РЕФЕРАТ

Атестаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «інженерія програмного забезпечення». ТНТУ, 2023. Сторінок , таблиць, рисунків, презентація, диск з застосунком.

Тема: Розробка освітньої платформи на основі технології .Net.

Дипломна робота присвячена розробці додатку для зручного використання в процесі навчання, як доповнення до освітнього процесу. В атестаційній роботі бакалавра описано основні методики розробки веб -застосунків, приведені переваги використання SPA та WebApi технологій. Висвітлено можливості, що надає розробнику платформа ASP.NET Core. Розроблено клієнт за допомогою фрейморку Angular. Використовуючи uml - діаграми проілюстровані зв'язки класів, описані варіанти використання та діаграми послідовностей. Створено юніт тести, застосовуючи бібліотеку Xunit. В процесі розробки застосовано патерни проектування CQRS та Mediator.

ANNOTATION

Bachelor's thesis. Ivan Puluj Ternopil National Technical University, Department of Software Engineering, speciality 121 "Software Engineering". TNTU, 2023. Pages, tables, figures, presentation, disc with application.

Theme: Development of an educational platform based on .Net technology.

The thesis is devoted to the development of an application for easy use in the learning process, as a supplement to the educational process. The bachelor's thesis describes the basic methods of developing web applications, the advantages of using SPA and WebApi technologies. The opportunities provided by the ASP.NET Core platform are highlighted. The client is developed using the Angular framework. Using uml diagrams, class relationships are illustrated, use cases and sequence diagrams are described. Unit tests are created using the Xunit library. The CQRS and Mediator design patterns are used in the development process.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS (Java Script) – мова програмування, використовується для створення інтерактивних та динамічних веб-сторінок.

API (Application Programming Interface) – прикладний програмнийінтерфейс.

HTML (HyperText Markup Language) – мова розмітки гіпертексту.CSS (Cascading Style Sheets) – спеціальна мова стилю сторінок.

UML (Unified Modeling Language) – уніфікована мова моделювання.DOM (Document Object Model) – об’єктна модель веб документа.

Юніт тести – модульні тести, які перевіряють роботу окремих функцій чи методів.

ВСТУП

У сучасному світі, де технологічний прогрес набуває все більшого значення, онлайн сервіси для освіти стають необхідною складовою ефективного та доступного навчання. Розвиток цих сервісів має велике значення для забезпечення якісної освіти та поширення знань у всьому світі.

По-перше, онлайн сервіси дозволяють здобувати освіту у будь-якому часі та місці. Завдяки ним, студенти можуть навчатися зручним для них темпом, не залежно від географічного розташування та режиму роботи. Це особливо важливо для людей, які мають обмежений доступ до традиційних освітніх закладів, таких як люди з інвалідністю, працюючі батьки або ті, хто проживає у віддалених регіонах.

По-друге, розвиток онлайн сервісів для освіти сприяє зростанню якості навчання. Ці сервіси надають доступ до різноманітних навчальних матеріалів, інтерактивних завдань та відеолекцій, що допомагають студентам краще засвоювати матеріал і зрозуміти складні концепції. Багато сервісів також пропонують персоналізоване навчання, адаптоване до індивідуальних потреб та рівня знань студента. Це сприяє підвищенню ефективності навчання та досягненню кращих результатів.

По-третє, розвиток онлайн сервісів для освіти відкриває можливості для глобального обміну знаннями та співпраці. Вчителі та студенти з різних країн можуть взаємодіяти, обмінюватися ідеями та досвідом, що сприяє культурному розмаїттю та розширенню горизонтів. Онлайн сервіси також дають можливість проводити віртуальні конференції, семінари та курси, залучаючи провідних експертів та спеціалістів з усього світу. Нарешті, онлайн сервіси допомагають зменшити фінансові бар'єри в освіті. Вони можуть бути більш доступними та вигідними з економічної точки зору, оскільки вимагають менших витрат на

транспорт, проживання та матеріали. Багато онлайн курсів і ресурсів також пропонуються безкоштовно або за символічну плату, що дає можливість більшій кількості людей отримати якісну освіту.

З урахуванням всіх цих факторів, розвиток онлайн сервісів для освіти є надзвичайно важливим. Вони розширюють можливості навчання, полегшують доступ до якісної освіти та сприяють глобальному обміну знаннями. Забезпечення розвитку цих сервісів має великий потенціал для поліпшення освітніх можливостей людей по всьому світу та побудови суспільства, що базується на знаннях та інноваціях.

В сучасному цифровому світі використання технологій у сфері освіти стає все більш важливим. З метою поліпшення процесу навчання та розвитку студентів, необхідно розробляти та впроваджувати сучасні інструменти та платформи. Незважаючи на наявність різних освітніх платформ, багато з них мають обмежені можливості або недоліки, які потребують подальшого вдосконалення, деякі надто складні й переповнені функціоналом.

Ця дипломна робота присвячена розробці освітньої платформи, яка базується на потужній технології .Net. Використання даної технології дозволить створити ефективну та функціональну платформу, яка враховуватиме сучасні потреби освіти.

Основна мета цього дослідження полягає у розробці комплексної освітньої платформи, яка забезпечуватиме інтерактивність, доступність та ефективність процесу навчання. Ця платформа надаватиме студентам можливість отримувати знання та навички у зручний для них спосіб, шляхом надання різноманітних навчальних матеріалів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Одним з лідерів у наданні освітніх онлайн сервісів для доповнення навчального процесу та полегшення процесу навчання, електронних журналів та щоденників, розкладу занять та обліку дітей, бази навчальних матеріалів є українська компанія "Нові Знання".

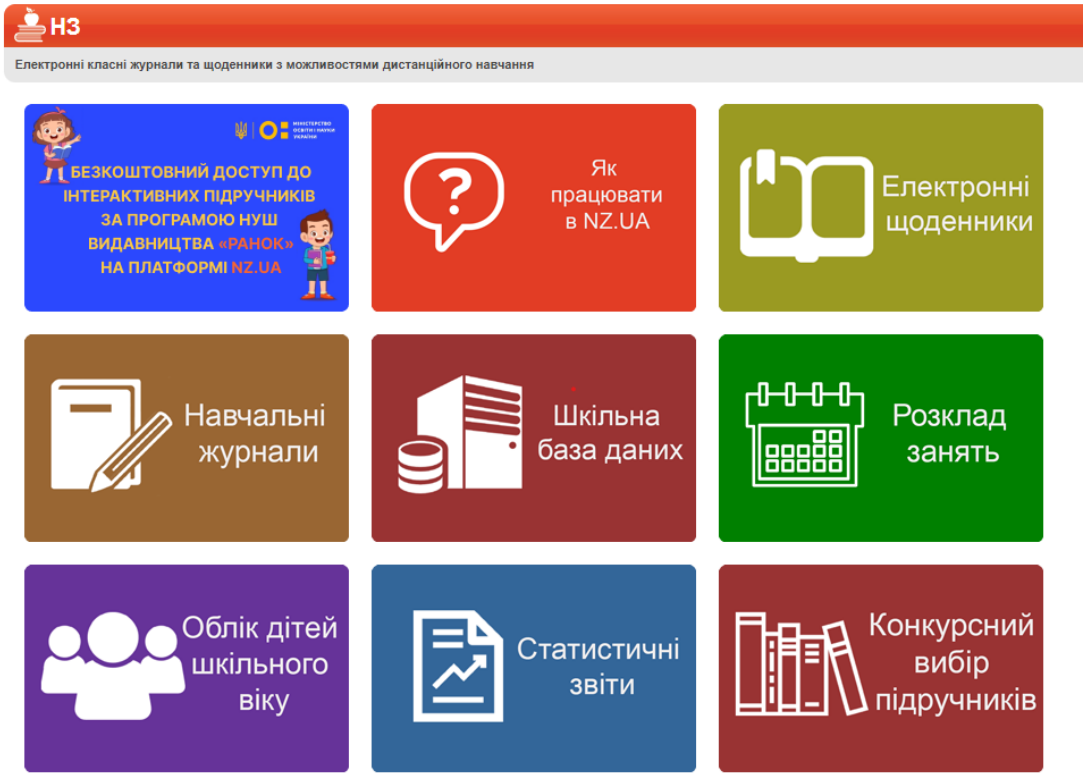

Рис. 1 – Веб – застосунок компанії Нові Знання.

Сучасний темп життя примушує людей змінювати звичні способи до навчання, дедалі частіше школи починають замінювати паперові журнали на

електронні аналоги, створюють класні чати, надсилають завдання та рішення у поштові скриньки.

У вирії цих змін опинився сервіс «НЗ», заснований ще у 2010 році закріпився на ринку та охопив значну кількість користувачів, яких стає дедалі більше.

На веб – сайті конкурента можна знайти багато популярних послуг:

- електронні щоденники;
- облік учнів;
- статистичні звіти;
- розклад занять [1].

Проте найбільшої популярності отримали електронні журнали. Вони дають змогу вчителям зручно формувати навчальний план, відслідковувати активність та успішність учнів, створювати електронні завдання та вносити зміни без зайвих зусиль. З допомогою цих онлайн засобів вчителі та учні значно зменшують затрати часу на засвоєння навчального матеріалу. Електронні журнали також дають можливість батькам у зручному форматі відслідковувати за успішністю їх дітей, аналізувати та помічати проблемні сфери навчання дитини. Електронні журнали надають можливість налаштування навчального процесу під потреби кожного учня. Вони дозволяють використовувати різні режими навчання, працювати в своєму темпі, повторювати матеріал, звертатися до додаткових ресурсів та матеріалів, що сприяє більш ефективному засвоєнню знань.

«НЗ» іде в ногу з часом і постійно оновлює свої сервіси, доповнює їх новими можливостями та покращує старі недоліки.

1.2 Обґрунтування вибору напрямку дослідження

У наш час ринок переповнений технологіями для побудови програмного забезпечення. Безліч мов програмування фреймворків, бібліотек. Існують різні підходити до створення веб – застосунків.

У світі, де технології швидко розвиваються, важливо використовувати сучасні технологічні рішення для розробки веб – застосунків, в тому числі й освітніх платформ. Використання технологій .Net, зокрема ASP.NET є актуальним та ефективним підходом до розробки високоякісних та інтерактивних освітніх ресурсів.

ASP.NET Core може працювати на різних платформах, таких як Windows, macOS і Linux [2].

Це дозволяє розробникам вибирати платформу розгортання, яка найкраще відповідає їх потребам. Технологія пропонує модульну архітектуру, що дозволяє розробникам вибирати лише ті компоненти, які їм потрібні. Фреймворк має велику кількість пакетів NuGet, які дозволяють легко розширювати можливості додатків.

Розробники можуть використовувати інструменти розробки, такі як Visual Studio, Visual Studio Code, або навіть командний рядок, для створення ASP.NET Core додатків. ASP.NET Core підтримує мови програмування C#, F# і Visual Basic.

ASP.NET Core - це потужний фреймворк для розробки веб-додатків, який надає широкі можливості для створення швидких, масштабованих і переносних додатків на різних платформах. Він надає розробникам багато інструментів і бібліотек, які спрощують процес розробки і дозволяють побудувати високоякісні веб-додатки.

Для розробки клієнтської частини додатку добре підходить Angular.

Angular - це фронтенд-фреймворк з відкритим кодом, який розробляється командою Angular Team під керівництвом Google, спільнотою приватних

розробників та іншими компаніями. Він написаний на мові TypeScript. Angular почав свій шлях як AngularJS у 2010 році, але потім був повністю перероблений тією ж командою розробників у 2016 році і отримав назву Angular 2 [3].

Останні кілька років принесли швидкий розвиток фреймворків JavaScript. Цей процес розпочався з випуску jQuery - бібліотеки JavaScript у 2006 році, яка була створена для розробки інтерактивних веб-програм. Після цього відбувся значний прогрес у розвитку організованих front-end фреймворків. Цей розвиток сприяв появі концепції SPA (односторінкових застосунків) - веб-додатків, які динамічно змінюють компоненти веб-сторінки без повного перезавантаження сторінки.

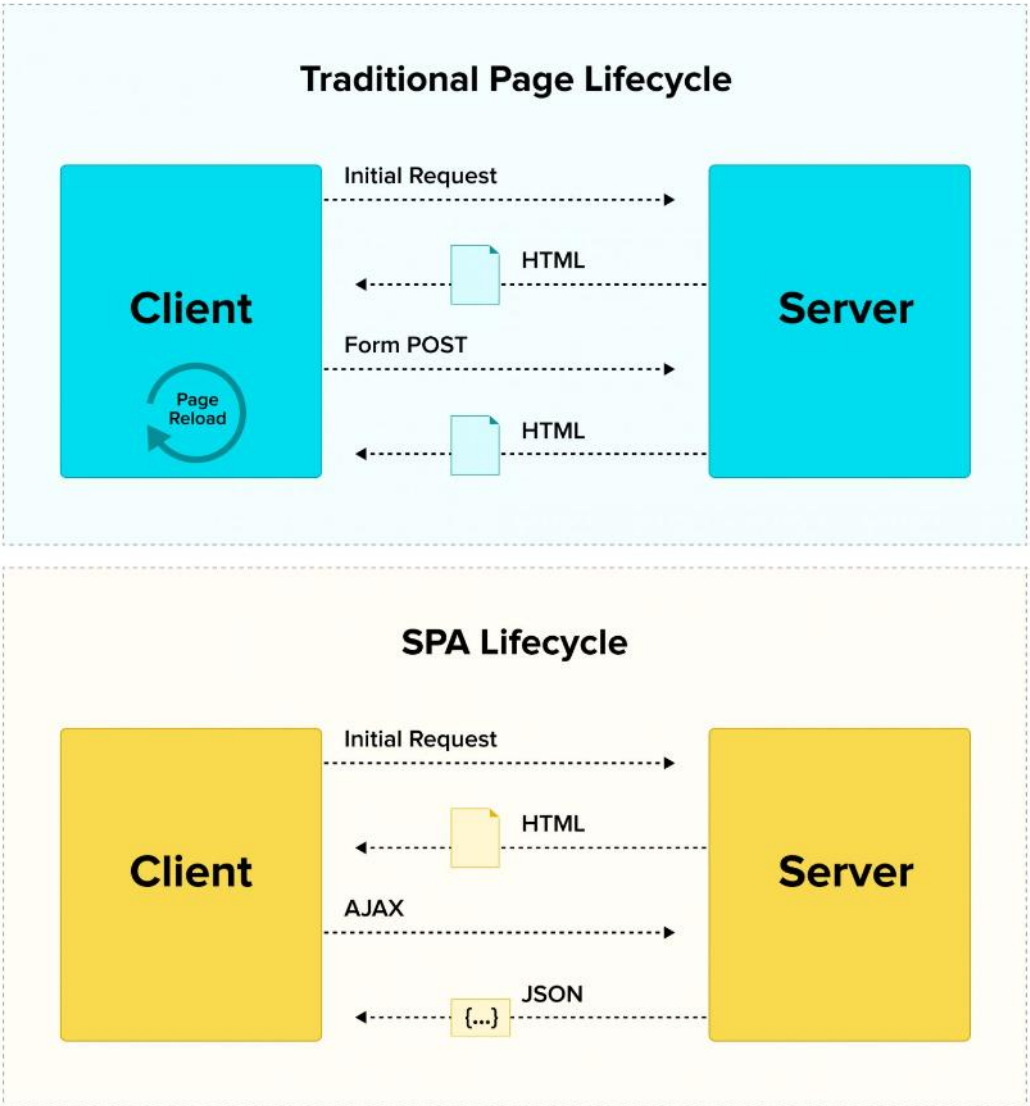

Рис. 2 – Життєвий цикл SPA додаткі

1.3 Технічний аспект проблеми

Важливою частиною процесу створення програмного забезпечення є вибір методології розробки програмного продукту. Найпопулярніші з них:

- екстремальне програмування;
- SCRUM.

Згідно з методом SCRUM, проєкт ділиться на ітерації (які називаються «спринт»), за 30 днів кожна. Перед початком спринту визначається функціональність, яка потрібна на даному етапі, після чого починає роботу команда розробників. Важливо, щоб протягом одного спринту вимоги залишалися незмінними [4].

