлення і збережені процедури. Деякі об'єкти, такі як сертифікати та асиметричні ключі, містяться в межах бази даних, але не містяться в межах схеми. Бази даних SQL Server зберігаються в файловій системі в файлах. Файли можуть бути згруповані в файлові групи. Сама база даних знаходиться в Docker-контейнері.

```
version: "3"
services:
    sqlserver:
        image: mcr.microsoft.com/mssql/server:2019-latest
        container_name: sqlserver_timemanagment
        environment:
            SA_PASSWORD: Password123!
            ACCEPT_EULA: "Y"
        ports:
            - "1433:1433"
```

Рис.3 – docker-compose.yml для налаштування бази даних

Docker - це відкрите програмне забезпечення, яке дозволяє упаковувати, розповсюджувати та виконувати додатки в ізольованих контейнерах. Використання Docker дозволяє інкапсулювати додатки та їх залежності у легковажних, переносних контейнерах, які працюють однаково на різних середовищах.

Просто кажучи, контейнер - це ізольований процес на вашому комп'ютері, який відокремлений від усіх інших процесів на хост-машині. Ця ізоляція використовує простори імен ядра та cgroups, функціональні можливості, які існують у Linux протягом довгого часу. Під час запуску контейнера використовується ізольована файлова система. Ця спеціальна файлова система надається контейнерним образом. Оскільки образ містить файлову систему контейнера, він повинен містити все необхідне для виконання програми - всі залежності, конфігурації, скрипти, виконувані файли та інше. Образ також містить іншу конфігурацію контейнера, таку як змінні середовища, команду за замовчуванням для запуску та інші метадані.

Для того, щоб розгорнути цей контейнер з його yml фалу, я використовував Docker Compose. Docker Compose - це інструмент, розроблений для допомоги визначенню та спільному використанню багатоконтейнерних додатків. За допомогою Compose ми можемо створити файл у форматі YAML, в якому визначаються сервіси, і за допомогою однієї команди запустити або зупинити все.

Велика перевага використання Compose полягає в тому, що ви можете визначити стек вашої програми у файлі, зберігати його в корені репозиторію вашого проекту (тепер він контролюється системою контролю версій) і легко дозволити комусь іншому сприяти вашому проекту. Фактично, ви можете побачити, що досить багато проектів на GitHub/GitLab використовують саме цей підхід.

