Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Інфокомунікаціїї*
(повна назва)

Кафедра *Інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *перший (бакалаврський)*

Інтерактивний університетський веб-форум: реалізація та безпека
(тема)

Виконав:
здобувач 4 року навчання,
групи *ТРІМІ-21-2*
Данило Мелешко
(власне ім'я та прізвище)

Спеціальності *172 Інформаційно-мережної інженерії*
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Інформаційно-мережна інженерія*
(повна назва освітньої програми)
Керівник *ст. викл. Олексій Федоров*
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ІМІ _____ *Валерій Безрук*
(підпис) (власне ім'я та прізвище)

2025

Не містить відомостей, заборонених до відкритого публікування

Студент _____ *Мелешко Д.Д.*
(підпис) (прізвище та ініціали)

Керівник _____ *Федоров О.В.*
(підпис) (прізвище та ініціали)

Харківський національний університет радіоелектроніки

Факультет *Інфокомунікацій*
(повна назва)

Кафедра *Інформаційно-мережної інженерії*
(повна назва)

Рівень вищої освіти *перший (бакалаврський)*

Спеціальність *172 Телекомунікації та радіотехніка*
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Інформаційно-мережна інженерія*
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ІМІ_____
(підпис)
“____” _____2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві *Мелешко Данилові Дмитровичу*
(прізвище, ім’я, по батькові)

1. Тема роботи *Інтерактивний університетський веб-форум: реалізація та безпека*

затверджені наказом університету від *23 травня* 2025 року № *410 Ст*

2. Термін подання здобувачем роботи до екзаменаційної комісії *23 червня 2025 р.*

3. Вихідні дані до роботи

Технології які використовувати в проєкті: Asp.Net Core, React, HTML, CSS
Мови програмування: C#, TypeScript
Модель бази даних - Реляційна база даних.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ
1. Технології та Frameworks проєкту
2. Розробка та розгортання проєкту
3. Аналіз результатів та можливі покращення
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) *Слайди у форматі Power Point (назва, мета роботи, постановка задачі, технології проєкту, розробка веб-сайту, розробка серверної сторони, розробка клієнтської сторони, розгортання веб-сайту, висновок)*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ*	*10.11.24*	*виконано*
2	*Підбір літератури за темою роботи*	*20.11-26.11.24*	*виконано*
3	*Виконання розділу 1*	*14.03-23.03.25*	*виконано*
4	*Виконання розділу 2*	*26.03-18.04.25*	*виконано*
5	*Виконання розділу 3*	*20.04-22.04.25*	*виконано*
6	*Оформлення пояснювальної записки*	*09.05-10.05.25*	*виконано*
7	*Оформлення презентаційного матеріалу*	*25.05-27.05.25*	*виконано*
8	*Підготовка до захисту у ЕК*	*30.05-18.06.25*	*виконано*

Дата видачі завдання *10.11.2024 р.*

Здобувач _____ *Данило Мелешко*
(підпис) (власне ім'я та прізвище)

Керівник роботи _____ *ст. викл. Олексій Федоров*
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 127 ст., 78 рис., 0 табл., 28 джерел, 2 додаток.

Мета роботи – створити та проаналізувати процес розробки веб-форуму “NureHub” та розглянути варіанти оптимізації проєкту. Виходячи з цього, задачами є: описати технології, що застосовуються для розроби веб-форуму, створити серверну та клієнтську частини проєкту, налаштування політики CORS (Cross-Origin Resource Sharing), створення ER (Entity Relationship) моделі бази даних, створення та налаштування авторизації й аутентифікації користувачів, створити ролі з різними правами доступу, забезпечення безпеки даних користувачів, створити необхідний функціонал для роботи форуму, розгорнути проєкту в інтернет, надати частину коду проєкту, розглянути можливі вдосконалення проєкту.