З усіх гнучких методологій методологія екстремального програмування (XP) — найбільше відома. XP стоїть на чотирьох китах: Комунікація, Зворотній зв'язок, Простота і Сміливість. Із них слідують дванадцять практик, яким повинні слідувати проєкти, що використовують XP. Багато з цих практик являють собою старі перевірені техніки, які, тим не менш, багато встигли забути (включаючи більшість передбачуваних процесів). XP не тільки воскрешає до життя такі техніки, але і з'єднує їх таким чином, що всі вони підтримують і підсилюють один одного.

Для розробки системи було вибрано методологію Rational Unified Process (RUP), також відому як "Раціональний уніфікований процес".

RUP є архітектурно орієнтованою методологією, яка підкреслює важливість реалізації і тестування архітектури системи ще на початкових стадіях проекту.

Використання поняття "виконуваної архітектури" (Executable Architecture) є ключовим у RUP. Це означає, що додаток має мати функціонально працюючу архітектурну базу, що дозволяє реалізувати важливі архітектурні компоненти. Реалізація цих основ має відбуватися на найраніших етапах проекту, щоб виявити можливі проблеми та внести корективи до архітектурних рішень ще на початку проекту. Таким чином, на початкових ітераціях слід вибирати компоненти, які потребують реалізації більшої частини архітектури.

RUP активно використовує візуальні засоби для аналізу та проектування. Зазвичай для цього використовується нотація UML (Unified Modeling Language) та відповідні моделювальні інструменти, наприклад, Rational Rose. Модель предметної області документується у вигляді діаграми класів, а модель вимог до системи - за допомогою діаграм вимог. Взаємодія компонентів системи між собою описується за допомогою діаграм послідовності та інших відповідних засобів моделювання.

В якості архітектури продукту було обрано паттерн CQRS, він має одну з найважливіших переваг - можливість незалежного розширення та оптимізації кожної складової додатку. Ви можете створити оптимальні умови для кожної частини, не впливаючи на інші. Це сприяє підвищенню продуктивності та надає більше простору для майбутніх змін.

SQRS є архітектурним підходом, який відрізняє команди (запити на зміну стану системи) від запитів (запити на отримання даних з системи). Цей підхід рекомендує розділити модель домену на дві частини: командний модуль, який виконує обробку команд і зміну стану системи, і запитовий модуль, який відповідає за обробку запитів і повернення даних. Використання цього підходу дозволяє оптимізувати обробку запитів і команд, враховуючи їхні особливості [5].

Поєднання методології Rational Unified Process (RUP) і підходу CQRS (Command Query Responsibility Segregation) може стати цікавою комбінацією для розробки складних систем. RUP надає широкий фреймворк для керування

проектом та розробки програмного забезпечення, підкреслюючи значення архітектурної орієнтації, розподілення роботи на ітерації та використання візуальних засобів моделювання. RUP також рекомендує ранню реалізацію архітектурно значущих компонентів, що сприяє оцінці рішень та внесенню коректив на ранніх етапах проекту. Це допомагає створити стабільну та масштабовану архітектуру для системи.

Поєднання RUP і CQRS має кілька переваг:

- за допомогою RUP можна забезпечити систематичний підхід до розробки, враховуючи архітектурні аспекти;
- рання реалізація архітектурно значущих компонентів дозволяє виявити проблеми та внести корективи на ранніх етапах проекту.
- використання CQRS дозволяє виокремити командний та запитовий потоки, полегшуючи оптимізацію та масштабування кожного з них окремо.
- застосування RUP та CQRS сприяє створенню більш гнучкої, масштабованої та ефективної системи.

Проте, важливо мати глибоке розуміння обох методологій та враховувати контекст конкретного проекту під час їх впровадження.

2 ПРОЄКТУВАННЯ ОСВІТНЬОЇ ПЛАТФОРМИ

2.1 Розробка моделі предметної області

Для початку роботи з моделлю предметної області визначимо вимоги які ми ставимо перед програмним продуктом.

Додаток буде забезпечувати вчителів та їх учнів базовим функціоналом для онлайн навчання. Користувачі можуть створювати курси та приєднуватися до них. Власник курсу(вчитель) може створювати завдання та публікувати новини курсу. Користувачі, що приєднались до курсу(учні) можуть переглядати новини та надсилати результат виконання завдання. Вчитель має можливість оцінити результат виконання завдання, учень бачить свої оцінки.

Користувачі мають змогу редагувати дані свого профілю та змінити пароль.

Можна виділити наступні вимоги:

- Функціональні:
  1) Можливість реєстрації в додатку;
  2) Можливість входу в додаток;
  3) Можливість редагування профіля;
  4) Можливість зміни паролю;
  5) Можливість створення, редагування та видалення курсу;
  6) Можливість створення, редагування та видалення новини;
  7) Можливість створення, редагування та видалення завдання;
  8) Можливість приєднання до курсу;

9) Можливість додавання результату виконання завдання;
10) Можливість оцінювання результату виконання завдання;
11) Можливість перегляду оцінок.

- Не функціональні:
  1) Додаток повинен мати мінімалістичний дизайн;
  2) Доступ до додатку необхідно убезпечити за допомогою протоколу HTTPS;
  3) Система повинна застосовувати шифрування паролів

З наведених вище вимог можна виділити сутності варіантів використання:

- Авторизований користувач;
- Гість.

Щоб полегшити модифікацію та адміністрування зазначених вище структурних елементів, рекомендується реалізувати їх як індивідуальні програмні файли.

Також можна виділити моделі предметної області:

- Користувач;
- Курс;
- Новина;
- Завдання;
- Рішення до завдання;
- Оцінка;

2.2 Розробка бізнес моделі

Таблиця 1.1 – Варіанти використання системи

Код	Основний актор	Найменування	Формування
1	2	3	4
Г1	Гість	Реєстрація в системі	Для доступу в систему гість має змогу зареєструватися в системі для подальшої роботи
Г2	Гість	Вхід в систему	Для входу в систему гість має ввести свій логін та пароль для подальшої роботи
К1	Користувач	Керування курсом	Користувач має змогу керувати курсом
К2	Користувач	Керування новинами	Користувач має змогу керувати новинами

1	2	3	4
К3	Користувач	Керування завданнями	Користувач має змогу керувати завданнями
К4	Користувач	Редагувати профіль	Користувач має змогу змінювати дані у профілі

За допомогою табличної форми варіантів використання можна створити діаграму варіантів використання, щоб краще візуально сприймати зв'язки між акторами та варіантами використання. Ця діаграма демонструє функціонал, який потрібно реалізувати в системі, відображаючи акторів та варіанти використання у графічному вигляді.

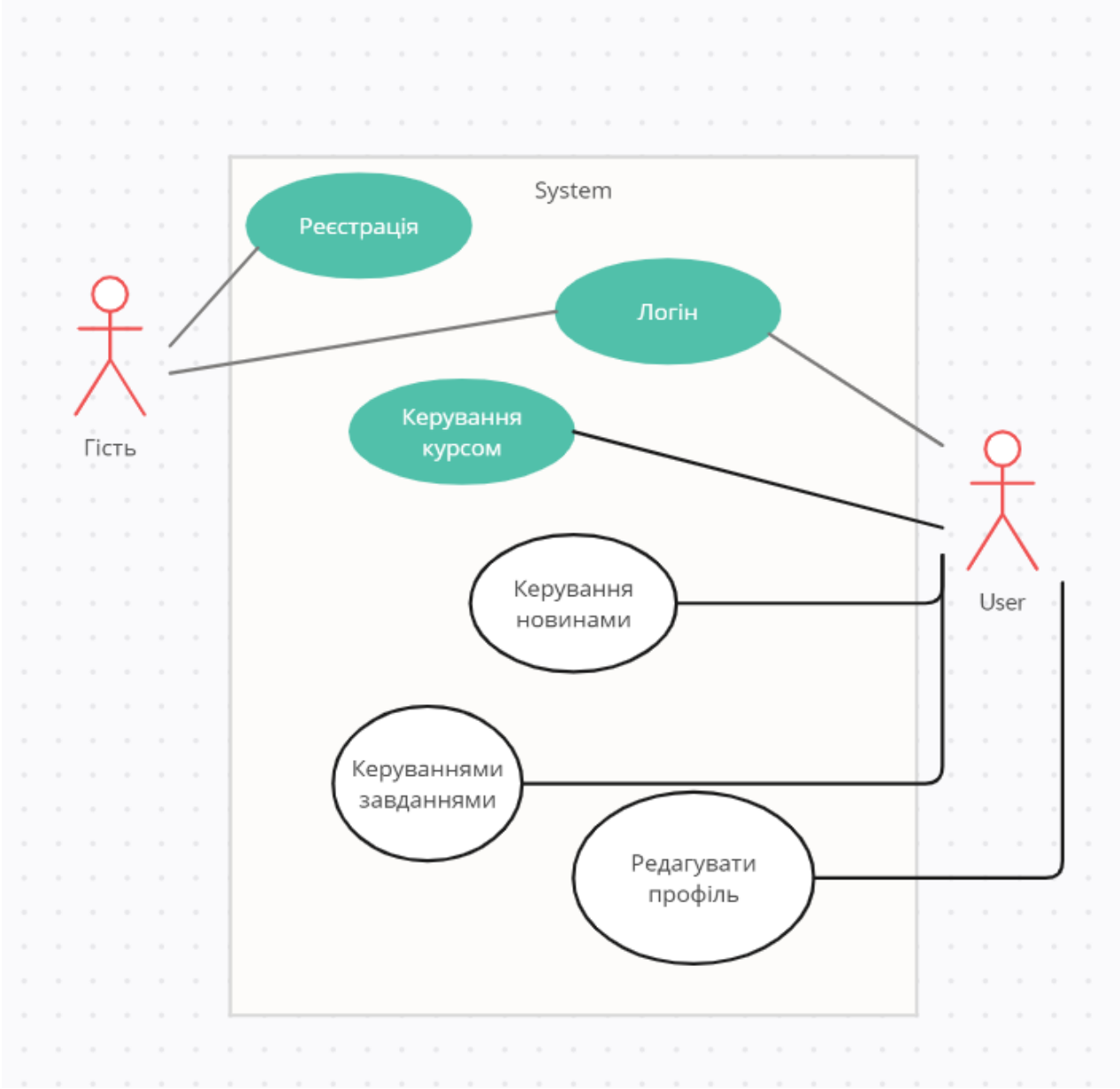

Рис. 3 – Діаграма варіантів використання.

2.3 Проєктування архітектури

Діаграма класів є статичним представленням структури моделі, яке відображає класи, типи даних, зміст та їх відношення. Вона дозволяє візуально представити статичну структуру системи, включаючи класи, інтерфейси, об'єкти та взаємодії між ними.

Кожен клас є основним будівельним блоком програмної системи і має назву, атрибути та операції. Діаграма класів використовує прямокутник, розділений на три області, для відображення цих елементів: верхня область містить назву класу, середня область - опис атрибутів (властивостей), а нижня область - назви операцій, які виконуються об'єктами цього класу.

Крім того, класи можуть мати модифікатори доступу, такі як private, protected та public, які визначають область доступу до їх членів. Атрибути класу визначають структуру даних, які зберігаються в об'єктах даного класу, тоді як операції визначають поведінку цих об'єктів. Об'єкти класу мають конкретні значення атрибутів, які можуть змінюватися під час виконання програми. З використанням діаграми класів можна краще розуміти статичну структуру системи та взаємозв'язки між класами в термінах об'єктно-орієнтованого програмування.

За допомогою отриманих відомостей про систему з першого розділу, ми готові перейти до проектування архітектури системи та визначення необхідних класів. Після визначення цих класів, ми зможемо побудувати діаграми класів за допомогою мови UML.

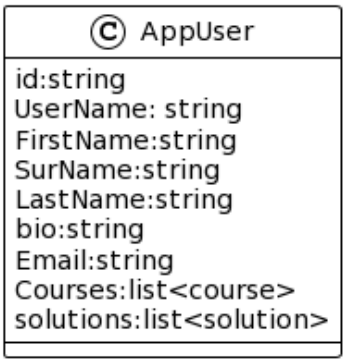

Рис. 4 – Клас користувача системи.

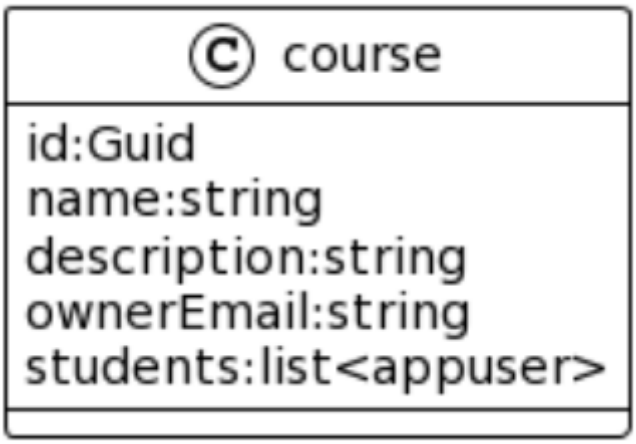

Рис. 5 – Клас курсу.

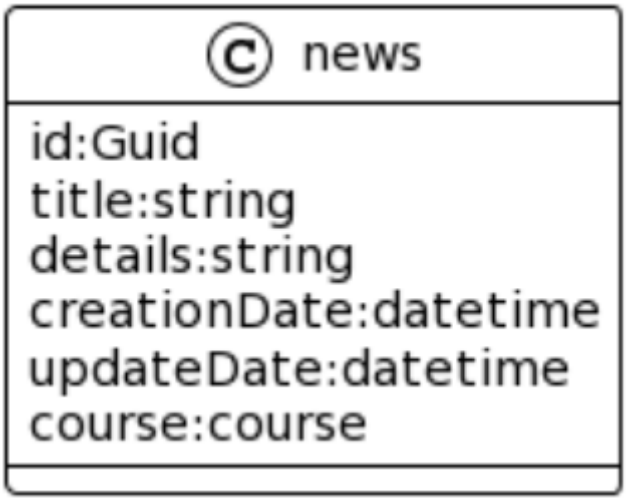

Рис. 6 – Клас новини.

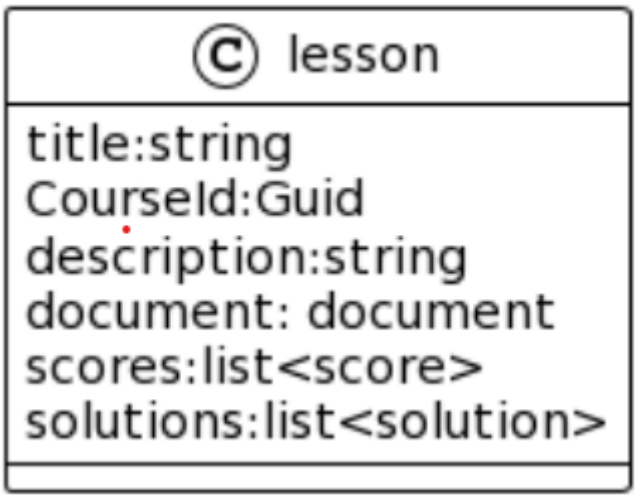

Рис. 7 – Клас уроку.

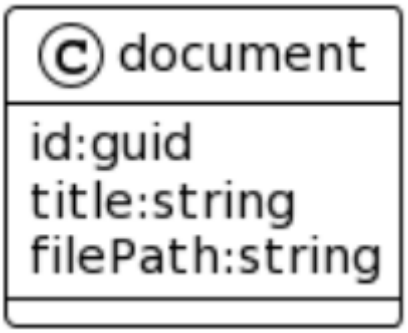

Рис. 8 – Клас файлу.