Для того, щоб підключитись до бази даних використовуючи EF, я перевизначив метод OnConfiguring в класі контексту, вписавши строку підключення (connection string),в якій вказана адреса контейнера з базою даних та дані авторизації до самого MSSQL SERVER, в якості аргументу.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer("Server=127.0.0.1, 1433;Database=tm_local;User Id=sa;Password=Password123!;" +
        "TrustServerCertificate=True;MultipleActiveResultSets=true",
        options => options.MigrationsAssembly(typeof(Context).Assembly.GetName().Name));
}
```

Рис. 4 – підключення до бази даних

Для оновлення схем бази даних або для додавання чи редагування таблиць, нам в нагоді знову став EF, який має вбудований функціонал для роботи з автоматичними міграціями, що викликаються у Package Manage Console.

Розглянемо основні команди:

1) enable-migrations:

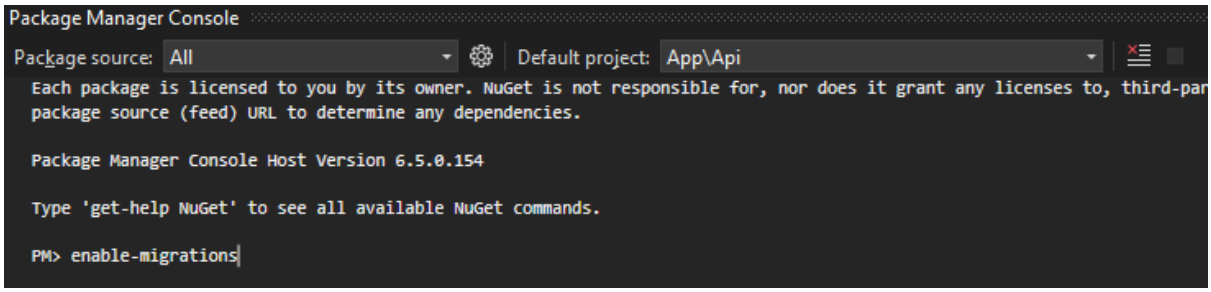


Рис. 5 – підключення міграцій до проекту

2) add-migration *назва міграції*:

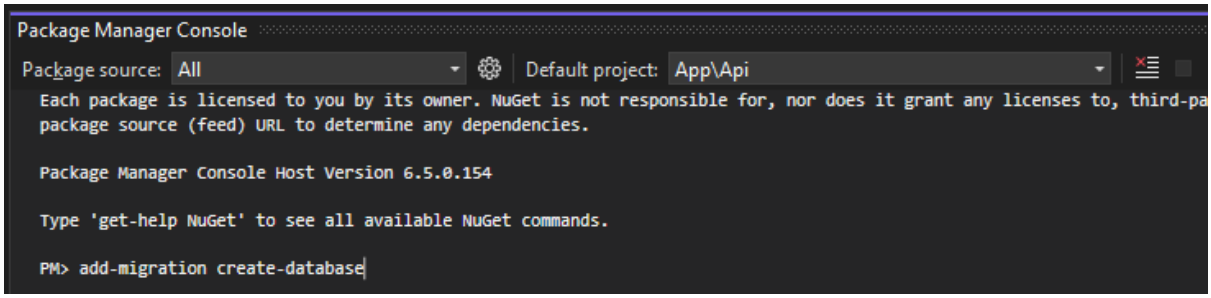


Рис.6 – створення міграції

3) update-database:

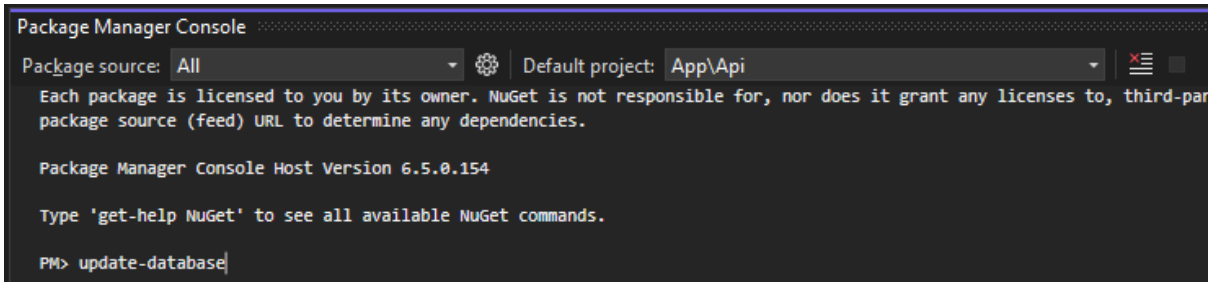


Рис. 7 – оновлення бази даних

4) у випадку, якщо була зроблена помилкова міграція, можна повернутись до будь-якої попередньої міграції використавши update-database -targetmigration: " *назва міграції*"

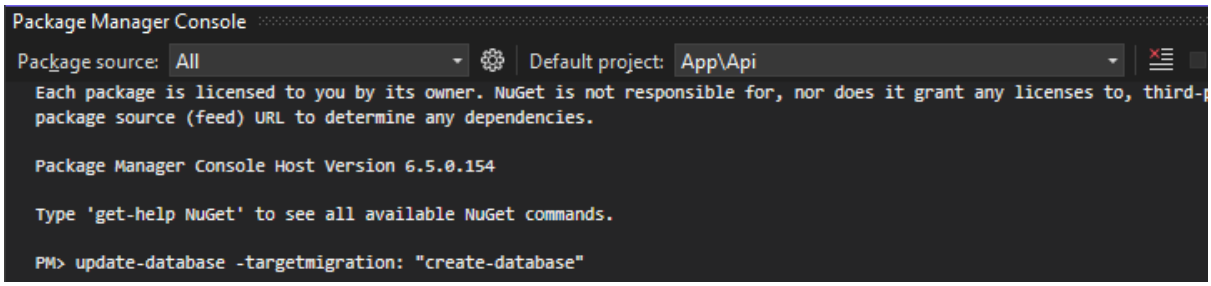


Рис. 8 – повернення до міграції “create-database”

1.5 Фронт-енд

Фронт-енд написаний на мові JavaScript з використанням бібліотеки React. JavaScript - це скриптова або програмна мова, яка дозволяє вам реалізовувати складні функції на веб-сторінках. Кожного разу, коли веб-сторінка виконує більше, ніж просто відображає статичну інформацію для перегляду - відображає актуалізовані контентні оновлення, інтерактивні карти, анімовану 2D/3D графіку, прокручувані відеоплеєри тощо - можна впевнитись, що JavaScript, ймовірно, використовується. Сучасний JavaScript – це “безпечна” мова програмування. Вона не надає низькорівневого доступу до пам’яті чи процесора, оскільки була створена для браузерів, які цього не потребують.

Можливості JavaScript значно залежать від середовища, у якому виконується скрипт. Вбудована в браузер JavaScript може робити все, що пов’язано з управлінням вебсторінками, взаємодією з користувачем та вебсервером.

Наприклад, JavaScript може:

- Додавати новий HTML-код на сторінку, змінювати наявний вміст, змінювати стилі.
- Реагувати на дії користувача, опрацьовувати натискання миші, переміщення вказівника, натискання на клавіші клавіатури.