В результаті проведеної роботи, було описано процес створення веб-форуму, описано технологій, створено серверну та клієнтську частини проєкту, налаштовано політики CORS, створено ER модель бази даних, створено та налаштовано авторизацію й аутентифікацію користувачів, створені ролі для викладачів, старост та звичайних студентів, з різними правами доступу, забезпечена безпека даних користувачів, створено необхідний функціонал для роботи форуму, розгорнуто проєкт в інтернет, надано код проєкту та проаналізовані можливі шляхи вдосконалення проєкту.

ABSTRACT

Explanatory note: 127 p., 78 fig., 0 tabl., 28 sources, 2 app.

The purpose of the work is to create and analyze the process of developing the NureHub web forum and consider options for optimizing the project. Based on this, the tasks are: to describe the technologies used to develop a web forum, create the server and client parts of the project, configure the CORS (Cross-Origin Resource Sharing) policy, create an ER (Entity Relationship) database model, create and configure user authorization and authentication, create roles with different access rights, ensure the security of user data, create the necessary functionality for the forum, deploy the project on the Internet, provide part of the project code, and consider possible project improvements.

As a result of the work done, the process of creating a web forum was described, technologies were described, the server and client parts of the project were created, CORS policies were configured, an ER database model was created, user authorization and authentication were created and configured, roles were created for teachers, elders and ordinary students with different access rights, user data security was ensured, the necessary functionality for the forum was created, the project was deployed on the Internet, the project code was provided and possible ways to improve the project were analyzed.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CSS (Cascading Style Sheets або каскадні таблиці стилів) – це мова візуального представлення вмісту HTML-документа;

HTML (HyperText Markup Language) – це спеціальна мова розмітки, яка застосовується при створенні сайтів в інтернеті;

SQL (Structured Query Language) – це стандартна мова запитів, яка використовується для роботи з реляційними базами даних;

ANC (Asp.Net Core) – це фреймворк розробки Http based застосунків, який використовує C# в якості мови програмування;

БД (База даних) - це організована колекція інформації, яка допомагає зберігати, записувати, читати, змінювати та видаляти дані в комп'ютерній системі;

NoSql (Not Only Sql) – це вид баз даних які на відміну від реляційних, зберігають інформацію у не структурованому і більш гнучкому форматі та інформація в них представлена у вигляді документів;

ORM (Object-Relational Mapping) – це техніка програмування, за допомогою якої, можна працювати з БД через об'єкти у конді;

CIL (Common Intermediate Language) – це низькорівневий код, який використовується .Net, як проміжний вихідним та машинним кодом;

JIT (Just-In-Time) – це технологія компіляції, яка компілює CIL в машинний код;

DTO (Data Transfer Object) – це шаблон програму програмування, який використовується для передачі даних між частинами програми;

CORS (Cross-Origin Resource Sharing) – механізм безпеки в браузері, який дозволяє або забороняє міждоменні HTTP-запити;

SDK (Software Development Kit) – це набір інструментів розробки (набір стандартних бібліотк) програмного забезпечення;

API (Application Programming Interface) – це тип програмного інтерфейсу, який забезпечує з’єднання між комп’ютерними програмами та пропонує послуги іншим програмним засобам;

URL (Uniform Resource Locator) – це стрічка, яка вказує на розміщення ресурсу в інтернеті;

ВМ (Віртуальна Машина) - модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації;

SaaS (Software as a service) – це модель хмарних обчислень, де постачальник пропонує клієнту користування прикладним програмним забезпеченням та керує всіма необхідними фізичними та програмними ресурсами;

PaaS (Platform as a service) – це модель хмарних обчислювальних послуг, де користувачі надають, створюють екземпляри, запускають та керують модульним пакетом обчислювальної платформи та застосунків без складності побудови та підтримки інфраструктури, пов'язаної з розробкою та запуском застосунків;

IaaS (Infrastructure as a service) – це модель хмарних обчислень, де постачальник хмарних послуг надає обчислювальні ресурси, такі як сховище, мережа, сервери та віртуалізація;

Jwt (JSON Web Token) – це стандарт для авторизації та аутентифікації даних користувача, який створює токен для доступу до системи, а також зберігає дані у форматі JSON.