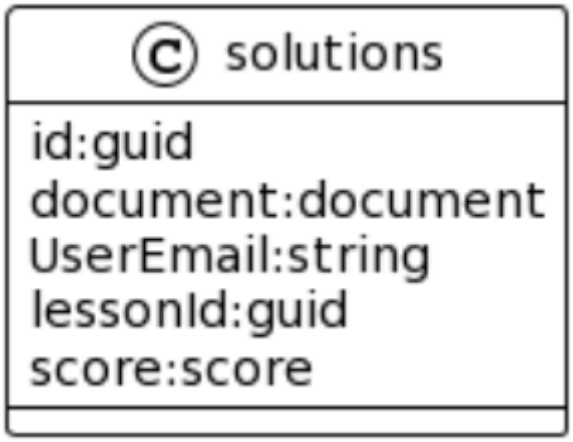

Рис. 9 – Клас рішення задачі.

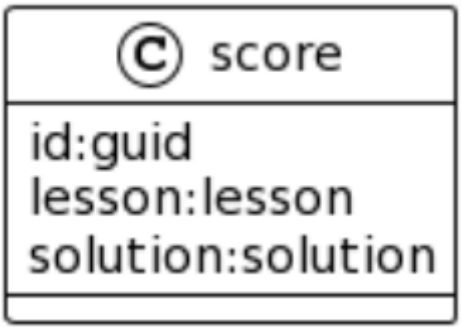

Рис. 10 – Клас оцінки.

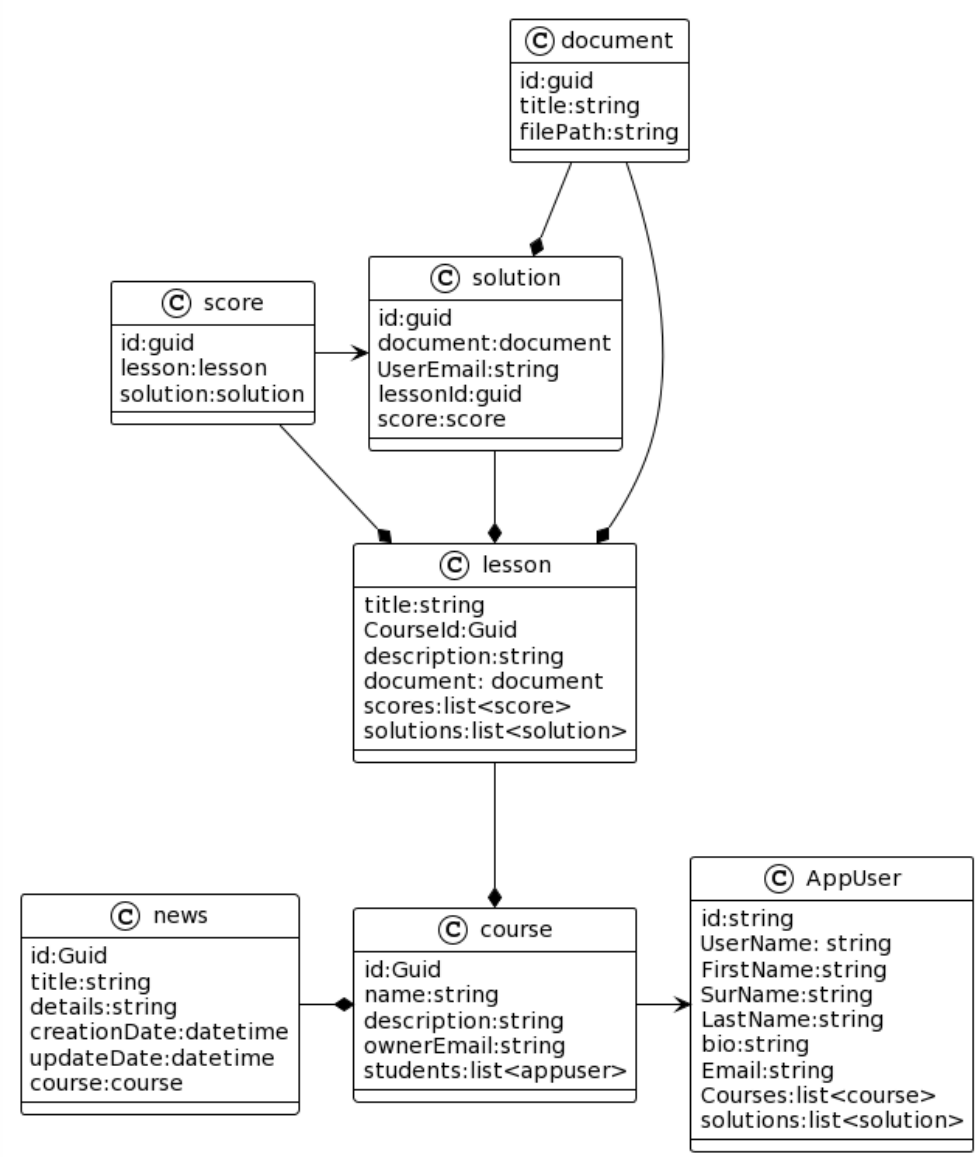

Рис. 11 – Діаграма відносин між сутностями.

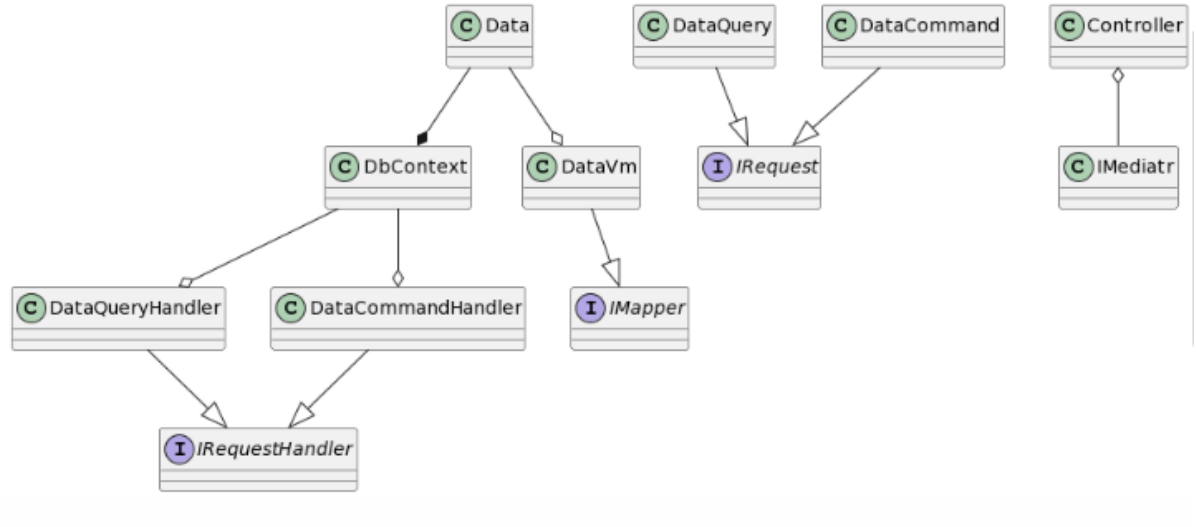

Рис. 12 – Діаграма відносин між сутністю і класами обробки даних.

3 КОНСТРУЮВАННЯ ОСВІТНЬОЇ ПЛАТФОРМИ

3.1 Реалізація ключових частин коду

Перш за все підключимо необхідні бібліотеки до проекту.

Для SchoolKy.Application

```
<ItemGroup>
    <PackageReference Include="AutoMapper" Version="12.0.1" />
    <PackageReference Include="FluentValidation" Version="11.5.2" />
    <PackageReference
Include="FluentValidation.DependencyInjectionExtensions" Version="11.5.2"
/>
    <PackageReference Include="MediatR" Version="12.0.1" />
    <PackageReference
Include="MediatR.Extensions.Microsoft.DependencyInjection" Version="11.1.0"
/>
    <PackageReference
Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.7"
/>
    <PackageReference Include="Microsoft.EntityFrameworkCore"
Version="7.0.7" />
  </ItemGroup>
```

Для SchoolKy.WebApi:

```
<ItemGroup>
    <PackageReference
Include="AutoMapper.Extensions.Microsoft.DependencyInjection"
Version="12.0.1" />
    <PackageReference Include="MailKit" Version="4.0.0" />
    <PackageReference
Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.7" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
Version="7.0.7">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="MimeKit" Version="4.0.0" />
  </ItemGroup>
```

Тепер коли ми встановили необхідні покати, перейдемо до написання основних класів.

Додамо у проект SchoolKy.Domain клас AppUser, який унаслідує Identityuser:

```
public class AppUser : IdentityUser
{
public string FirstName { get; set; }
public string LastName { get; set; }
public string SurName { get; set; }
public string? Bio { get; set; }
public List<Course> Courses { get; set; }
}
```

Додамо у проект SchoolKy.Domain клас Course

```
public class Course
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string OwnerEmail { get; set; }
        public List<AppUser> Users { get; set; }
    }
```

Додамо у проект SchoolKy.Persistence клас AppDbContext

```
public class AppDbContext : IdentityDbContext<AppUser>, IAppDbContext
    {
        public DbSet<Course> Courses { get; set; }


        public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            base.OnModelCreating(modelBuilder);

            modelBuilder.ApplyConfiguration(new CourseConfiguration());


        }

    }
```

Додамо у проект AccountService для керування профілем користувача

```
public class AccountService : IAccountService
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
```

```
        public AccountService(UserManager<AppUser> userManager,
SignInManager<AppUser> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }

        public async Task<ResultResponse>
ChangePasswordAsync(ChangePasswordRequest request, string userEmail)
        {
            var user = await _userManager.FindByNameAsync(userEmail);
            if(user == null)
            {
                return new ResultResponse
                {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
                };
            }
            var result = await _userManager.ChangePasswordAsync(user,
request.Password, request.NewPassword);
            string msg = "Errors: ";
            foreach(var error in result.Errors.Select(e =>
e.Description.ToString()))
            {
                msg = string.Concat(msg, error);
            }
            return result.Succeeded ? new ResultResponse { Code =
HttpStatusCode.OK, Message = "Password was changed" } :
                new ResultResponse { Code = HttpStatusCode.BadRequest,
Message = msg };
        }

        public async Task<ResultResponse>
UpdateProfileAsync(UpdateProfileRequest request, string userEmail)
        {
            if (!string.IsNullOrWhiteSpace(request.PhoneNumber))
            {
                var userWithSamePhone = await
_userManager.Users.FirstOrDefaultAsync(u => u.PhoneNumber ==
request.PhoneNumber);
                if(userWithSamePhone != null && userWithSamePhone.Email !=
userEmail)
                {
                    return new ResultResponse { Code =
HttpStatusCode.BadRequest, Message = string.Format("User with {0} phone
number already exists", request.PhoneNumber) };
                }
            }
            var user = await _userManager.FindByNameAsync(userEmail);
            if (user == null)
            {
                return new ResultResponse { Code = HttpStatusCode.NotFound,
Message = "User not found" };
            }

            user.FirstName = request.FirstName;
            user.LastName = request.LastName;
```

```
            user.SurName = request.SurName;
            user.Bio = request.Bio;

            var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
            if (request.PhoneNumber != phoneNumber)
            {
                var setPhoneResult = await
_userManager.SetPhoneNumberAsync(user, request.PhoneNumber);
            }

            var result = await _userManager.UpdateAsync(user);
            string msg = "Errors: ";
            foreach (var error in result.Errors.Select(e =>
e.Description.ToString()))
            {
                msg = string.Concat(msg, error);
            }
            await _signInManager.RefreshSignInAsync(user);
            return result.Succeeded ? new ResultResponse { Code =
HttpStatusCode.OK, Message = "Profile was updated" } :
                new ResultResponse { Code = HttpStatusCode.BadRequest,
Message = msg };
        }
    }
```

Додамо у проект JwtHandler для аутентифікації

```
public class JwtHandler
    {
        private readonly IConfiguration _configuration;
        private readonly UserManager<AppUser> _userManager;

        public JwtHandler(IConfiguration configuration,
UserManager<AppUser> userManager)
        {
            _configuration = configuration;
            _userManager = userManager;
        }

        public async Task<JwtSecurityToken> GetTokenAsync(AppUser user)
        {
            var jwtOptions = new JwtSecurityToken(
                issuer: _configuration["JwtSettings:Issuer"],
                audience: _configuration["JwtSettings:Audience"],
                claims: await GetClaimsAsync(user),
                expires:
DateTime.Now.AddMinutes(Convert.ToDouble(_configuration["JwtSettings:Expira
tionTimeInMinutes"])),
                signingCredentials: GetSigningCredentials());
            return jwtOptions;
        }

        private SigningCredentials GetSigningCredentials()
        {
            var key =
Encoding.UTF8.GetBytes(_configuration["JwtSettings:SecurityKey"]);
            var secret = new SymmetricSecurityKey(key);
```

```
            return new SigningCredentials(secret,
SecurityAlgorithms.HmacSha256);
        }

        private async Task<List<Claim>> GetClaimsAsync(AppUser user)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Email)
            };
            foreach(var role in await _userManager.GetRolesAsync(user))
            {
                claims.Add(new Claim(ClaimTypes.Role, role));
            }
            return claims;
        }
    }
```

Додамо у проект TokenService для аутентифікації

```
public class TokenService : ITokenService
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly JwtHandler _jwtHandler;

        public TokenService(UserManager<AppUser> userManager,
SignInManager<AppUser> signInManager, JwtHandler jwtHandler)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _jwtHandler = jwtHandler;
        }

        public async Task<ResultResponse<string>> LoginAsync(TokenRequest
request)
        {
            var user = await _userManager.FindByNameAsync(request.Email);
            if (user == null)
            {
                return new ResultResponse<string>
                {
                    Code = HttpStatusCode.NotFound,
                    Message = string.Concat($"User with {0} email not
found", request.Email)
                };
            }
            var passwordValid = await _userManager.CheckPasswordAsync(user,
request.Password);
            if (!passwordValid)
            {
                return new ResultResponse<string>
                {
                    Code = HttpStatusCode.BadRequest,
                    Message = "Invalid password"
                };
            }
```

```
            var secToken = await _jwtHandler.GetTokenAsync(user);
            var jwt = new JwtSecurityTokenHandler().WriteToken(secToken);
            return new ResultResponse<string>
            {
                Code = HttpStatusCode.OK,
                Message = "Login successful",
                Data = jwt
            };
        }
    }
```