- Відправляти запити через мережу до віддалених серверів, завантажувати та відвантажувати файли (так звані технології AJAX і COMET).
- Отримувати і надсилати куки, ставити запитання відвідувачам, показувати повідомлення.
- Запам’ятовувати дані на стороні клієнта (“local storage”, ”session storage”), які будуть доступні в майбутніх сесіях на цьому вебсайті.

Я використовую React – відкриту JavaScript бібліотеку для розробки користувацького інтерфейсу. Він використовується для побудови веб-додатків з високою продуктивністю та масштабованістю. React був розроблений Facebook і зроблений доступним для вільного використання та розповсюдження у вигляді відкритого джерела.

Основною ідеєю React є розбиття користувацького інтерфейсу на компоненти, які можна повторно використовувати. Компоненти - це малий, незалежний блок коду, який має свою внутрішню логіку та відображення. Вони можуть включати в себе HTML-подібний код, стилі, логіку обробки подій та багато іншого.

Одна з ключових особливостей React - це використання віртуального DOM (Document Object Model). Він підтримує внутрішнє подання структури веб-сторінки та швидко оновлює тільки необхідні частини DOM при змінах даних. Це дозволяє забезпечити високу продуктивність та ефективне оновлення користувацького інтерфейсу.

React також підтримує використання JSX - розширення синтаксису JavaScript, яке дозволяє включати HTML-подібний код безпосередньо в JavaScript. Це робить код React більш читабельним та легким для розуміння.

Для побудови компонентів свого додатку я використовую функціональні компоненти. Вони є спрощеним способом написання компонентів, що складаються тільки з render-метода і не мають власного стану. Замість визначення класу, який поширює React.Component, ми можемо створити функцію, яка приймає пропси і повертає те, що треба відрендерити. Функціональні компоненти коротші у написанні, і більшість компонентів можна оформити таким чином:

```
interface SpinnerProps {
  show: boolean;
  spinnerText?: string;
}

export const Spinner: FC<SpinnerProps> = ({ show, spinnerText }) => {
  const { t } = useTranslation();
  const classes = useStyles();

  return show ? (
    <Box className={classes.container}>
      <Box className={classes.spinnerPosition}>
        <CircularProgress size={40} />
        {spinnerText || t('global.pleaseWaitLoading')}
      </Box>
    </Box>
  ) : null;
};
```

Рис. 9 – Фрагмент коду функціонального компоненту

На цьому фрагменті коду також зображене використання TypeScript – оголошення інтерфейсу SpinnerProps та вказаний вбудований тип FC (FunctionalComponenet) для компоненту Spinner. TypeScript - це розширення JavaScript, що додає статичну типізацію до JavaScript коду. Вона розроблена компанією Microsoft і заснована на синтаксисі ECMAScript (стандарт JavaScript).

Основна ідея TypeScript полягає в тому, щоб забезпечити більшу надійність і підтримку в розробці великих проектів на JavaScript. Вона дозволяє визначати типи змінних, параметрів функцій, властивостей об'єктів тощо, що допомагає виявляти помилки на етапі компіляції.

Основні переваги TypeScript:

- Статична типізація: TypeScript дозволяє вказувати типи змінних, що дозволяє виявляти помилки типів на етапі компіляції. Це полегшує відлагодження коду та забезпечує більшу надійність програм.
- Підтримка сучасних функцій JavaScript: TypeScript підтримує синтаксис ECMAScript 6 і більш нові функції JavaScript, такі як arrow functions, генератори, async/await і багато інших. Ви можете використовувати нові можливості JavaScript, навіть якщо ваш браузер не підтримує їх повністю.
- Розширена підтримка ООП: TypeScript підтримує розширену функціональність об'єктно-орієнтованого програмування, включаючи класи, спадкування, інтерфейси, модифікатори доступу тощо. Це робить код більш структурованим і легким для розуміння.
- Інструменти для розробки: TypeScript має велику кількість інструментів, таких як компілятор TypeScript (tsc), який перетворює код TypeScript у JavaScript, редактори коду з підтримкою TypeScript,

такі як Visual Studio Code, і широку спільноту розробників, яка активно підтримує мову.

Використовуючи TypeScript, розробники можуть покращити якість свого коду, забезпечити більшу безпеку його виконання та полегшити підтримку та розширення проектів на JavaScript.

Для керування станом додатку я використовую Redux. Redux - це бібліотека у JavaScript, що пропонує прогнозовану та однозначну модель оновлення стану, що спрощує управління станом додатків у великих та складних системах.

Redux надає просту та прогнозовану модель керування станом додатків, дозволяє зберігати стан додатку в одному місці та полегшує підтримку. Він часто використовується разом з React, але може бути використаний і з іншими фреймворками або бібліотеками JavaScript.