ВСТУП

Вже давно ні в кого не залишилось сумнівів в ролі інтернету та інтернет застосунків в нашому житті. Майже кожна людина проводить близько половини свого часу, або більше, дивлячись щось на ютубі, обговорюючи якусь ситуацію з друзями чи іншими людьми в інтернеті, шукаючи корисну інформацію, або працюючи.

Інтеграція інтернету в життя майже кожної людини на землі вже давно закінчилось, що означає те, що життя більшості опирається на існування інтернету. Проте необхідно пам’ятати, незважаючи на кількість можливостей і інформації яку він надає, наша інтегрованість в нього, робить нас вразливими для тих хто має можливість отримувати наші дані, або маніпулювати даними які ми можемо отримувати. Це означає, що інтернет може бути не лише тим, що наші дані можуть потрапити не в ті руки, а і бути дуже сильним інструментом маніпуляції, що можна було неодноразово побачити під час цієї війни.

Якщо подивитись на одні з найпопулярніших ресурсів і застосунків, можна побачити досить велику вибірку в якій точно будуть знаходитись такі ір як: google, youtube, tiktok, telegram та інші. Майже всі ці ір безумовно дуже корисні та мають дуже вагому репутацію, проте не всі з них. Одним з перелічених ір – є телеграм, і не зважаючи на те, що телеграм є безумовно дуже якісно зробленим застосунком для комунікації, важко заперечувати, що його репутація, а скоріше його творця та власника, не достатньо чистою, особливо під час війни.

Навіть якщо ризики використання цього застосунку не проявлять себе, ідея відмови використання такими застосунками на користь інших, з чистішою репутацією, або створення своїх застосунків які б принаймні покривали певні потреби, хоча б на певних рівнях не є поганою.

Не складно помітити, що різні університети України демонструють свої думки з приводу цього питання, і вчасності про телеграм. Багато університетів, і в тому числі ХНУРЕ, демонстрували певне непокоєння з цього приводу. Саме

через це була створена ідея цього проєкту, проєкти який не просто може стати альтернативним джерелом комунікації між студентами, а проєктом який може замінити процес надання важливої інформації від до студентів.

З цим проєктом я хочу показати та проаналізувати, мінімальні потреби до створення інтернет застосунку, на прикладі веб-форуму, який буде вирішувати питання використання ресурсів, як телеграму, для донесення певної інформації до студентів.

1 ТЕХНОЛОГІЇ ТА FRAMEWORKS ПРОЄКТУ

Створення сайту – це комплексна задача, для виконання якої необхідно обрати й використати багато інструментів розробки. При обиранні технологій, які будуть використані при розробці, необхідно подивитись на потреби які необхідні для забезпечення функціонування фінального продукту, а також на технології з якими розробник найбільш знайомий.

В цьому розділі описані технології, які були використані для створення цього проєкту.

1.1 Мова програмування

Мова програмування — це система нотацій для написання комп'ютерних програм. Мови програмування описуються з точки зору їхнього синтаксису (форми) та семантики (значення) [1].

Кожна мова програмування має свій набір ключових слів, можливостей та синтаксис. Мова програмування – це мова високого рівня, яку не розуміє комп’ютер. Для того, щоб код був зрозумілий машині, він компілюється у зрозумілий комп’ютеру код (машинний код).

Мови програмування бувають різні: високого або низького рівня, ті що використовують компілятори та ті що використовують інтерпретатори.

Код написаний мовою програмування низького рівня, досить близький до машинного коду, а код написаний мовою програмування високого рівня має багато абстракцій та автоматизованих процесів, що спрощує життя розробникам.

Різниця між мовами програмування з інтерпретаторами та мовами програмування з компіляторами полягає в тому, як код написаний мовою програмування перетворюється на машинний код.

Інтерпретатори перед тим, як перетворити програмний код на машинний, спочатку виконують його. Компілятори, в свою чергу, одразу перетворюють програмний код на машинний, без його виконання.

1.1.1 TypeScript

TypeScript не є абсолютно новою мовою програмування. TypeScript це надбудова над JavaScript. Він був створений, щоб зменшити часове навантаження JavaScript розробників, на пошук та виправлення помилок.