Додамо у проект сервіс користувача

```
public class UserService : IUserService
    {
        private readonly UserManager<AppUser> _userManager;
        public UserService(UserManager<AppUser> userManager
            )
        {
            _userManager = userManager;
        }
        public async Task<ResultResponse<List<UserResponse>>> GetAllAsync()
        {
            var users = await _userManager.Users.ToListAsync();

            var result = new List<UserResponse>();
            foreach ( var user in users )
            {
                var res = new UserResponse
                {
                    Id = user.Id,
                    Email = user.Email,
                    FirstName = user.FirstName,
                    LastName = user.LastName,
                    SurName = user.SurName,
                    Bio = user.Bio,
                    PhoneNumber = user.PhoneNumber
                };
                result.Add(res);
            }
            var response = new ResultResponse<List<UserResponse>>()
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse<UserResponse>> GetByIdAsync(string
userId)
        {
            var user = await _userManager.FindByIdAsync(userId);
            if ( user == null )
            {
                return new ResultResponse<UserResponse> {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
```

```
                };
            }
            var result = new UserResponse
            {
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                SurName = user.SurName,
                Bio = user.Bio,
                PhoneNumber = user.PhoneNumber
            };
            var response = new ResultResponse<UserResponse>
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse<UserResponse>>
GetByEmailAsync(string userEmail)
        {
            var user = await _userManager.FindByNameAsync(userEmail);
            if (user == null)
            {
                return new ResultResponse<UserResponse>
                {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
                };
            }
            var result = new UserResponse
            {
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                SurName = user.SurName,
                Bio = user.Bio,
                PhoneNumber = user.PhoneNumber
            };
            var response = new ResultResponse<UserResponse>
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse> RegisterAsync(RegisterRequest
request)
        {
            if(!string.IsNullOrEmpty(request.PhoneNumber)) {
                var userWithSamePhone = await
_userManager.Users.FirstOrDefaultAsync(u => u.PhoneNumber ==
request.PhoneNumber);
                if(userWithSamePhone != null)
```

```
                {
                    return new ResultResponse { Code =
HttpStatusCode.BadRequest, Message = string.Format($"Phone number {0} is
already taken", request.PhoneNumber) };
                }
            }

            var userWithSameEmail = await
_userManager.FindByNameAsync(request.Email);
            if (userWithSameEmail == null)
            {
                var user = new AppUser
                {
                    Email = request.Email,
                    FirstName = request.FirstName,
                    LastName = request.LastName,
                    SurName = request.SurName,
                    UserName = request.Email,
                    PhoneNumber = request.PhoneNumber,
                    EmailConfirmed = true
                };

                var result = await _userManager.CreateAsync(user,
request.Password);

                if (result.Succeeded)
                {
                    return new ResultResponse
                    {
                        Code = HttpStatusCode.OK,
                        Message = string.Format("User {0} registered.!",
user.UserName)
                    };
                }
                else
                {
                    return new ResultResponse
                    {
                        Code = HttpStatusCode.BadRequest,
                        Message = result.Errors.ToString()
                    };
                }
            }
            else
            {
                return new ResultResponse { Code =
HttpStatusCode.BadRequest, Message = string.Format("Email {0} is already
taken", request.Email) };
            }
        }
    }

public class CreateCourseCommand : IRequest<Guid>
    {
        public string UserEmail { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
```

```
    }

public class CreateCourseCommandHandler :
IRequestHandler<CreateCourseCommand, Guid>
    {
        private readonly IAppDbContext _appDbContext;

        public CreateCourseCommandHandler(IAppDbContext appDbContext) =>
_appDbContext = appDbContext;

        public async Task<Guid> Handle(CreateCourseCommand request,
CancellationToken cancellationToken)
        {
            var course = new Course
            {
                OwnerEmail = request.UserEmail,
                Name = request.Name,
                Description = request.Description,
                Id = Guid.NewGuid()
            };

            await _appDbContext.Courses.AddAsync(course,
cancellationToken);
            await _appDbContext.SaveChangesAsync(cancellationToken);
            return course.Id;
        }
    }

public class CreateCourseCommandValidator :
AbstractValidator<CreateCourseCommand>
    {
        public CreateCourseCommandValidator() {
            RuleFor(createNewsCommand =>
createNewsCommand.Name).NotEmpty().MaximumLength(250);
            RuleFor(createNewsCommand =>
createNewsCommand.UserEmail).NotEqual(string.Empty);
        }
    }

public class DeleteCourseCommand : IRequest
    {
        public string UserEmail { get; set; }
        public Guid Id { get; set; }
    }

public class DeleteCourseCommandHandler :
IRequestHandler<DeleteCourseCommand>
    {
        private readonly IAppDbContext _appDbContext;

        public DeleteCourseCommandHandler(IAppDbContext appDbContext) =>
_appDbContext = appDbContext;

        public async Task Handle(DeleteCourseCommand command,
CancellationToken cancellationToken)
        {
```

```
            var entity = await _appDbContext.Courses.FindAsync(new object[]
{ command.Id }, cancellationToken);

            if(entity == null || entity.OwnerEmail != command.UserEmail)
            {
                throw new NotFoundException(nameof(News), command.Id);
            }

            _appDbContext.Courses.Remove(entity);
            await _appDbContext.SaveChangesAsync(cancellationToken);
        }
    }

public DeleteCourseCommandValidator() {
            RuleFor(deleteNewsCommand =>
deleteNewsCommand.Id).NotEqual(Guid.Empty);
            RuleFor(deleteNewsCommand =>
deleteNewsCommand.UserEmail).NotEqual(string.Empty);
        }

public class CourseDetailsVm : IMapWith<Course>
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string OwnerEmail { get; set; }
        public string OnwerFirstName { get; set; }
        public string OnwerLastName { get; set; }
        public string OnwerSurName { get; set; }
        public bool IsOwner { get; set; }
        public List<News> News { get; set; }

        public void Mapping(Profile profile)
        {
            profile.CreateMap<Course, CourseDetailsVm>()
                .ForMember(vm => vm.Name, opt => opt.MapFrom(c => c.Name))
                .ForMember(vm => vm.Description, opt => opt.MapFrom(c =>
c.Description))
                .ForMember(vm => vm.Id, opt => opt.MapFrom(c => c.Id))
                .ForMember(vm => vm.OwnerEmail, opt => opt.MapFrom(c =>
c.OwnerEmail))
                .ForMember(vm => vm.News, opt => opt.MapFrom(c => c.News));

        }
    }

public class GetCourseDetailsQueryHandler :
IRequestHandler<GetCourseDetailsQuery, CourseDetailsVm>
    {
        private readonly IAppDbContext _appDbContext;
        private readonly IMapper _mapper;

        public GetCourseDetailsQueryHandler(IAppDbContext appDbContext,
IMapper mapper) => (_appDbContext, _mapper) = (appDbContext, mapper);

        public async Task<CourseDetailsVm> Handle(GetCourseDetailsQuery
query, CancellationToken cancellationToken)
```

```
        {
            var entity = await _appDbContext.Courses.Include(n =>
n.News).FirstOrDefaultAsync(c => c.Id == query.Id, cancellationToken);

            if (entity == null) {
                throw new NotFoundException(nameof(Course), query.Id);
            }

            return _mapper.Map<CourseDetailsVm>(entity);
        }
    }
```

3.2 Розробка інтерфейсу користувача

Для розробки інтерфейсу виконаємо в консолі команду в локації проекту:

ng create app WorldCities

Додамо AppRoutingModule

```
import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { CitiesComponent } from './cities/cities.component';

import { CityEditComponent } from './cities/city-edit.component';

import { CountriesComponent } from './countries/countries.component';

import { CountryEditComponent } from './countries/country-edit.component';


const routes: Routes = [

  { path: '', component: HomeComponent, pathMatch: 'full' },

  { path: 'cities', component: CitiesComponent },

  { path: 'city/:id', component: CityEditComponent },
```

```
  { path: 'city', component: CityEditComponent },
  { path: 'countries', component: CountriesComponent },
  { path: 'country/:id', component: CountryEditComponent },
  { path: 'country', component: CountryEditComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Додамо AppModule

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { AngularMaterialModule } from './angular-material.module';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { CitiesComponent } from './cities/cities.component';
```

```
import { CountriesComponent } from './countries/countries.component';
import { CityEditComponent } from './cities/city-edit.component';
import { CountryEditComponent } from './countries/country-edit.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    NavMenuComponent,
    CitiesComponent,
    CountriesComponent,
    CityEditComponent,
    CountryEditComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    AngularMaterialModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Додамо AuthService

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, Subject, tap } from 'rxjs';

import { environment } from './../../environments/environment';
import { LoginRequest } from './login-request';
import { LoginResult } from './login-result';

@Injectable({
  providedIn: 'root',
})
export class AuthService {

  private tokenKey: string = "token";

  private _authStatus = new Subject<boolean>();
  public authStatus = this._authStatus.asObservable();

  constructor(
    protected http: HttpClient) {
  }

  isAuthenticated(): boolean {
    return this.getToken() !== null;
  }

  getToken(): string | null {
    return localStorage.getItem(this.tokenKey);
  }

  init(): void {
    if (this.isAuthenticated())
      this.setAuthStatus(true);
  }

  login(item: LoginRequest): Observable<LoginResult> {
    var url = environment.baseUrl + "api/Token/Login";
    return this.http.post<LoginResult>(url, item)
      .pipe(tap(loginResult => {
        if (loginResult.code == 200 && loginResult.data) {
          localStorage.setItem(this.tokenKey, loginResult.data);
          this.setAuthStatus(true);
        }
      }));
  }

  logout() {
    localStorage.removeItem(this.tokenKey);
    this.setAuthStatus(false);
  }

  private setAuthStatus(isAuthenticated: boolean): void {
    this._authStatus.next(isAuthenticated);
  }
}
```

Додамо LoginComponent

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { FormGroup, FormControl, Validators, AbstractControl, AsyncValidatorFn } from
'@angular/forms';
import { BaseFormComponent } from '../base-form.component';
import { AuthService } from './auth.service';
import { LoginRequest } from './login-request';
import { LoginResult } from './login-result';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})

export class LoginComponent
  extends BaseFormComponent implements OnInit {

  title?: string;
  loginResult?: LoginResult;

  constructor(
    private activatedRoute: ActivatedRoute,
    private router: Router,
    private authService: AuthService) {
    super();
  }
  ngOnInit() {
    this.form = new FormGroup({
      email: new FormControl('', Validators.required),
      password: new FormControl('', Validators.required)
    });
  }
  onSubmit() {
    var loginRequest = <LoginRequest>{};
    loginRequest.email = this.form.controls['email'].value;
    loginRequest.password = this.form.controls['password'].value;
    this.authService
      .login(loginRequest)
      .subscribe(result => {
        console.log(result);
        this.loginResult = result;
        if (result.code == 200) {
          this.router.navigate(["/"]);
        }
      }, error => {
        console.log(error);
        if (error.status == 401) {
          this.loginResult = error.error;
        }
      });
  }
}
</div>
```

3.3 Тестування програмного забезпечення

Юніт-тестування - це спосіб, який допомагає перевірити, чи працюють окремі модулі програмного забезпечення належним чином. Після перевірки різних модулів, вони можуть бути об'єднані і протестовані як єдине ціле або випущені виробництво.

З цим визначенням легко зрозуміти важливість правильного визначення та ізоляції різних блоків. Юніти - це найменші частини програмного забезпечення, які підлягають тестуванню. Вони мають кілька входів і один вихід. У об'єктно-орієнтованому програмуванні (ООП) блок часто є методом супер-, абстрактного або похідного класу, але може бути й статичною функцією допоміжного класу.

Незважаючи на те, що юніт-тести є стандартом для високоякісних проектів, багато розробників і менеджерів проектів недооцінюють їх важливість. Вони прагнуть прискорити процес розробки і зменшити його вартість. Однак, створення юніт-тестів разом з розробкою може бути викликом для невеликих проектів з низьким рівнем прибутку, оскільки це потребує додаткової роботи. Проте, вони мають велику перевагу для середніх і великих проектів та корпоративних рішень, особливо коли вимагається спільна робота великої кількості розробників. Ми реалізуємо внутрішні модульні тести в ASP.NET Core за допомогою інструменту тестування xUnit.net.

XUnit є однією з провідних бібліотек для автоматичного тестування в екосистемі .NET [6]. Вона надає широкий спектр функціональних можливостей для створення й виконання тестів одиниць коду, включаючи організацію тестових наборів, використання атрибутів для налаштування поведінки тестів, а також зручні механізми перевірки стверджень. Основна філософія XUnit базується на підходах "тест-факт" (Facts) та "тест-теорія" (Theories). Тест-факти використовуються для перевірки конкретних тверджень про поведінку коду. Тест-теорії дозволяють визначати параметри та передбачувані результати, на основі яких генеруються набори тестів для перевірки різних комбінацій вхідних значень. XUnit надає потужні засоби для організації тестових наборів та налаштування їх виконання. За допомогою атрибутів можна вказати, які методи відповідають за

запуск тестів, які конструктори мають бути викликані перед кожним тестом, а також які дії повинні відбутися перед і після виконання всього набору тестів. Помилки тестування та невиконані ствердження виводяться у зручному форматі, що полегшує виявлення та усунення проблем. XUnit також підтримує розширення зовнішніх інструментів для тестування та генерації звітів, таких як TestDriven.NET і ReSharper.

Додамо в проєкт необхідні пакети:

- Microsoft.NET.Test.Sdk ;
- xunit ;
- xunit.runner.visualstudio ;
- Moq ;
- Microsoft.EntityFrameworkCore.InMemory.

Виконаємо наступну команду в командному рядку:

dotnet new xunit -o WorldCitiesAPI.Tests.

Додамо AppContextFactory

```
public class AppContextFactory
    {
        public static Guid CourseAId = Guid.NewGuid();
        public static Guid CourseBId = Guid.NewGuid();
        public static string UserAEmail = "UserAEmail@gmail.com";
        public static string UserBEmail = "UserBEmail@gmail.com";

        public static Guid NewsIdForDelete = Guid.NewGuid();
        public static Guid NewsIdForUpdate = Guid.NewGuid();

        public static AppDbContext Create()
        {
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase(Guid.NewGuid().ToString())
                .Options;

            var context = new AppDbContext(options);

            context.Courses.AddRange(
                new Course
                {
                    Id = CourseAId,
                    Name = "name1",
                    Description = "descrption1",
                    OwnerEmail = UserAEmail
```

```
                },
                new Course
                {
                    Id = CourseBId,
                    Name = "name2",
                    Description = "descrption2",
                    OwnerEmail = UserBEmail
                });

            context.News.AddRange(
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details1",
                    EditDate = null,
                    Id = Guid.Parse("A6BB65BB-5AC2-4AFA-8A28-
2616F675B825"),
                    Title = "Title1",
                    CourseId = CourseAId,
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details2",
                    EditDate = null,
                    Id = Guid.Parse("909F7C29-891B-4BE1-8504-
21F84F262084"),
                    Title = "Title2",
                    CourseId = CourseBId,
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details3",
                    EditDate = null,
                    Id = NewsIdForDelete,
                    Title = "Title3",
                    CourseId = CourseAId,
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details4",
                    EditDate = null,
                    Id = NewsIdForUpdate,
                    Title = "Title4",
                    CourseId = CourseBId
                });
            context.SaveChanges();
            return context;
        }
        public static void Destroy(AppDbContext context)
        {
            context.Database.EnsureDeleted();
            context.Dispose();
        }
    }
```

```
public class QueryTestFixture : IDisposable
    {
        public AppDbContext Context;
        public IMapper Mapper;
        public QueryTestFixture()
        {
            Context = AppContextFactory.Create();
            var configrationProvider = new MapperConfiguration(cfg =>
            {
                cfg.AddProfile(new
AssemblyMappingProfile(typeof(IAppDbContext).Assembly));
            });
            Mapper = configrationProvider.CreateMapper();
        }
        public void Dispose()
        {
            AppContextFactory.Destroy(Context);
        }

        [CollectionDefinition("QueryCollection")]
        public class QueryCollection : ICollectionFixture<QueryTestFixture>
{ }
    }

public abstract class TestCommandBase : IDisposable
    {
        protected readonly AppDbContext Context;
        public TestCommandBase()
        {
            Context = AppContextFactory.Create();
        }

        public void Dispose()
        {
            AppContextFactory.Destroy(Context);
        }
    }
```

Створимо наш перший тест

```
public class CreateNewsCommandHandlerTests : TestCommandBase
    {
        [Fact]
        public async Task CreateNewsCommandHandler_Success()
        {
            //Arrange
            var handler = new CreateNewsCommandHandler(Context);
            var newsName = "news name";
            var newsDetails = "news details";

            //Act
            var newsId = await handler.Handle(
                new CreateNewsCommand
                {
                    Title = newsName,
                    Details = newsDetails,
                    CourseId = AppContextFactory.CourseAId,
```

```
                    UserEmail = AppContextFactory.UserAEmail
                }, CancellationToken.None);