2.Наукове підґрунтя

Як вже згадувалось у вступі, за основу свого додатку я обрав канбан-дошку. Основною метою розробки застосунку було полегшити процес навчання та самоорганізацію студентів.

Я опирався на дослідження Джефрі Сальца та Роберта Гекмана "*Exploring Which Agile Principles Students Internalize When Using a Kanban Process Methodology*", висновки якого доводять, що у контексті гнучкого Agile підходу, проектно-орієнтоване навчання доведено здатним допомогти студентам здобути практичний досвід у роботі з нестабільними середовищами.

Суть дослідження:

Щоб допомогти студентам користуватися процесом Kanban, студентам було надано пояснення процесу, яке зайняло годину часу на занятті (включаючи запитання та відповіді студентів) і включало основні концепції Kanban: візуалізацію робочого процесу, обмеження роботи в процесі та фокусування на потоці роботи. Крім того, обговорювалися також можливі етапи завдання (тобто стовпці на дошці Kanban). Однак командам було надано свободу визначати стовпці на їхній дошці, які вони вважали найбільш корисними.

З точки зору завдань, кожну команду просили визначити, над чим вони хотіли б працювати (тобто завдання високого рівня, такі як зв'язування погодних даних з раніше зібраними даними). Ці ідеї були представлені (у пріоритетному порядку) у стовпці "To Do". Порядок пріоритетів завдань визначався самою командою студентів, і студенти могли змінювати пріоритет завдань у будь-який момент, як вони вважали за потрібне. Таким чином, наприклад, погляд команди на пріоритет майбутніх завдань (які були в стовпці "To Do") міг змінюватись після отримання інформації від одного виконаного завдання. Потім, якщо було дозволяла кількість

місць (в залежності від кількості одночасних завдань на кожному етапі), завданню дозволялося перейти до наступного стовпця на дошці. Іншими словами, коли завдання було виконано у певному стовпці, його переносили до наступного стовпця і так далі по всій дошці, поки завдання не було завершено. Якщо дошка дозволяє (в залежності від обмежень роботи в процесі), можна починати нові завдання. Для визначення та контролю роботи команди використовували Trello (www.trello.com), який є веб-інструментом для візуалізації дошки. Кожна команда також вирішила, який розмір завдань (частин роботи) має бути виконаний. Однак командам пояснювалося, що чим меншими та деталізованішими є завдання, тим легше команді розуміти можливі складнощі.

За результатом дослідження, коментарі студентів були спрямовані на розподіл праці, ефективну координацію, індивідуальну відповідальність та критичні практики для самоорганізованих команд. Нижче наведено кілька прикладів коментарів студентів:

"Kanban допомагав призначати та регулювати проектні завдання між членами команди."

"Kanban дозволяє команді самостійно управляти візуальними процесами та робочими потоками."

"Ми використовували Kanban, щоб переконатися, що наш проект знаходиться у процесі виконання, і що кожен з нас вносить свій внесок."

"Kanban покращує комунікацію між вами та іншими членами команди. Він надихає співпрацю в команді."

"Agile Kanban-дошки були корисні для координації між командою, де наші завдання були розподілені, а прогрес до завершення контролювався."

"Основною перевагою була можливість відстежувати кожне завдання, яке потребувало виконання, і знати, кому були призначені завдання."

3.ПРОГРАМНА РЕАЛІЗАЦІЯ

Я реалізував веб-застосунок. Коли користувач заходить на сайт вперше, він потрапляє на сторінку авторизації. У користувача є можливість ввести емейл та пароль або перейти на сторінку реєстрації. Важливо згадати, що я використовую хешування для паролів за допомогою алгоритму SHA256.