JavaScript дуже проста мова програмування, завдяки своїй динамічній типізації, яка робить цю мову гнучкою та простішою за купу сучасних мов програмування. Динамічна типізація дозволяє змінним змінювати їх тип даних залежно від значення яке вони приймають. Однак, динамічна типізація також має і вагомий недолік. Через динамічну типізацію, розробники нерідко вистрілюють собі в ногу в процесі розробки, якщо не бути достатньо уважним.

Якщо десь відбулась помилка яка пов’язана з типом даних, залежно від розміру і складності проєкту, це може значно ускладнити процес знаходження та виправлення помилок. TypeScript був створений, щоб вирішити це, зробивши строго типізовану версію JavaScript.

Строга типізація може допомогти зі знаходженням та розумінням проблем, що може сильно зменшити часові витрати на виправлення.

Варто зазначати, TypeScript все ж не забороняє динамічну типізацію. При бажанні, можна присвоїти змінній тип даних any, що означає що змінна приймаю дані будь-якого типу.

Через те, що TypeScript не є окремою мовою яку розпізнає компілятор, перед тим як скомпілювати код, код спочатку переводитись в зрозумілий компілятору JavaScript. Який в свою чергу, перетворюється на машинний код.

1.1.2 Мова програмування C#

C# - це сучасна, кросплатформна, високорівнева, строго типізована, об’єктно-орієнтована мова програмування, розроблена Microsoft та яка використовує платформу .Net для розробки різного роду проєктів.

Для того, щоб запустити C# код, він спочатку компілюється в CIL код, який в свою чергу, завдяки JIT компілятору (Just-In-Time) перетворює CIL в машинний код [2].

C# має об’єктно орієнтовану структуру (все побудовано на класах), свій garbage collector який дозволяє писати код без витоку пам’яті, тобто нема потреби чистити пам’ять самому кожен раз, коли виділяються пам’ять в кучі. C# також має безліч інших функцій таких як LINQ (Language-Integrated Query), можливість писати асинхронний або багато поточний код, використовувати pointers для керування виділеною пам’яттю та багато іншого [3].

C# - це кросплатформна мова програмування, що означає що використання не залежить від жодної операційної системи.

C# має доступ до розширеної бібліотеки .Net SDK. Деякі бібліотеки є незалежними, як бібліотека для керування файловою системою або бібліотека колекцій. Інші є специфічні для одного робочого навантаження, як Asp.Net Core [3]. Окрім цього, за допомогою NuGet, можна знаходити та встановлювати бібліотеки з відкритого доступу.

1.2 Framework

Framework – це платформа, яка забезпечує усе необхідне для розробки програмного застосунку. Щоб краще зрозуміти, Framework можна уявити як підготовлений шаблон з набором необхідних інструментів. Framework використовує спільні ресурси, такі як бібліотеки, довідкові документи та об’єднує їх в один пакет. Цей пакет можна змінювати під потреби проєкту. За

допомогою фреймворку розробник може додавати або змінювати функції, щоб надати застосунку нову функціональність [4].

З огляду на кількість мов програмування, існую досить багато різних фреймворків для різних мов програмування, та для різних сфер, включно веб-розробку.

Існує купа різних back-end і front-end фреймворків які дуже сильно прискорюють процес розробки програмного забезпечення, автоматизуючи поширені завдання для веб-розробників.

1.2.1 React

React – це front-end фреймворк, його також називають бібліотекою, створений для спрощення та прискорення процесу розробки інтерфейсу користувача.

React пропонує свій підхід до розробки інтерфейсу, а саме поділення інтерфейсу на невеликі частини, або компоненти. Компоненти – це поєднання HTML та JavaScript, яке фіксує всю необхідну логіку, для відображення цієї частини інтерфейсу користувача. Таким чином, ці компоненти можна послідовно зібрати в більш складні частини програми [5].

Завдяки цьому підходу, можна не тільки спростити орієнтацію в коді та збільшити рівень читабельності. Завдяки цьому, компоненти які використовуються багато разів, можна просто повторно використати вже створений компонент, що зменшує кількість коду.