            //Assert
            Assert.NotNull(await Context.News.SingleOrDefaultAsync(n =>
            n.Id == newsId && n.Title == newsName && n.Details ==
newsDetails));
        }
    }
```

На етапі Arrange ми встановлюємо необхідні ресурси для проведення тестування. На етапі Act ми перевіряємо поведінку пристрою. Зазвичай цей етап складається з однієї інструкції, яка відповідає за те, що ми хочемо перевірити. Метою фази Assert є перевірка того, чи задані умови правильно виконуються за допомогою значень, отриманих на етапі Act. Для цього ми використовуємо клас Assert, який надається xUnit. Цей клас містить різні статичні методи, які можна використовувати для перевірки виконання цих умов.

В даному випадку ми створили 8 юніт – тестів, 1 для команди створення новини, 3 для команди видалення новини , 3 для команди оновлення новини та 1 для запиту деталей новини.

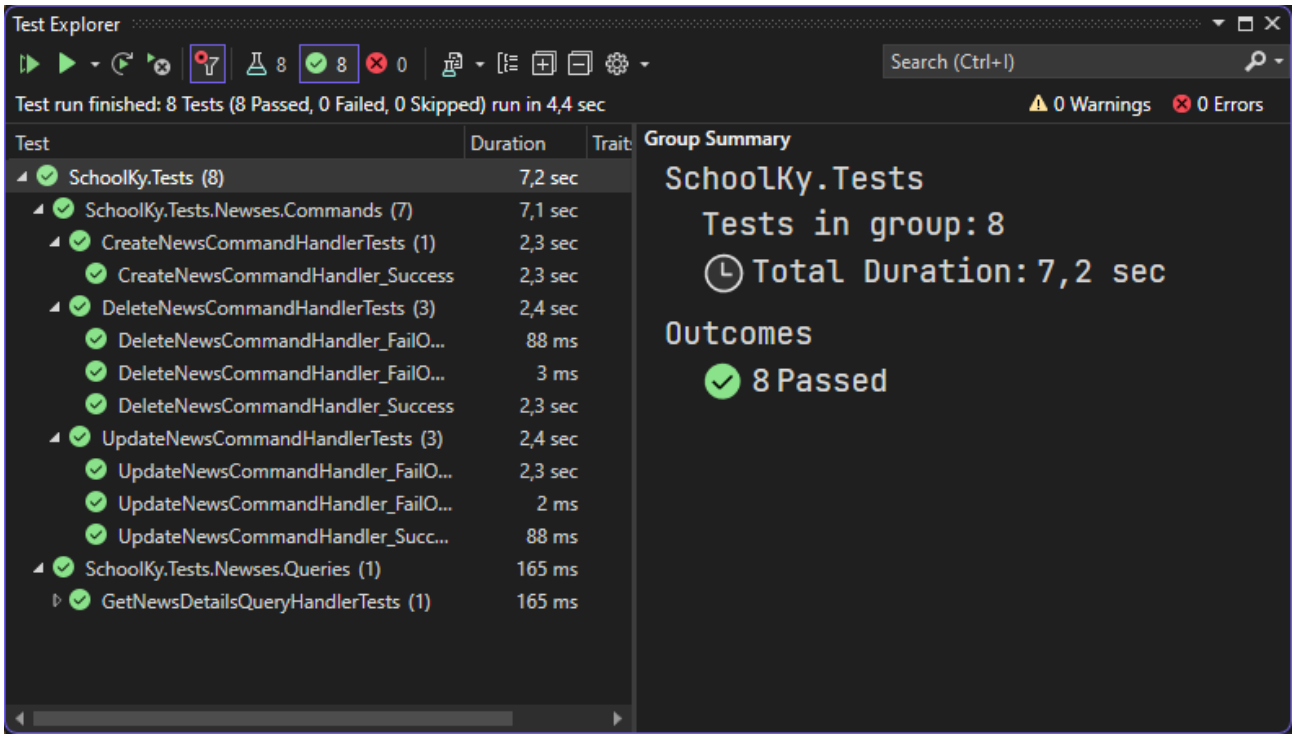

Рис. 13 – Результат виконання тестів

2.2.4 Результати розробки

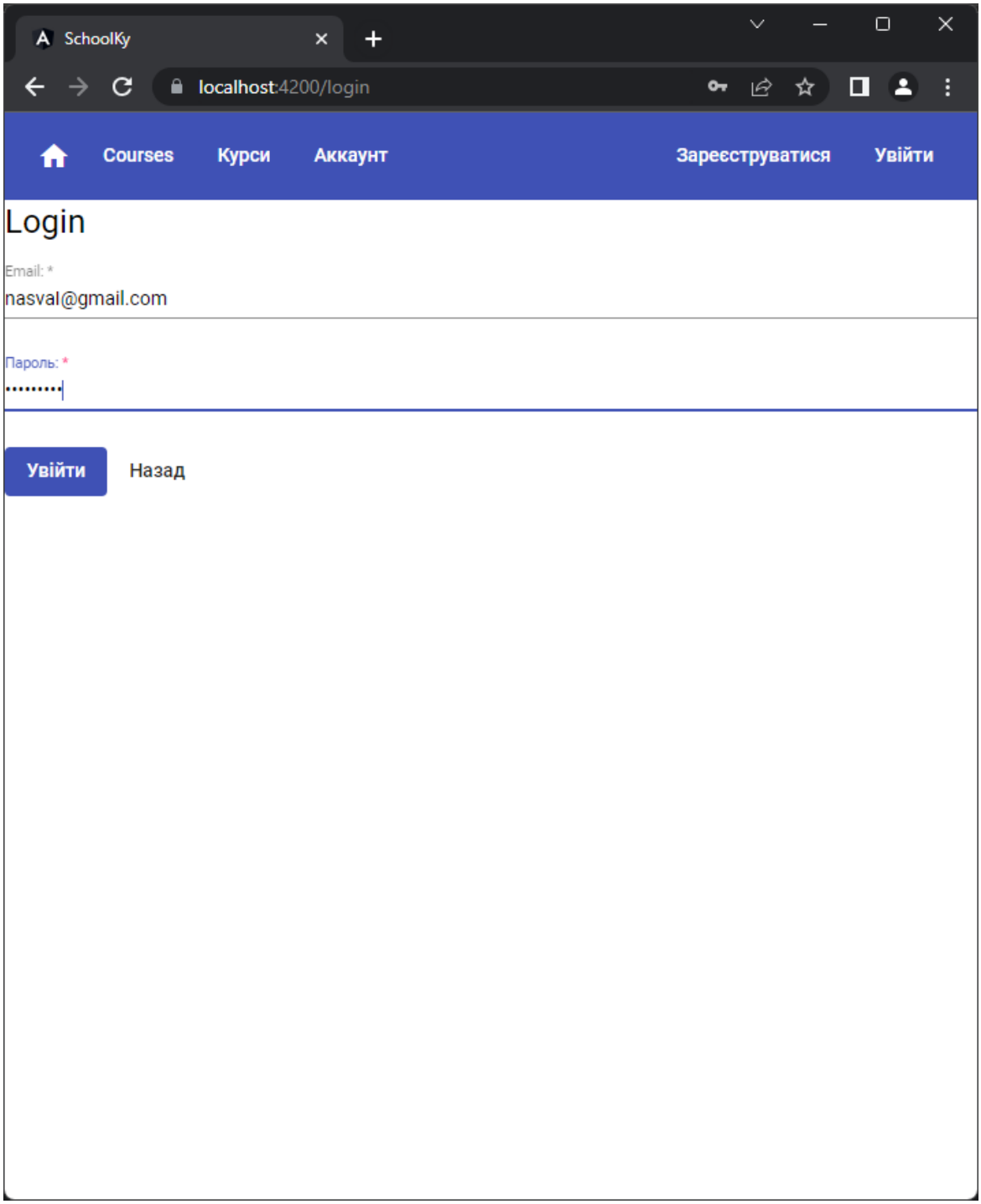

Рис. 14 – Сторінка логіну.

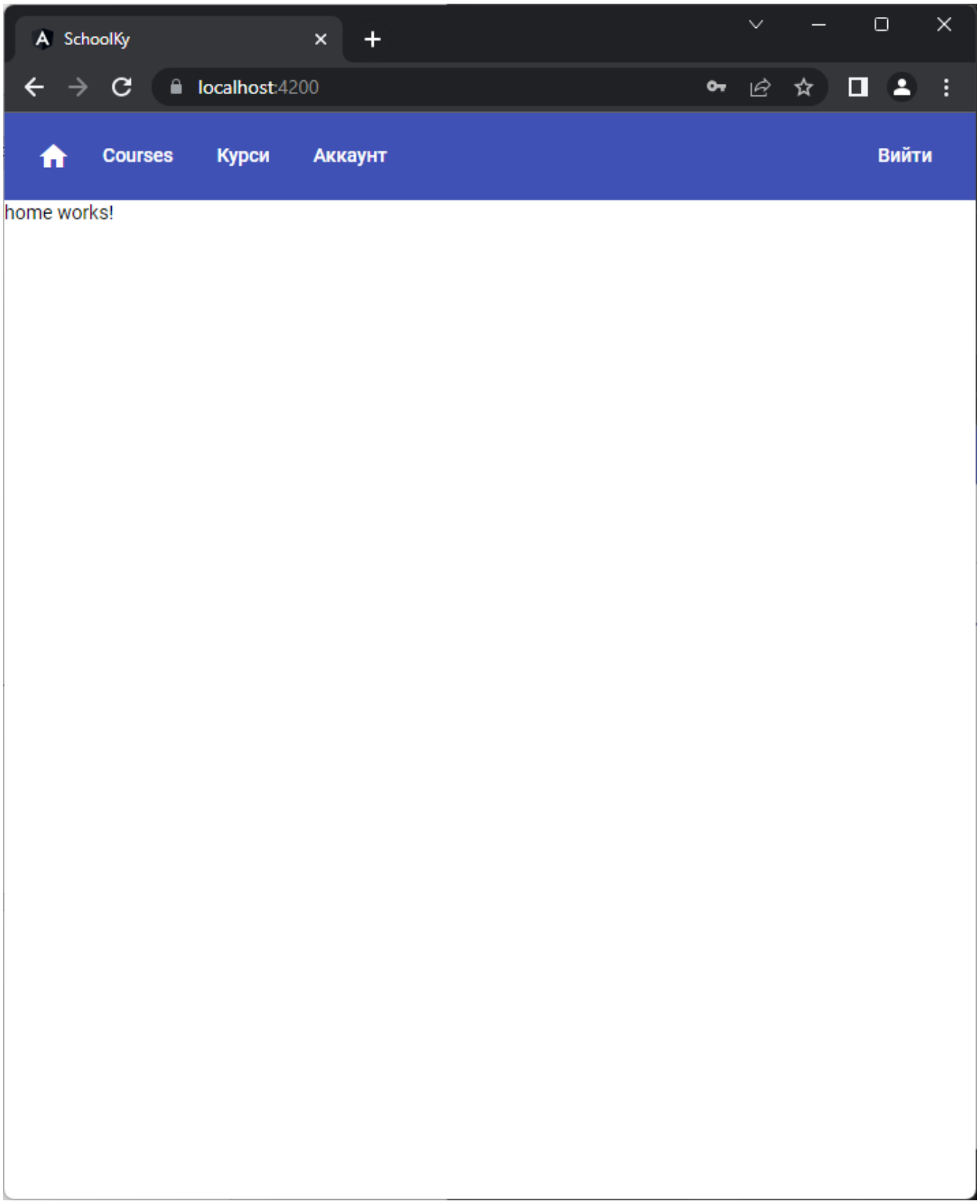

Рис. 15 – Головна сторінка.

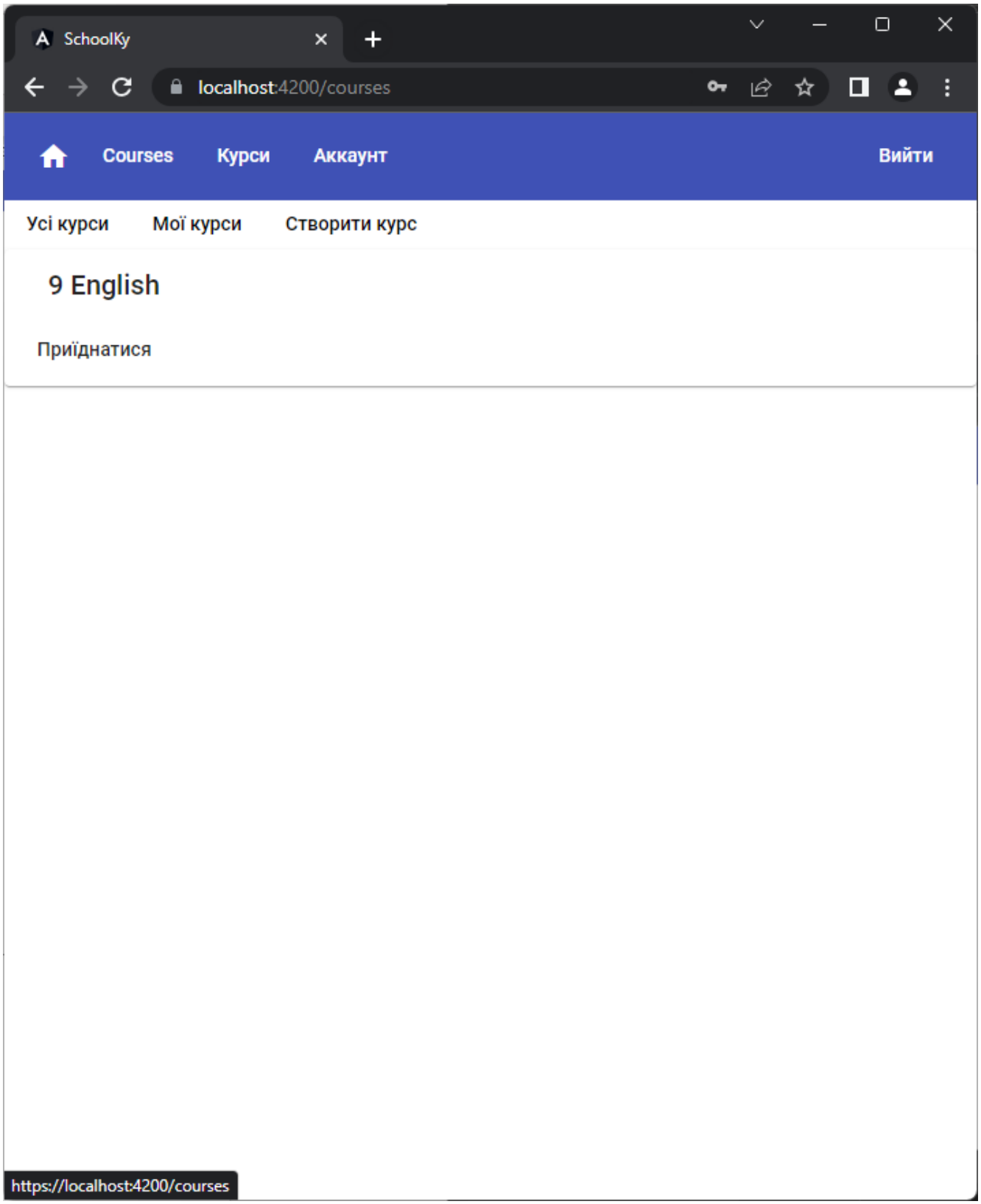

Рис. 16 – Сторінка курсів.

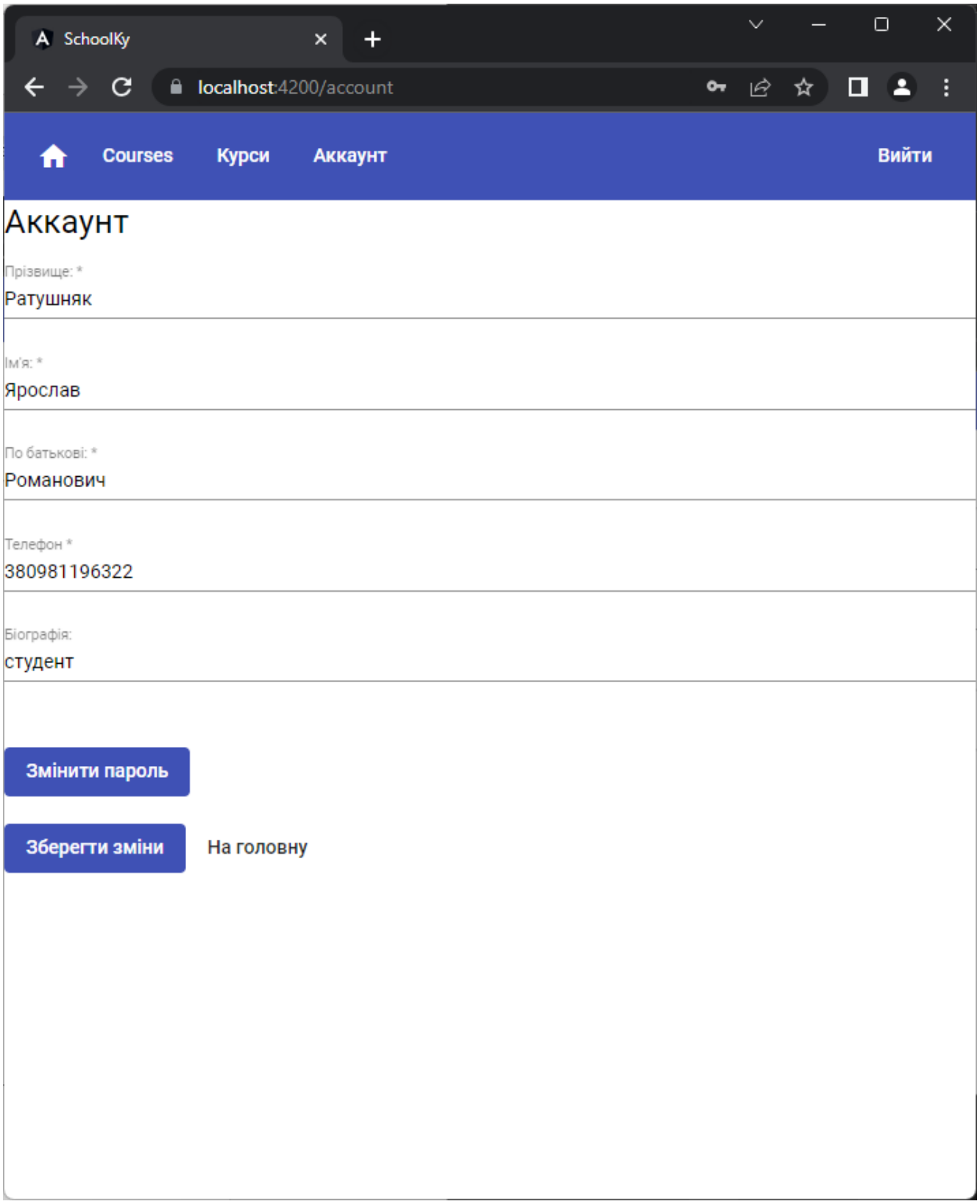

Рис. 17 – Сторінка редагування профіля.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Психофізіологічне розвантаження для працівників

За умови високого рівня робіт з ЕОМ, згідно ДСанПіН 3.3.2-007-98[7], рекомендується психофізіологічне розвантаження у спеціально обладнаних приміщеннях (кімнати психофізіологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

Психофізіологічне розвантаження для працівників - це набір методів і технік, спрямованих на зниження психологічного та фізіологічного напруження, що виникає в процесі виконання робочих завдань. Його основна мета - покращення загального благополуччя та здоров'я працівників.

Психологічне напруження, пов'язане з роботою, може виникати через велику кількість завдань, стресові ситуації, незадовільні умови праці або вимоги, конфлікти тощо. Фізіологічне напруження може включати сидячий спосіб життя, монотонну роботу, погану освітленість, шум, незручні робочі пози тощо. Це може призводити до виснаження, втому, стресу, погіршення психічного та фізичного здоров'я працівника. Психофізіологічне розвантаження має на меті забезпечити працівникам необхідний відпочинок та відновлення енергії, зниження рівня стресу та підвищення їх загального комфорту на робочому місці.

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов’язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ зі словесним самонавіюванням.

У рекомендованому сеансі, який має проводитися в кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим

оформленням, відділяються три періоди, що відповідають фазам відновлювального процесу.

Перший період - абстрагування працівників від виробничої обстановки - відповідає фазі залишкового збудження. Лунають повільна мелодійна музика, пташиний спів. Обравши зручну позу, працівники адаптуються і психологічно готуються до наступних періодів.

Другий - заспокоєння - відповідає фазі відновлювального гальмування. Пропонується показ фотослайдів із зображеннями квітучого лугу, березового гаю, гладенької поверхні ставка тощо. Через навушники транслюється спокійна музика, а на її фоні негучно, повільно висловлюються заспокійливі формули аутогенного тренування.

Як функціональне освітлення застосовують зелене світло. Яскравість світла має поступово знижуватись протягом періоду, а наприкінці його світло вимикається зовсім на одну - дві хвилини. Екран теж гасне.

Третій період - активізація - відповідає фазі підвищеної збудженості. На початку періоду світло вимкнене, через певний час на екрані з'являється червона пряма, розміри і яскравість якої поступово збільшуються. Наприкінці періоду лунає бадьора музика. Вимовляються тричі мобілізуючі формули аутогенного тренування, яким мають передувати глибоке вдихання та довге глибоке видихання.

Сеанси психологічного розвантаження можуть проводитись за єдиною програмою через індивідуальні навушники і складатись із двох періодів по 5 хвилин кожний:

- повне розслаблення;
- активізація працездатності.

У разі потреби на фоні музичних програм можуть вимовлятися окремі фрази навіювання відпочинку, гарного самопочуття і на заключному етапі - бадьорості.

Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являються бадьорість, гарний настрій. Загальний стан відчутно поліпшується.

4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора

Ергономічний дизайн робочого місця оператора комп'ютера є надзвичайно важливим, оскільки він безпосередньо впливає на здоров'я, комфорт і продуктивність працівника. Ергономіка - це наука, що вивчає взаємодію між людиною і її оточенням, тому розумний підхід до організації робочого простору є ключовим для забезпечення оптимальних умов праці. Основна мета ергономічного дизайну полягає у тому, щоб пристосувати робоче місце до потреб людини, зменшити навантаження на її фізичне і психологічне здоров'я і забезпечити максимальну продуктивність. Оптимально організоване робоче місце оператора комп'ютера має наступні важливі аспекти:

1) Стул та стіл: Комфортне сидіння та правильна підтримка спини допомагають уникнути напруги в м'язах і хребті, а також запобігають виникненню проблем зі спинним стовбуром. Стіл повинен бути відповідної висоти, щоб працівник міг зручно розмістити клавіатуру та мишку, забезпечуючи при цьому природні пози рук і запобігаючи зайвому напруженню.
2) Монітор: Екран монітора повинен бути розміщений на відстані від очей працівника, забезпечуючи оптимальну видимість і запобігаючи зайвому напруженню очей. Рекомендується також використовувати екрани з

антибліковим покриттям та налаштувати яскравість та контрастність відповідно до умов освітлення приміщення.

3) Клавіатура та мишка: Організація клавіатури та мишки повинна забезпечити природні пози рук і запобігати виникненню травм, таких як синдром карпального каналу. Клавіатура повинна бути розміщена на рівні нижньої частини передпліччя, а мишка - в зручному положенні, щоб уникнути надмірного напруження зап'ястя.
4) Освітлення та шум: Правильне освітлення приміщення дуже важливе для запобігання зморшкам, втрати зору та загальної втоми. Рекомендується використовувати природне або штучне освітлення, яке не викликає блискітання на екрані монітора. Крім того, важливо забезпечити низький рівень шуму, оскільки надмірний шум може заважати концентрації та підвищувати стрес працівника.
5) Паузи і розтяжки: Важливо нагадувати працівникам про необхідність регулярних пауз і фізичних вправ, щоб попередити затягнення та напруження м'язів. Короткі перерви можуть покращити кровообіг, зняти напругу та покращити фокусування.

Ергономічний дизайн робочого місця допомагає зменшити ризик травм, пов'язаних з неправильними рухами або позиціями тіла. Оптимальна організація робочого місця може уникнути надмірного напруження м'язів, дискомфорту або неприємностей, що можуть призвести до пошкодження м'язів, кісток або суглобів.

За врегулювання стандартів ергономіки робочого місця сидячи відповідає – ДСТУ 8604:2015 Дизайн і ергономіка [20]. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги.

Згідно вищезгаданого нормативного акту площа кабінету, в якому буде проходити робота повинна бути не менш 6 м², а об’єм не менш 24 м³. Для внутрішньої обробки приміщення повинні використовуватися дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі – 0,7-0,8; для стін – 0,5-0,6; для підлоги – 0,3-0,5.

Конструкція робочого столу повинна забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання. Конструкція крісла повинна забезпечувати підтримку раціональної робочої пози під час роботи з відео-дисплейним терміналом (Далі ВДТ) і ПЕОМ, дозволяти змінювати позу з метою зниження статичного напруження м'язів шийно-плечової області і спини для попередження розвитку втоми працюючого.

Поверхня сидіння, спинки та інших елементів стільця (крісла) повинна бути напівм'якою, з покриттям, що не електризується, неслизьке та повітронепроникне, що забезпечує легке очищення від забруднення.

Висота робочої поверхні столу, за відсутності можливості її регулювання повинна складати 725 мм. Робочий стіл повинен мати простір для ніг висотою не менше 600 мм, шириною – не менше 500 мм, не менше 450 мм в глибину на рівні колін і на рівні простягнутої ноги – не менше 650 мм. Робоче місце має бути обладнане підставкою для ніг, має ширину не менше 300 мм, глибину не менше 400 мм, регулювання по висоті в межах 150 мм за кутом нахилу опорної поверхні підставки до 20 градусів.

Відстань від очей користувача до екрану дисплея має становити 500-700 мм. Кут зору 10-20°, але не більше 40°; кут між верхнім краєм дисплея і рівнем очей користувача має становити не менше 10°. Кращим є розташування екрану перпендикулярно до лінії зору користувача. Робочі місця по відношенню до світлових прорізів повинні розташовуватися не ближче 3 м так, щоб природне світло падало збоку, переважно зліва. Освітленість також впливає на стан здоров'я і працездатність людини.

У відповідності зі ДБН В.2.5-28:2018[8] встановлені наступні вимоги до освітленості:

Для штучного освітлення:

- Комбіноване освітлення – освітленість 1500 лк;
- Загальне освітлення – освітленість 400 лк.

Для природного освітлення:

- Верхнє або комбіноване освітлення – коефіцієнт природної освітленості (далі КПО) 10%;
- Бічне освітлення – КПО 3.5%.

Для суміщеного освітлення:

- Верхнє або комбіноване освітлення – КПО 3-6%;
- Бічне освітлення – КПО 1.1-2%.

ВИСНОВКИ

У дипломні роботі описано процес розробки веб – застосунку з допомогою технологій ASP.NET Web Api та Angualar. Проведено аналіз предметної області. Були проаналізовані конкуренти, їх переваги та недоліки, які слугували прикладом для побудови освітньої платформи. Під час виконання кваліфікаційної роботи було досліджено методи та інструменти для розробки сучасних Spa застосунків.

Було сформовано модель системи, яка складалася з формування вимог, опису сутностей, діаграм використання та взаємодії класів. Це дозволило визначити шляхи взаємодії компонентів та функціональні потреби застосунку.

Наступним кроком була розробка додатку, реалізація класів та інтерфейсів, побудова інтерфейсу користувача. Також під час розробки були реалізовані юніт-тести. В результаті був розроблений застосунок, який забезпечує надійний та зручний спосіб для ведення навчального процесу онлайн. Було проведено тестування програми, перевірено роботу додатку та відповідність до визначених вимог.

В результаті виконання кваліфікаційної роботи бакалавра було розроблено та протестовано освітню платформу, побудовану на основі технології ASP.NET Core з допомогою фреймфорку Angular. Застосунок цілком відповідає вказаному завданню та сформованим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Нові Знання - Вхід на сайт - Електронні щоденники та журнали з можливостями дистанційного навчання [Електронний ресурс] – Режим доступу до ресурсу: https://nz.ua/.
2) Overview of ASP.NET Core [Електронний ресурс] / D. Roth, R. Anderson, S. Luttin. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0.
3) Angular (web framework) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Angular_(web_framework).
4) Методологія розробки програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Методологія_розробки_програмного_забезпечення.
5) CQRS pattern [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs.
6) Illich O. Мистецтво юніт-тестування в .NET [Електронний ресурс] / Oles Illich – Режим доступу до ресурсу: https://dou.ua/forums/topic/40477/.
7) Березюк О. В. Безпека життєдіяльності / О. В. Березюк, М. С. Лемешев. – Вінниця: ВНТУ, 2011.
8) Грибан В. Г. Охорона праці / В. Г. Грибан, О. В. Негодченко. – Київ: Центр учбової літератури, 2009.

ДОДАТКИ

ДОДАТОК А

Лістинг коду розробленої програми