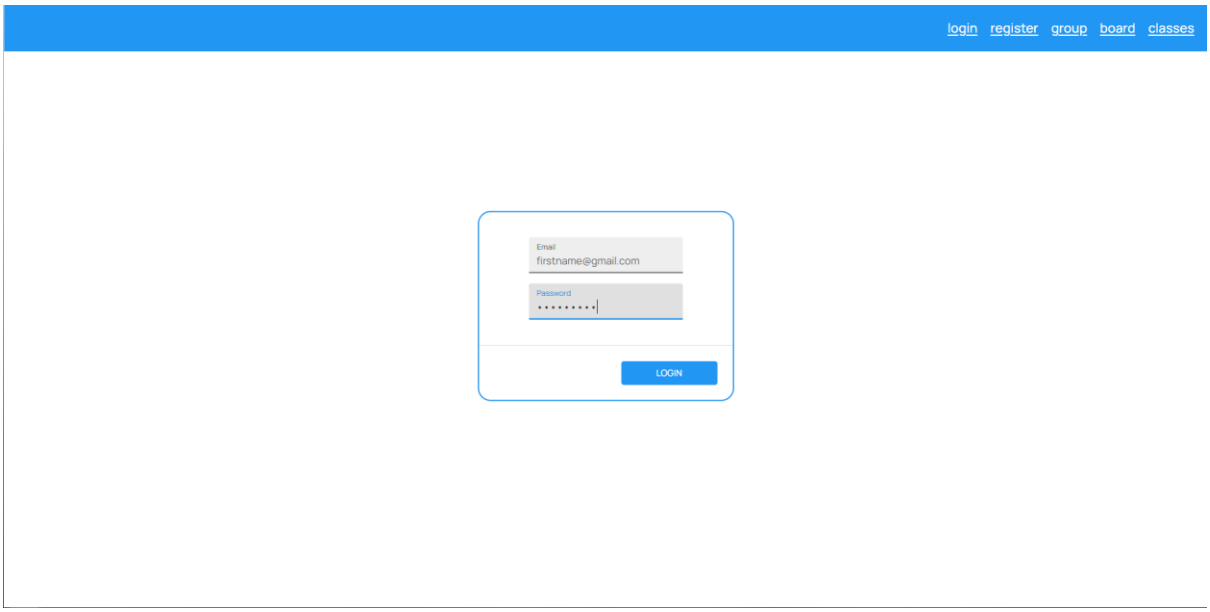


Рис. 10 – сторінка авторизації

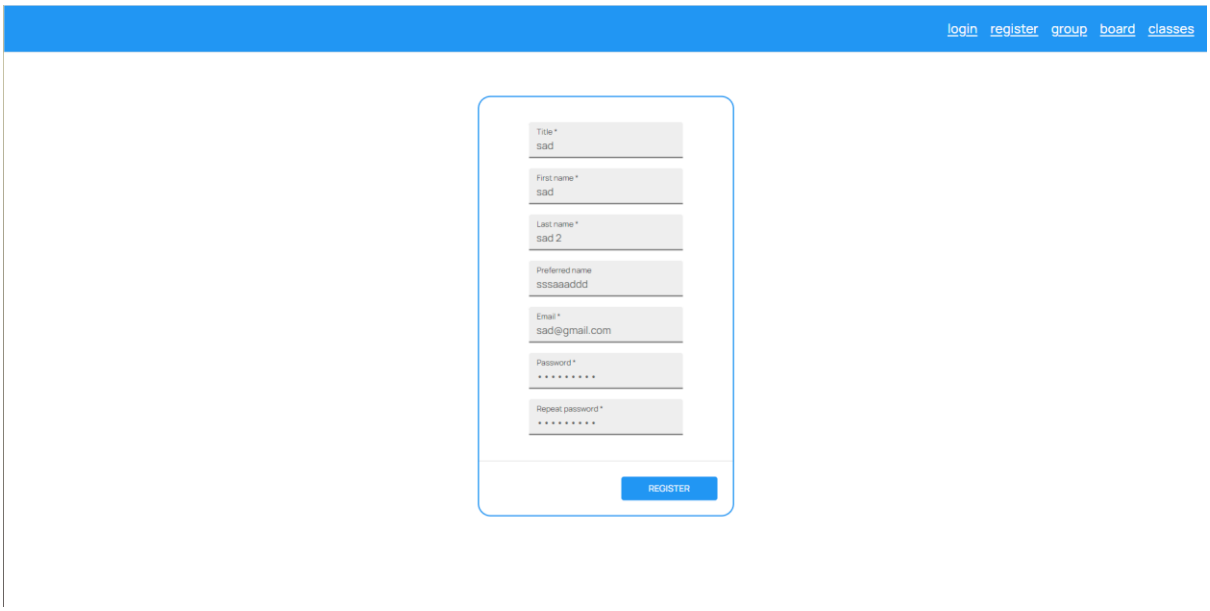


Рис. 11 – сторінка реєстрації

Після авторизації користувач має можливість переглянути свою дошку, класи (предмети), групи, а також змінити свої дані. Головною сторінкою є сторінка з дошкою, яка для нового користувача буде мати такий вигляд:

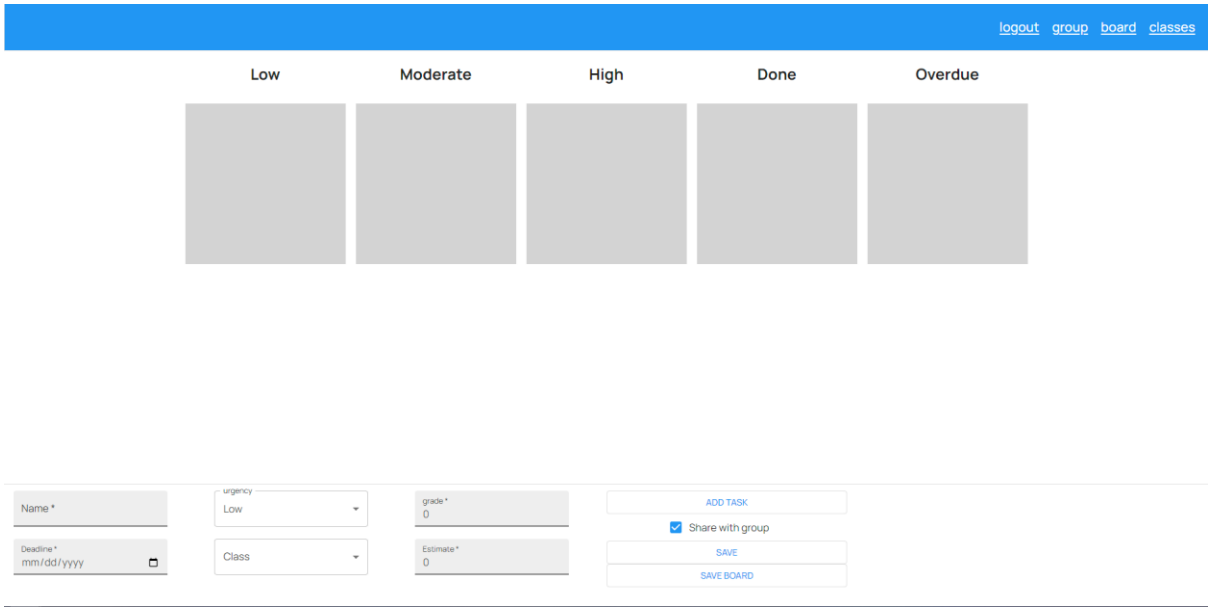


Рис. 12 – сторінка дошки

На цій сторінці ми бачимо поділ завдань на 5 рівнів пріоритетності:

- Низький пріоритет
- Середній пріоритет
- Високий пріоритет
- Здано
- Прострочено

Для кожного завдання, крім вибору пріоритету, користувач може встановити назву, крайній термін виконання, вибрати предмет, до якого відноситься завдання, поточну оцінку та час, який на