Окрім цього, React має свій концепт react-хуків. React-хук – це вбудовані функції для роботи з компонентами та можуть бути використані лише в межах компонента.

React має 9 вбудованих хуків, які дозволяють зберігати інформацію в пам’яті (useState), виконувати набір дії при зміні значення якоїсь змінної (useEffect), створювати змінні які посилаються на різні HTML елементи, або отримувати інформацію з батьківських компонентів [6].

Окрім вбудованих react-хуків, React також дозволяє створення власних хуків.

1.2.2 Asp.Net Core та .Net

.Net – це безкоштовна, кросплатформна, відкрита платформа для створення багатьох видів додатків. На ній можна запускати програми, написані різними мовами програмування, найпопулярніша з яких є C#. Вона спирається на високопродуктивне середовище виконання, яке використовується у виробництві багатьох великомасштабних додатків [7].

ANC (Asp.Net Core) – це фреймворк розробки кросплатформних веб застосунків, який в якості платформи використовує .Net.

Платформа розробки .Net разом з ANC надає всі необхідні інструменти для розробки HTTP застосунків, та не тільки, (рис.1.1) використовуючи C# як основну мову розробки, та Kestrel в якості веб-сервера, який використовується ANC за замовчення для того, щоб запускати web-site.

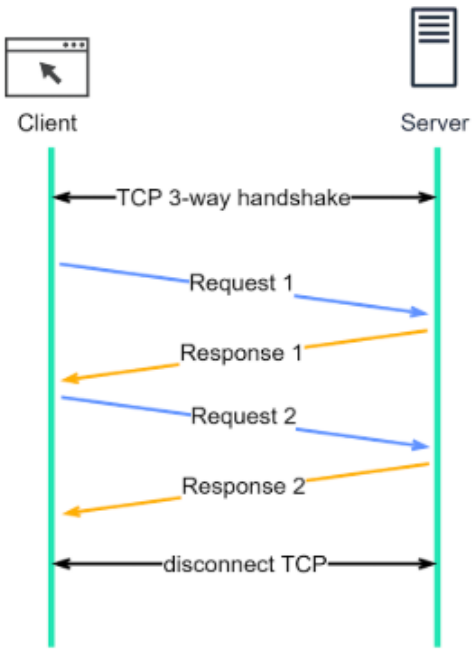

Рисунок 1.1 – Комунікація в Http based застосунках, де користувач (Client) отримує необхідні дані з серверу (Server) роблячи запити та отримуючи відповіді

Ці технології є одними з найкращих для розробки API які використовують C# в якості мови програмування.

Значення API полягає в тому, що API допомагає двом програмам або додаткам спілкуватися один з одним, надаючи їм необхідні інструменти та функції. Він приймає запит від користувача і надсилає його постачальнику послуг, а потім знову надсилає результат, отриманий від постачальника послуг, потрібному користувачеві [8].

Одні з основних термінів в ANC, та є ключовими для розуміння робити API, є: middleware, services, razor pages, controllers, та інше.

Дуже важливим серед них є middleware. Middleware – це програмне забезпечення, або набір інструкцій, яке спрацьовує кожен раз коли сервер обробляє запити які приходять на сервер. Тобто можна сказати, що middleware є свого роду конвеєром через який проходять запити відправлені серверу.

Services (dependency injection). ANC має вбудовану функцію впровадження залежностей (dependency injection). Залежності додаються за допомогою builder.Services та створюють всі необхідні об’єкти які необхідні для роботи API.

1.3 База даних

База даних – це організована колекція інформації, яка допомагає зберігати, записувати, читати, змінювати та видаляти дані в комп’ютерній системі [9].

БД (База даних) використовуються майже в кожному проєкті. Саме завдяки базам даних сервер може використовувати інформацію, яка була надіслана серверу місяці тому, і дозволяти сайту бути значно гнучкішим та динамічним.