```
var builder = WebApplication.CreateBuilder(args);


builder.Services.AddAutoMapper(config =>
{
    config.AddProfile(new AssemblyMappingProfile(Assembly.GetExecutingAssembly()));
    config.AddProfile(new AssemblyMappingProfile(typeof(IAppDbContext).Assembly));
});

builder.Services.AddApplication();
builder.Services.AddPersistence(builder.Configuration);

builder.Services.AddIdentity<AppUser, IdentityRole>(options =>
{
    options.SignIn.RequireConfirmedAccount = true;
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequiredLength = 8;
}).AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddAuthentication(opt =>
{
    opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        RequireExpirationTime = true,
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["JwtSettings:Issuer"],
        ValidAudience = builder.Configuration["JwtSettings:Audience"],
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtSettings:Security
Key"]))
    };
});

builder.Services.AddControllers().AddJsonOptions(options =>
{
    options.JsonSerializerOptions.WriteIndented = true;
});
builder.Services.AddEndpointsApiExplorer();

builder.Services.AddCors(options => {
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyHeader();
        policy.AllowAnyMethod();
        policy.AllowAnyOrigin();
    });
});
```

```
builder.Services.AddScoped<JwtHandler>();
builder.Services.AddScoped<IAccountService, AccountService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<ITokenService, TokenService>();
var app = builder.Build();

using (var scope = app.Services.CreateScope())
{
    var serviceProvider = scope.ServiceProvider;
    try
    {
        var context = serviceProvider.GetRequiredService<AppDbContext>();
        DbInitializer.Initialize(context);
    }
    catch (Exception exception)
    {
    }
}

app.UseCustomExceptionHandler();
app.UseRouting();
app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

app.UseCors("AllowAll");

app.MapControllers();

app.Run();

public class NewsController : BaseController
    {
        private readonly IMapper _mapper;

        public NewsController(IMapper mapper) => _mapper = mapper;

        [HttpGet]
        public async Task<ActionResult<NewsListVm>> GetAll(Guid id)
        {
            var query = new GetNewsListQuery
            {
                Id = id
            };
            var vm = await Mediator.Send(query);
            return Ok(vm);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<NewsDetailsVm>> Get(Guid id)
        {
            var query = new GetNewsDetailsQuery
            {
                UserEmail = UserEmail,
                Id = id
            };
            var vm = await Mediator.Send(query);
            return Ok(vm);
        }

        [HttpPost]
        public async Task<ActionResult<Guid>> Create([FromBody] CreateNewsDto
createNewsDto)
        {
            var command = _mapper.Map<CreateNewsCommand>(createNewsDto);
            command.UserEmail = UserEmail;
```

```
            var newsId = await Mediator.Send(command);
            return Ok(newsId);
        }

        [HttpPut]
        public async Task<IActionResult> Update([FromBody] UpdateNewsDto updateNewsDto)
        {
            var command = _mapper.Map<UpdateNewsCommand>(updateNewsDto);
            command.UserEmail = UserEmail;
            await Mediator.Send(command);
            return NoContent();
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(Guid id)
        {
            var command = new DeleteNewsCommand
            {
                Id = id,
                UserEmail = UserEmail
            };
            await Mediator.Send(command);
            return NoContent();
        }
    }

[Route("api/[controller]")]
    [ApiController]
    public class TokenController : ControllerBase
    {
        private readonly ITokenService _tokenService;

        public TokenController(ITokenService tokenService)
        {
            _tokenService = tokenService;
        }