думку користувача, є необхідною для виконання. Також при створенні завдання студент може поділитись зі своїми групами. Якщо він це зробить, для кожного студента з групи буде створена копія цього завдання. Самі завдання можна переміщати по дошці завдяки drag-and-drop, і, натиснувши «зберегти дошку», зберегти всі завдання з новими пріоритетами.

Вигляд заповненої дошки:

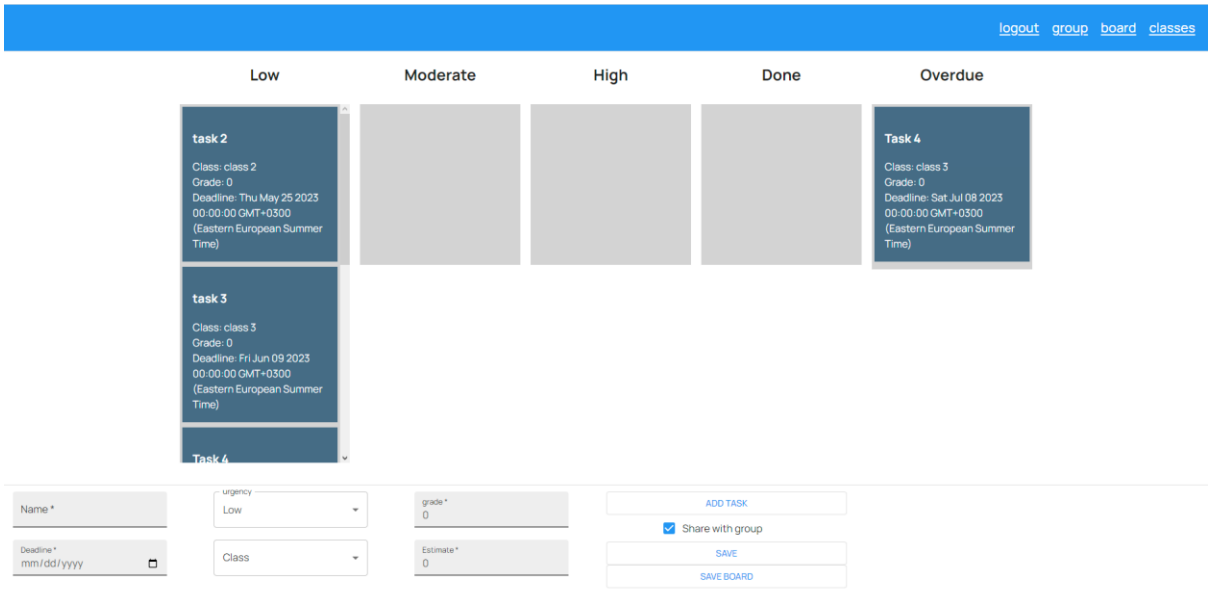


Рис. 13 – заповнена дошка

На рисунку 13 створено 7 завдань, 6 з яких мають низький пріоритет, і 1 є простроченою. Як бачимо, ці завдання відносяться до різних класів, що дає змогу зручно перевіряти всі завдання. Також якщо в окремій колонці буде дуже багато завдань, можна прогорнути її окремо від інших, що покращує користувацький досвід (user experience).