        /// <summary>
        /// Get Token (Email, Password)
        /// </summary>
        /// <param name="model"></param>
        /// <returns>Status 200 OK</returns>
        [HttpPost("Login")]
        public async Task<ActionResult> Login(TokenRequest model)
        {
            var response = await _tokenService.LoginAsync(model);
            return Ok(response);
        }
    }

public class UserController : BaseController
    {
        private readonly IUserService _userService;

        public UserController(IUserService userService)
        {
            _userService = userService;
        }

        /// <summary>
        /// Get Users Details
        /// </summary>
        /// <returns>Status 200 OK</returns>
        [AllowAnonymous]
        [HttpGet("GetAll")]
        public async Task<IActionResult> GetAll()
        {
```

```
            var users = await _userService.GetAllAsync();
            return Ok(users);
        }

        /// <summary>
        /// Get User By Id
        /// </summary>
        /// <param name="id"></param>
        /// <returns>Status 200 OK</returns>
        [HttpGet("Get/{id}")]
        public async Task<IActionResult> GetById(string id)
        {
            var user = await _userService.GetByIdAsync(id);
            return Ok(user);
        }

        [Authorize]
        [HttpGet("GetProfile")]
        public async Task<IActionResult> GetProfile()
        {
            var user = await _userService.GetByEmailAsync(UserEmail);
            return Ok(user);
        }

        /// <summary>
        /// Register a User
        /// </summary>
        /// <param name="request"></param>
        /// <returns>Status 200 OK</returns>
        [AllowAnonymous]
        [HttpPost("Register")]
        public async Task<IActionResult> RegisterAsync(RegisterRequest request)
        {
            return Ok(await _userService.RegisterAsync(request));
        }
    }
[Route("api/[controller]")]
    public class CourseController : BaseController
    {
        private readonly IMapper _mapper;
        public CourseController(IMapper mapper)
        {
            _mapper = mapper;
        }

        [HttpGet("GetAll")]
        public async Task<ActionResult<CourseListVm>> GetAll()
        {
            var query = new GetCourseListQuery();
            var vm = await Mediator.Send(query);
            return Ok(vm);
        }
        [HttpGet("GetUserAll")]
        public async Task<ActionResult<UserCourseListVm>> GetUserAll()
        {
            var query = new GetUserCourseListQuery()
            {
                UserEmail = this.UserEmail
            };
            var vm = await Mediator.Send(query);
            return Ok(vm);
        }

        [HttpGet("Get/{id}")]
        public async Task<ActionResult<CourseDetailsVm>> Get(Guid id)
        {
            var query = new GetCourseDetailsQuery
```

```
            {
                UserEmail = this.UserEmail,
                Id = id
            };
            var vm = await Mediator.Send(query);
            return Ok(vm);
        }

        [HttpPost("Create")]
        public async Task<ActionResult<Guid>> Create(CreateCourseDto createCourseDto)
        {
            var command = _mapper.Map<CreateCourseCommand>(createCourseDto);
            command.UserEmail = UserEmail;
            var courseId = await Mediator.Send(command);
            return Ok(courseId);
        }

        [HttpPut]
        public async Task<IActionResult> Update(UpdateCourseDto updateCourseDto)
        {
            var command = _mapper.Map<UpdateCourseCommand>(updateCourseDto);
            command.OwnerEmail = UserEmail;
            await Mediator.Send(command);
            return NoContent();
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(Guid id)
        {
            var command = new DeleteCourseCommand
            {
                Id = id,
                UserEmail = UserEmail
            };
            await Mediator.Send(command);
            return NoContent();
        }
    }
[ApiController]
    [Route("api/[controller]/[action]")]
    public abstract class BaseController : ControllerBase
    {
        private IMediator _mediator;
        protected IMediator Mediator => _mediator ??=
HttpContext.RequestServices.GetService<IMediator>();
        internal string UserEmail => !User.Identity.IsAuthenticated ? string.Empty :
User.FindFirst(ClaimTypes.Name).Value;
    }

public class AccountController : BaseController
    {
        private readonly IAccountService _accountService;

        public AccountController(IAccountService accountService)
        {
            _accountService = accountService;
        }

        /// <summary>
        /// Update Profile
        /// </summary>
        /// <param name="model"></param>
        /// <returns>Status 200 OK</returns>
        [HttpPut("UpdateProfile")]
        public async Task<ActionResult> UpdateProfile(UpdateProfileRequest model)
        {
            var response = await _accountService.UpdateProfileAsync(model, UserEmail);
```

```
            return Ok(response);
        }

        /// <summary>
        /// Change Password
        /// </summary>
        /// <param name="model"></param>
        /// <returns>Status 200 OK</returns>
        [HttpPut("ChangePassword")]
        public async Task<ActionResult> ChangePassword(ChangePasswordRequest model)
        {
            var response = await _accountService.ChangePasswordAsync(model, UserEmail);
            return Ok(response);
        }
    }

public class AccountService : IAccountService
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;

        public AccountService(UserManager<AppUser> userManager, SignInManager<AppUser>
signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }

        public async Task<ResultResponse> ChangePasswordAsync(ChangePasswordRequest
request, string userEmail)
        {
            var user = await _userManager.FindByNameAsync(userEmail);
            if(user == null)
            {
                return new ResultResponse
                {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
                };
            }
            var result = await _userManager.ChangePasswordAsync(user, request.Password,
request.NewPassword);
            string msg = "Errors: ";
            foreach(var error in result.Errors.Select(e => e.Description.ToString()))
            {
                msg = string.Concat(msg, error);
            }
            return result.Succeeded ? new ResultResponse { Code = HttpStatusCode.OK,
Message = "Password was changed" } :
                new ResultResponse { Code = HttpStatusCode.BadRequest, Message = msg };
        }

        public async Task<ResultResponse> UpdateProfileAsync(UpdateProfileRequest
request, string userEmail)
        {
            if (!string.IsNullOrWhiteSpace(request.PhoneNumber))
            {
                var userWithSamePhone = await _userManager.Users.FirstOrDefaultAsync(u
=> u.PhoneNumber == request.PhoneNumber);
                if(userWithSamePhone != null && userWithSamePhone.Email != userEmail)
                {
                    return new ResultResponse { Code = HttpStatusCode.BadRequest,
Message = string.Format("User with {0} phone number already exists",
request.PhoneNumber) };
                }
            }
            var user = await _userManager.FindByNameAsync(userEmail);
```

```
            if (user == null)
            {
                return new ResultResponse { Code = HttpStatusCode.NotFound, Message =
"User not found" };
            }

            user.FirstName = request.FirstName;
            user.LastName = request.LastName;
            user.SurName = request.SurName;
            user.Bio = request.Bio;

            var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
            if (request.PhoneNumber != phoneNumber)
            {
                var setPhoneResult = await _userManager.SetPhoneNumberAsync(user,
request.PhoneNumber);
            }

            var result = await _userManager.UpdateAsync(user);
            string msg = "Errors: ";
            foreach (var error in result.Errors.Select(e => e.Description.ToString()))
            {
                msg = string.Concat(msg, error);
            }
            await _signInManager.RefreshSignInAsync(user);
            return result.Succeeded ? new ResultResponse { Code = HttpStatusCode.OK,
Message = "Profile was updated" } :
                new ResultResponse { Code = HttpStatusCode.BadRequest, Message = msg };
        }
    }
public class JwtHandler
    {
        private readonly IConfiguration _configuration;
        private readonly UserManager<AppUser> _userManager;

        public JwtHandler(IConfiguration configuration, UserManager<AppUser>
userManager)
        {
            _configuration = configuration;
            _userManager = userManager;
        }

        public async Task<JwtSecurityToken> GetTokenAsync(AppUser user)
        {
            var jwtOptions = new JwtSecurityToken(
                issuer: _configuration["JwtSettings:Issuer"],
                audience: _configuration["JwtSettings:Audience"],
                claims: await GetClaimsAsync(user),
                expires:
DateTime.Now.AddMinutes(Convert.ToDouble(_configuration["JwtSettings:ExpirationTimeInMi
nutes"])),
                signingCredentials: GetSigningCredentials());
            return jwtOptions;
        }

        private SigningCredentials GetSigningCredentials()
        {
            var key =
Encoding.UTF8.GetBytes(_configuration["JwtSettings:SecurityKey"]);
            var secret = new SymmetricSecurityKey(key);
            return new SigningCredentials(secret, SecurityAlgorithms.HmacSha256);
        }

        private async Task<List<Claim>> GetClaimsAsync(AppUser user)
        {
            var claims = new List<Claim>
            {
```

```
                new Claim(ClaimTypes.Name, user.Email)
            };
            foreach(var role in await _userManager.GetRolesAsync(user))
            {
                claims.Add(new Claim(ClaimTypes.Role, role));
            }
            return claims;
        }
    }

public class TokenService : ITokenService
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly JwtHandler _jwtHandler;

        public TokenService(UserManager<AppUser> userManager, SignInManager<AppUser>
signInManager, JwtHandler jwtHandler)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _jwtHandler = jwtHandler;
        }

        public async Task<ResultResponse<string>> LoginAsync(TokenRequest request)
        {
            var user = await _userManager.FindByNameAsync(request.Email);
            if (user == null)
            {
                return new ResultResponse<string>
                {
                    Code = HttpStatusCode.NotFound,
                    Message = string.Concat($"User with {0} email not found",
request.Email)
                };
            }
            var passwordValid = await _userManager.CheckPasswordAsync(user,
request.Password);
            if (!passwordValid)
            {
                return new ResultResponse<string>
                {
                    Code = HttpStatusCode.BadRequest,
                    Message = "Invalid password"
                };
            }
            var secToken = await _jwtHandler.GetTokenAsync(user);
            var jwt = new JwtSecurityTokenHandler().WriteToken(secToken);
            return new ResultResponse<string>
            {
                Code = HttpStatusCode.OK,
                Message = "Login successful",
                Data = jwt
            };
        }
    }

public class UserService : IUserService
    {
        private readonly UserManager<AppUser> _userManager;
        public UserService(UserManager<AppUser> userManager
            )
        {
            _userManager = userManager;
        }
        public async Task<ResultResponse<List<UserResponse>>> GetAllAsync()
```

```
        {
            var users = await _userManager.Users.ToListAsync();

            var result = new List<UserResponse>();
            foreach ( var user in users )
            {
                var res = new UserResponse
                {
                    Id = user.Id,
                    Email = user.Email,
                    FirstName = user.FirstName,
                    LastName = user.LastName,
                    SurName = user.SurName,
                    Bio = user.Bio,
                    PhoneNumber = user.PhoneNumber
                };
                result.Add(res);
            }
            var response = new ResultResponse<List<UserResponse>>()
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse<UserResponse>> GetByIdAsync(string userId)
        {
            var user = await _userManager.FindByIdAsync(userId);
            if ( user == null )
            {
                return new ResultResponse<UserResponse> {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
                };
            }
            var result = new UserResponse
            {
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                SurName = user.SurName,
                Bio = user.Bio,
                PhoneNumber = user.PhoneNumber
            };
            var response = new ResultResponse<UserResponse>
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse<UserResponse>> GetByEmailAsync(string
userEmail)
        {
            var user = await _userManager.FindByNameAsync(userEmail);
            if (user == null)
            {
                return new ResultResponse<UserResponse>
                {
                    Code = HttpStatusCode.NotFound,
                    Message = "User not found"
                };
            }
            var result = new UserResponse
```

```
            {
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                SurName = user.SurName,
                Bio = user.Bio,
                PhoneNumber = user.PhoneNumber
            };
            var response = new ResultResponse<UserResponse>
            {
                Code = HttpStatusCode.OK,
                Data = result
            };
            return response;
        }

        public async Task<ResultResponse> RegisterAsync(RegisterRequest request)
        {
            if(!string.IsNullOrEmpty(request.PhoneNumber)) {
                var userWithSamePhone = await _userManager.Users.FirstOrDefaultAsync(u
=> u.PhoneNumber == request.PhoneNumber);
                if(userWithSamePhone != null)
                {
                    return new ResultResponse { Code = HttpStatusCode.BadRequest,
Message = string.Format($"Phone number {0} is already taken", request.PhoneNumber) };
                }
            }

            var userWithSameEmail = await _userManager.FindByNameAsync(request.Email);
            if (userWithSameEmail == null)
            {
                var user = new AppUser
                {
                    Email = request.Email,
                    FirstName = request.FirstName,
                    LastName = request.LastName,
                    SurName = request.SurName,
                    UserName = request.Email,
                    PhoneNumber = request.PhoneNumber,
                    EmailConfirmed = true
                };

                var result = await _userManager.CreateAsync(user, request.Password);

                if (result.Succeeded)
                {
                    return new ResultResponse
                    {
                        Code = HttpStatusCode.OK,
                        Message = string.Format("User {0} registered.!", user.UserName)
                    };
                }
                else
                {
                    return new ResultResponse
                    {
                        Code = HttpStatusCode.BadRequest,
                        Message = result.Errors.ToString()
                    };
                }
            }
            else
            {
                return new ResultResponse { Code = HttpStatusCode.BadRequest, Message =
string.Format("Email {0} is already taken", request.Email) };
            }
```

```
        }
    }
```

```
{
  "DbConnection": "Server=DESKTOP-NV482SM\\KWOTEFOX;Database=SchoolKy;User
Id=kwotefox;Password=nimda4321;Integrated
Security=False;MultipleActiveResultSets=True;TrustServerCertificate=True",
  "MailConfiguration": {
    "From": "info@gmail.com",
    "Host": "smtp.ethereal.email",
    "Port": 587,
    "UserName": "adaline.pfkfjg@ethereal.email",
    "Password": "jdsdjg831",
    "DisplayName": "Kwotefox"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "JwtSettings": {
    "SecurityKey": "MyVeryOwnSecurityKeyNimda4321",
    "Issuer": "SchoolKy",
    "Audience": "https://localhost:4200",
    "ExpirationTimeInMinutes": 60
  }
}
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SchoolKy.Application.Interfaces;
using SchoolKy.Domain;
using SchoolKy.Persistence.EntityTypeConfigurations;

namespace SchoolKy.Persistence
{
    public class AppDbContext : IdentityDbContext<AppUser>, IAppDbContext
    {
        public DbSet<Course> Courses { get; set; }
        public DbSet<News> News { get; set; }

        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            base.OnModelCreating(modelBuilder);
            modelBuilder.ApplyConfiguration(new NewsConfiguration());
            modelBuilder.ApplyConfiguration(new CourseConfiguration());

            string ADMIN_ID = "A6BB65BB-5AC2-4AFA-8A28-2616F675B821";
            string ROLE_ID = "A6BB65BB-5AC2-4AFA-8A28-2616F675B823";

            //seed admin role
            modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
            {
                Name = "SuperAdmin",
                NormalizedName = "SUPERADMIN",
                Id = ROLE_ID,
                ConcurrencyStamp = ROLE_ID
            });

            //create user
            var appUser = new AppUser
```

```
            {
                Id = ADMIN_ID,
                Email = "schoolkyadmin@gmail.com",
                EmailConfirmed = true,
                FirstName = "Олександр",
                SurName = "Андрійович",
                LastName = "Ковальчук",
                UserName = "schoolkyadmin@gmail.com",
                Bio = "Адмін платформи",
                NormalizedUserName = "SCHOOLKYADMIN@GMAIL.COM"
            };

            //set user password
            PasswordHasher<AppUser> ph = new PasswordHasher<AppUser>();
            appUser.PasswordHash = ph.HashPassword(appUser, "nimda4321");

            //seed user
            modelBuilder.Entity<AppUser>().HasData(appUser);

            //set user role to admin
            modelBuilder.Entity<IdentityUserRole<string>>().HasData(new
IdentityUserRole<string>
            {
                RoleId = ROLE_ID,
                UserId = ADMIN_ID
            });

            modelBuilder.Entity<Course>().HasData(new Course
            {
                Id = Guid.NewGuid(),
                Name = "9 English",
                Description = "English course for 9 class",
                OwnerEmail = appUser.Email,
            });
        }

    }
}
public class CreateCourseCommand : IRequest<Guid>
    {
        public string UserEmail { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }
public class CreateCourseCommandHandler : IRequestHandler<CreateCourseCommand, Guid>
    {
        private readonly IAppDbContext _appDbContext;

        public CreateCourseCommandHandler(IAppDbContext appDbContext) => _appDbContext
= appDbContext;

        public async Task<Guid> Handle(CreateCourseCommand request, CancellationToken
cancellationToken)
        {
            var course = new Course
            {
                OwnerEmail = request.UserEmail,
                Name = request.Name,
                Description = request.Description,
                Id = Guid.NewGuid()
            };

            await _appDbContext.Courses.AddAsync(course, cancellationToken);
            await _appDbContext.SaveChangesAsync(cancellationToken);
            return course.Id;
        }
    }
```

```
public class CreateCourseCommandValidator : AbstractValidator<CreateCourseCommand>
    {
        public CreateCourseCommandValidator() {
            RuleFor(createNewsCommand =>
createNewsCommand.Name).NotEmpty().MaximumLength(250);
            RuleFor(createNewsCommand =>
createNewsCommand.UserEmail).NotEqual(string.Empty);
        }
    }
public class DeleteCourseCommand : IRequest
    {
        public string UserEmail { get; set; }
        public Guid Id { get; set; }
    }

public class DeleteCourseCommandHandler : IRequestHandler<DeleteCourseCommand>
    {
        private readonly IAppDbContext _appDbContext;

        public DeleteCourseCommandHandler(IAppDbContext appDbContext) => _appDbContext
= appDbContext;

        public async Task Handle(DeleteCourseCommand command, CancellationToken
cancellationToken)
        {
            var entity = await _appDbContext.Courses.FindAsync(new object[] {
command.Id }, cancellationToken);

            if(entity == null || entity.OwnerEmail != command.UserEmail)
            {
                throw new NotFoundException(nameof(News), command.Id);
            }

            _appDbContext.Courses.Remove(entity);
            await _appDbContext.SaveChangesAsync(cancellationToken);
        }
    }

public class DeleteCourseCommandValidator : AbstractValidator<DeleteCourseCommand>
    {
        public DeleteCourseCommandValidator() {
            RuleFor(deleteNewsCommand => deleteNewsCommand.Id).NotEqual(Guid.Empty);
            RuleFor(deleteNewsCommand =>
deleteNewsCommand.UserEmail).NotEqual(string.Empty);
        }
    }
public class UpdateCourseCommand : IRequest
    {
        public Guid Id { get; set; }
        public string OwnerEmail { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }

public class UpdateCourseCommandHandler : IRequestHandler<UpdateCourseCommand>
    {
        private readonly IAppDbContext _appDbContext;
        public UpdateCourseCommandHandler(IAppDbContext appDbContext) => _appDbContext
= appDbContext;

        public async Task Handle(UpdateCourseCommand command, CancellationToken
cancellationToken) {
            var entity = await _appDbContext.Courses.FirstOrDefaultAsync(c =>  c.Id ==
command.Id, cancellationToken);

            if(entity == null || entity.OwnerEmail != command.OwnerEmail) {
```

```
                throw new NotFoundException(nameof(News), command.Id);
            }

            entity.Description = command.Description;
            entity.Name = command.Name;

            await _appDbContext.SaveChangesAsync(cancellationToken);

            return;
        }
    }

public class UpdateCourseCommandValidator : AbstractValidator<UpdateCourseCommand>
    {
        public UpdateCourseCommandValidator()
        {
            RuleFor(updateNewsCommand => updateNewsCommand.Id).NotEqual(Guid.Empty);
            RuleFor(updateNewsCommand =>
updateNewsCommand.OwnerEmail).NotEqual(string.Empty);
            RuleFor(updateNewsCommand =>
updateNewsCommand.Name).NotEmpty().MaximumLength(250);
        }
    }

public class AppContextFactory
    {
        public static Guid CourseAId = Guid.NewGuid();
        public static Guid CourseBId = Guid.NewGuid();
        public static string UserAEmail = "UserAEmail@gmail.com";
        public static string UserBEmail = "UserBEmail@gmail.com";

        public static Guid NewsIdForDelete = Guid.NewGuid();
        public static Guid NewsIdForUpdate = Guid.NewGuid();

        public static AppDbContext Create()
        {
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase(Guid.NewGuid().ToString())
                .Options;

            var context = new AppDbContext(options);

            context.Courses.AddRange(
                new Course
                {
                    Id = CourseAId,
                    Name = "name1",
                    Description = "descrption1",
                    OwnerEmail = UserAEmail
                },
                new Course
                {
                    Id = CourseBId,
                    Name = "name2",
                    Description = "descrption2",
                    OwnerEmail = UserBEmail
                });

            context.News.AddRange(
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details1",
                    EditDate = null,
                    Id = Guid.Parse("A6BB65BB-5AC2-4AFA-8A28-2616F675B825"),
                    Title = "Title1",
                    CourseId = CourseAId,
```

```
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details2",
                    EditDate = null,
                    Id = Guid.Parse("909F7C29-891B-4BE1-8504-21F84F262084"),
                    Title = "Title2",
                    CourseId = CourseBId,
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details3",
                    EditDate = null,
                    Id = NewsIdForDelete,
                    Title = "Title3",
                    CourseId = CourseAId,
                },
                new News
                {
                    CreationDate = DateTime.Today,
                    Details = "Details4",
                    EditDate = null,
                    Id = NewsIdForUpdate,
                    Title = "Title4",
                    CourseId = CourseBId
                });
            context.SaveChanges();
            return context;
        }
        public static void Destroy(AppDbContext context)
        {
            context.Database.EnsureDeleted();
            context.Dispose();
        }
    }

public class CreateNewsCommandHandlerTests : TestCommandBase
    {
        [Fact]
        public async Task CreateNewsCommandHandler_Success()
        {
            //Arrange
            var handler = new CreateNewsCommandHandler(Context);
            var newsName = "news name";
            var newsDetails = "news details";

            //Act
            var newsId = await handler.Handle(
                new CreateNewsCommand
                {
                    Title = newsName,
                    Details = newsDetails,
                    CourseId = AppContextFactory.CourseAId,
                    UserEmail = AppContextFactory.UserAEmail
                }, CancellationToken.None);

            //Assert
            Assert.NotNull(await Context.News.SingleOrDefaultAsync(n =>
            n.Id == newsId && n.Title == newsName && n.Details == newsDetails));
        }
    }

public class DeleteNewsCommandHandlerTests : TestCommandBase
    {
        [Fact]
```

```
        public async Task DeleteNewsCommandHandler_Success()
        {
            // Arrange
            var handler = new DeleteNewsCommandHandler(Context);

            // Act
            await handler.Handle(new DeleteNewsCommand
            {
                Id = AppContextFactory.NewsIdForDelete,
                UserEmail = AppContextFactory.UserAEmail
            }, CancellationToken.None);

            // Assert
            Assert.Null(Context.News.SingleOrDefault(note =>
                note.Id == AppContextFactory.NewsIdForDelete));
        }

        [Fact]
        public async Task DeleteNewsCommandHandler_FailOnWrongId()
        {
            // Arrange
            var handler = new DeleteNewsCommandHandler(Context);

            // Act
            // Assert
            await Assert.ThrowsAsync<NotFoundException>(async () =>
                await handler.Handle(
                    new DeleteNewsCommand
                    {
                        Id = Guid.NewGuid(),
                        UserEmail = AppContextFactory.UserAEmail
                    },
                    CancellationToken.None));
        }

        [Fact]
        public async Task DeleteNewsCommandHandler_FailOnWrongUserId()
        {
            // Arrange
            var deleteHandler = new DeleteNewsCommandHandler(Context);
            var createHandler = new CreateNewsCommandHandler(Context);
            var newsId = await createHandler.Handle(
                new CreateNewsCommand
                {
                    Title = "NewsTitle",
                    Details = "Details",
                    UserEmail = AppContextFactory.UserAEmail,
                    CourseId = AppContextFactory.CourseAId
                }, CancellationToken.None);

            // Act

            // Assert
            await Assert.ThrowsAsync<NotFoundException>(async () =>
                await deleteHandler.Handle(
                    new DeleteNewsCommand
                    {
                        Id = newsId,
                        UserEmail = AppContextFactory.UserBEmail
                    }, CancellationToken.None));
        }
    }

public class UpdateNewsCommandHandlerTests : TestCommandBase
    {
        [Fact]
        public async Task UpdateNewsCommandHandler_Success()
```

```
        {
            // Arrange
            var handler = new UpdateNewsCommandHandler(Context);
            var updatedTitle = "new title";

            // Act
            await handler.Handle(new UpdateNewsCommand
            {
                Id = AppContextFactory.NewsIdForUpdate,
                UserEmail = AppContextFactory.UserBEmail,
                Title = updatedTitle
            }, CancellationToken.None);

            // Assert
            Assert.NotNull(await Context.News.SingleOrDefaultAsync(note =>
                note.Id == AppContextFactory.NewsIdForUpdate &&
                note.Title == updatedTitle));
        }

        [Fact]
        public async Task UpdateNewsCommandHandler_FailOnWrongId()
        {
            // Arrange
            var handler = new UpdateNewsCommandHandler(Context);

            // Act
            // Assert
            await Assert.ThrowsAsync<NotFoundException>(async () =>
                await handler.Handle(
                    new UpdateNewsCommand
                    {
                        Id = Guid.NewGuid(),
                        UserEmail = AppContextFactory.UserAEmail
                    },
                    CancellationToken.None));
        }

        [Fact]
        public async Task UpdateNewsCommandHandler_FailOnWrongUserId()
        {
            // Arrange
            var handler = new UpdateNewsCommandHandler(Context);

            // Act
            // Assert
            await Assert.ThrowsAsync<NotFoundException>(async () =>
            {
                await handler.Handle(
                    new UpdateNewsCommand
                    {
                        Id = AppContextFactory.NewsIdForUpdate,
                        UserEmail = AppContextFactory.UserAEmail
                    },
                    CancellationToken.None);
            });
        }
    }
```

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    NavMenuComponent,
    CoursesComponent,
```

```
    LoginComponent,
    RegisterComponent,
    AccountEditComponent,
    ChangePasswordComponent,
    CourseComponent,
    EditCourseComponent,
    CreateCourseComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    AngularMaterialModule,
    ReactiveFormsModule
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

import { Component, OnInit } from '@angular/core';
import { AuthService } from './auth/auth.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  title = 'SchoolKy';
  constructor(private authService: AuthService) { }
  ngOnInit(): void {
    this.authService.init();
  }
}


const routes: Routes = [
  { path: '', component: HomeComponent, pathMatch: 'full' },
  { path: 'courses', component: CoursesComponent },
  { path: 'login', component: LoginComponent },
  { path: 'registration', component: RegisterComponent },
  { path: 'account', component: AccountEditComponent },
  { path: 'change-password', component: ChangePasswordComponent },
  { path: 'edit-course/:id', component: EditCourseComponent },
  { path: 'create-course', component: CreateCourseComponent },

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

@NgModule({
  imports: [
    MatButtonModule,
    MatIconModule,
    MatToolbarModule,
    MatTableModule,
```

```
    MatPaginatorModule,
    MatSortModule,
    MatInputModule,
    MatSelectModule,
    MatCardModule
  ],
  exports: [
    MatButtonModule,
    MatIconModule,
    MatToolbarModule,
    MatTableModule,
    MatPaginatorModule,
    MatSortModule,
    MatInputModule,
    MatSelectModule,
    MatCardModule
  ]
})
export class AngularMaterialModule { }
```

```
import { Injectable } from '@angular/core';
import {
  HttpInterceptor, HttpRequest, HttpHandler, HttpEvent,
  HttpErrorResponse
} from '@angular/common/http';
import { Router } from '@angular/router';
import { catchError, Observable, throwError } from 'rxjs';
import { AuthService } from './auth.service';
@Injectable({
  providedIn: 'root'
})
export class AuthInterceptor implements HttpInterceptor {
  constructor(
    private authService: AuthService,
    private router: Router) { }
  intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    // get the auth token
    var token = this.authService.getToken();
    // if the token is present, clone the request
    // replacing the original headers with the authorization
    if (token) {
      req = req.clone({
        setHeaders: {
          Authorization: `Bearer ${token}`
        }
      });
    }
    // send the request to the next handler
    return next.handle(req).pipe(
      catchError((error) => {
        // Perform logout on 401 - Unauthorized HTTP response errors
        if (error instanceof HttpErrorResponse && error.status === 401) {
          this.authService.logout();
          this.router.navigate(['login']);
        }
        return throwError(error);
      })
    );
  }
}
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, Subject, tap } from 'rxjs';
```

```
import { environment } from './../../environments/environment';
import { LoginRequest } from './login-request';
import { LoginResult } from './login-result';

@Injectable({
  providedIn: 'root',
})
export class AuthService {

  private tokenKey: string = "token";

  private _authStatus = new Subject<boolean>();
  public authStatus = this._authStatus.asObservable();

  constructor(
    protected http: HttpClient) {
  }

  isAuthenticated(): boolean {
    return this.getToken() !== null;
  }

  getToken(): string | null {
    return localStorage.getItem(this.tokenKey);
  }

  init(): void {
    if (this.isAuthenticated())
      this.setAuthStatus(true);
  }

  login(item: LoginRequest): Observable<LoginResult> {
    var url = environment.baseUrl + "api/Token/Login";
    return this.http.post<LoginResult>(url, item)
      .pipe(tap(loginResult => {
        if (loginResult.code == 200 && loginResult.data) {
          localStorage.setItem(this.tokenKey, loginResult.data);
          this.setAuthStatus(true);
        }
      }));
  }

  logout() {
    localStorage.removeItem(this.tokenKey);
    this.setAuthStatus(false);
  }

  private setAuthStatus(isAuthenticated: boolean): void {
    this._authStatus.next(isAuthenticated);
  }
}
<div class="login">
  <h1>Login</h1>
  <form [formGroup]="form" (ngSubmit)="onSubmit()">
    <p>
      <mat-error *ngIf="loginResult && loginResult.code != 200">
        <strong>ERROR</strong>: {{loginResult.message}}
      </mat-error>
    </p>
    <mat-form-field>
      <mat-label>Email:</mat-label>
      <input matInput formControlName="email" required
             placeholder="Введіть e-мейл">
      <mat-error *ngFor="let error of getErrors(form.get('email')!,
 'Email')">
        {{error}}
```

```
      </mat-error>
    </mat-form-field>
    <mat-form-field>
      <mat-label>Пароль:</mat-label>
      <input matInput type="password" formControlName="password" required
             placeholder="Введіть пароль">
      <mat-error *ngFor="let error of getErrors(form.get('password')!,
 'Password')">
        {{error}}
      </mat-error>
    </mat-form-field>
    <div>
      <button mat-flat-button color="primary"
              type="submit">
        Увійти
      </button>
      <button mat-flat-button color="secondary"
              [routerLink]="['/']">
        Назад
      </button>
    </div>
  </form>
</div>
```

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { FormGroup, FormControl, Validators, AbstractControl, AsyncValidatorFn } from
'@angular/forms';
import { BaseFormComponent } from '../base-form.component';
import { AuthService } from './auth.service';
import { LoginRequest } from './login-request';
import { LoginResult } from './login-result';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})

export class LoginComponent
  extends BaseFormComponent implements OnInit {

  title?: string;
  loginResult?: LoginResult;

  constructor(
    private activatedRoute: ActivatedRoute,
    private router: Router,
    private authService: AuthService) {
    super();
  }
  ngOnInit() {
    this.form = new FormGroup({
      email: new FormControl('', Validators.required),
      password: new FormControl('', Validators.required)
    });
  }
  onSubmit() {
    var loginRequest = <LoginRequest>{};
    loginRequest.email = this.form.controls['email'].value;
    loginRequest.password = this.form.controls['password'].value;
    this.authService
      .login(loginRequest)
      .subscribe(result => {
        console.log(result);
        this.loginResult = result;
```

```
        if (result.code == 200) {
          this.router.navigate(["/"]);
        }
      }, error => {
        console.log(error);
        if (error.status == 401) {
          this.loginResult = error.error;
        }
      });
  }
}
```

ДОДАТОК Б

Диск з розробленою програмою