У користувача є можливість створювати групи. Це зручно для підготовки до роботи в парах або в групах для проектів. Для створення групи потрібно лише дати їй назву. Далі можна додавати інших користувачів.

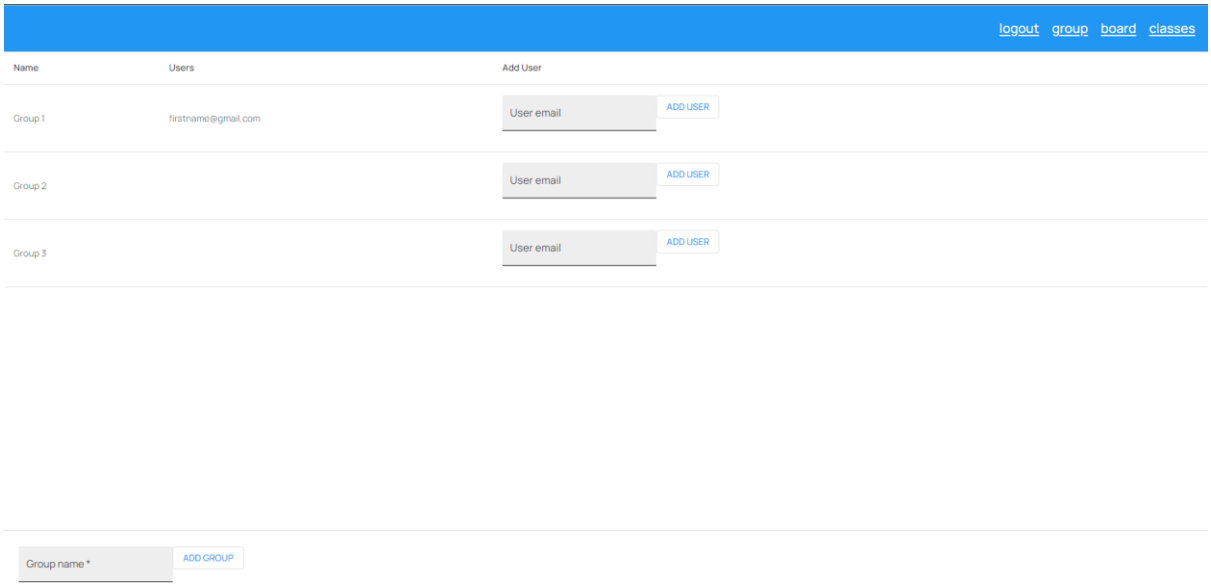


Рис. 14 – сторінка груп

Також користувач може переглядати свої предмети, а також створювати нові. Тут відображається назва предмету, необхідна кількість балів та поточна кількість балів, яка акумулюється автоматично з дошки користувача.

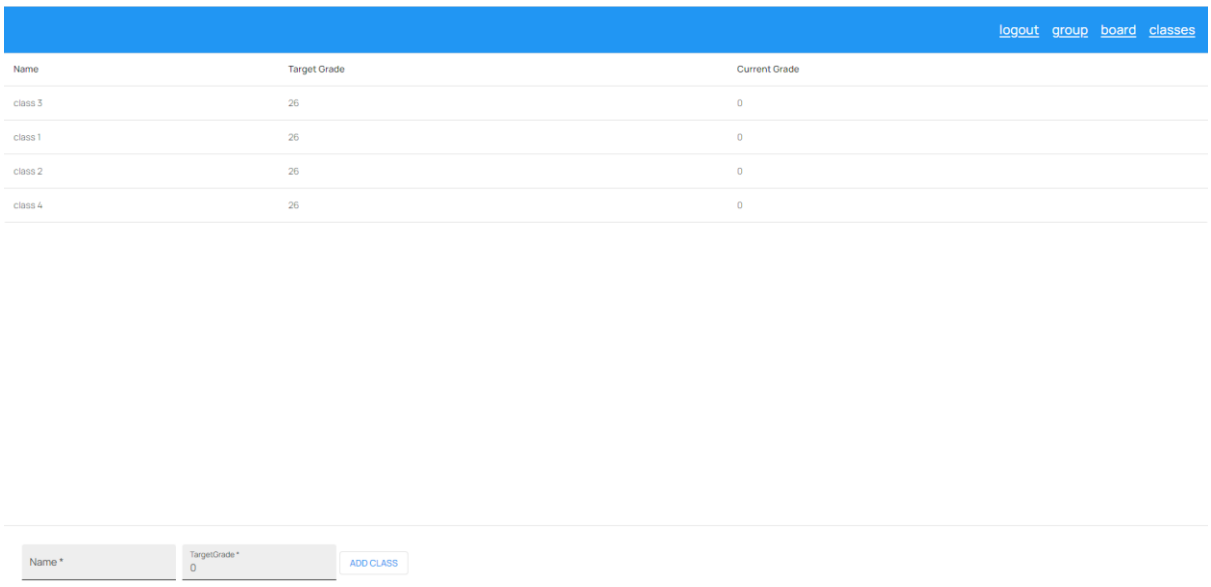
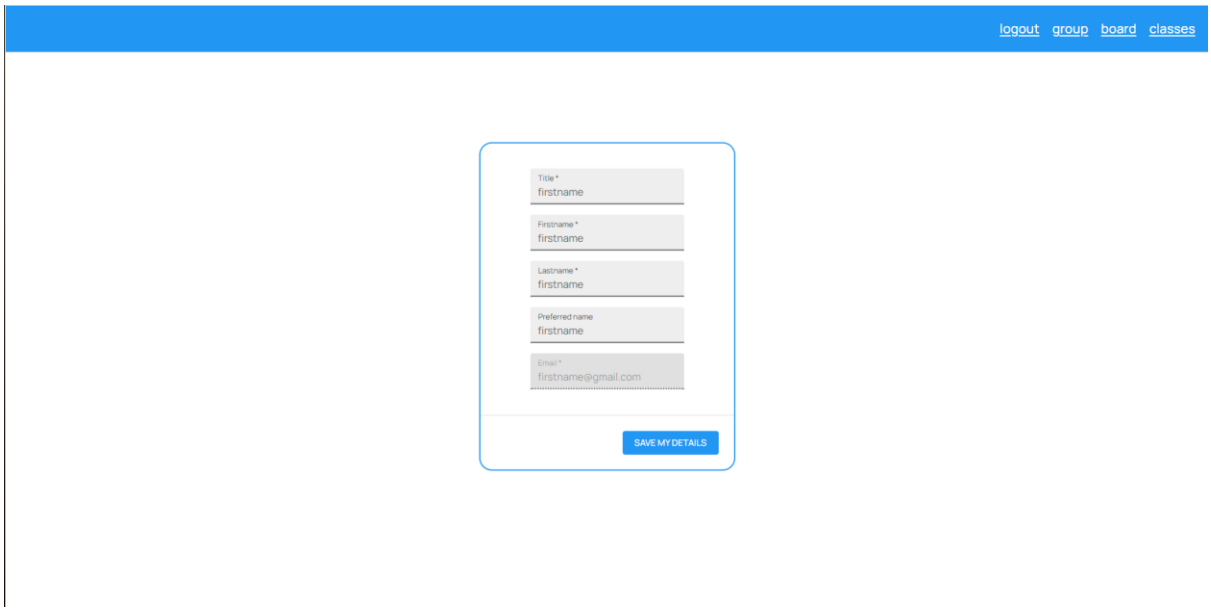


Рис. 15 – сторінка предметів

Користувач має можливість змінювати свої параметри, а саме:

- Ім’я
- Прізвище
- Ім’я, що буде відображатись усім
- Та «титул» (цільове використання – це умовне mr/mrs, проте користувач може використовувати все що захоче)



ВИСНОВКИ

Метою цієї роботи було створення максимально простого та зручного веб-додатку для студентів на основі канбан-дошки.

Для досягнення цієї цілі було використано .NET Core, MS SQL Server та React, розглянуто їх особливості, принципи роботи та взаємодію.

Під час роботи над проектом, було створено функціонал самої дошки, створення на керування групами, а також створення та керування предметами та автоматичний підрахунок балів для кожного предмета. В ході реалізації було розібрано більшість з кроків створення веб-застосунку, а результат є модульним, тому функціонал може бути легко розширений з урахуванням нових потреб.

ПОСИЛАННЯ

1. Microservices - Flexible Software Architecture (Eberhard Wolf): https://github.com/jidibinlin/Free-DevOps-Books-1/blob/master/book/Microservices%20-%20Flexible%20Software%20Architecture.pdf
2. How to use the mediator design pattern in C# (Joydip Kanjilal): https://www.infoworld.com/article/3204528/how-to-use-the-mediator-design-pattern-in-c.html
3. Entity Framework CodeFirst Approach (Bharath Y M): https://www.ecanarys.com/Blogs/ArticleID/228/Entity-Framework-CodeFirst-Approach
4. Документація Microsoft: https://learn.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-ver16
5. Документація Docker: https://docs.docker.com/get-started/
6. Документація React: https://uk.legacy.reactjs.org/tutorial/tutorial.html
7. Exploring Which Agile Principles Students Internalize When Using a Kanban Process Methodology (Jeffrey Saltz and Robert Heckman): http://jise.org/Volume31/n1/JISEv31n1p51.pdf