Всі бази даних виконують одні й ті ж самі функції: зберігти, записати, прочитати (повернути), змінити та видалити. І якщо останні 3-4 функції незалежно від бази даних виглядають приблизно однаково, то зберігання інформації може відрізнятись за форматом.

На сьогодні існує два формати зберігання даних: реляційні та нереляційні.

Реляційні бази даних – це колекція структурованої інформації, сформована у таблиці. Вони представленні у вигляді рядків та стовпців, як в Microsoft Excel, де стовпці мають певні атрибути (id, дата створення, контент, ім’я відправника) та рядки які містять фактичні дані [10].

NoSQL, на відміну від реляційних БД, зберігає великі обсяги неструктурованої інформації, яку можна згрупувати в загальні категорії: документ, ключ-значення, колонка та граф без чітких зв’язків між собою [10].

1.3.1 Реляційні бази даних

Як вже було з’ясовано, реляційні БД організовують і зберігають інформацію у вигляді таблиць, які зв’язані між собою. Однак окрім цього, реляційні бази даних, такі як: MySql, SqlServer, PostgrSql та інші, мають кілька важливих концептів про які необхідно знати перед тим як починати працювати з ними.

Майже кожна реляційна база даних використовує мову SQL (Structured Query Language) для маніпулювання даними. SQL – це мова програмування для зберігання та обробки інформації в реляційних базах даних, та є найпопулярнішою мовою запитів [11]. SQL був винайдений у 1970-х роках на основі реляційної моделі зберігання даних, та на початку був відомий структурована англійська мова запитів (SEQUEL) [11]. Мова SQL має багато різних ключових слів, таких як: SELECT, WHERE, FROM, CREATE, GROUP BY, SET, UPDATE, JOIN, DELETE, DROP та інші. Використовуючи різні комбінації цих ключових слів, можна взаємодіяти з нашими таблицями (рис.1.2).

```
SELECT studentID, FullName, sat_score
FROM student
ORDER BY FullName DESC;
```

Рисунок 1.2 – Приклад виведення даних стовпців studentID, FullName та sat_score з таблиці student, які відсортовані у зворотному порядку за значеннями стовпця FullName

Ще дуже важливим є розуміння типів даних які існують в таких БД. Хоча не всі бази даних мають однаковий набір типів даних, в більшості своїй, різниця між ними в назві, хоча деякі бази даних мають певні типи даних які не мають інші. Приклад типів даних:

- char(n), varchar(n), nchar(n), nvarchar(n). Всі ці типу зберігають текст, різниця між ними може бути в максимально доступній кількості символів або в кодуванні яке використовується для них, та n є лімітом в кількості символів, яке можна вказати;

- bit, bool/boolean, int/integer, float, double, decimal, smallint, money. Всі ці типи даних представлять числа, буть то цілі (bit, bool, int) або з плаваючою комою (float, double, decimal, money).

– date, time, datetime. Ці типи даних призначені для зберігання дат та часу.

Окрім цього, дуже важливим концептом який використовується в реляційних БД – є ключі (primary key та foreign key).

Річ у тому, що жодна таблиця не може мітити рядки, дані яких повністю збігаються за кожним стовпцем, для того, щоб обійти цю проблему, були придумані primary key. Кожна таблиця має містити лише один стовпець з primary key і дані в таких стовпцях не можуть повторюватись, або не містити жодних даних (не можуть мати null). Задача такого ключа зробити кожен рядок унікальним, та за часту до кожної таблиці додають стовпець id так надають йому цей ключ.

Окрім проблеми з появою однакових рядків, розробники також подбали про можливі складнощі зі зберігання не дуже релевантних даних. При необхідності зберігання великої кількості різної інформації може виникнути ситуація, коли в таблиці споживачів потрібно зберегти інформацію, яка прямо не пов’язана з ними, але все одно має знаходитись в таблиці споживачів. Для цього були придумані публічні ключі (foreign key).

Наприклад, якщо необхідно зробити БД, яка буде зберігати інформацію о замовників якогось онлайн магазину та замовленнях. Для цього одразу

створюються дві таблиці одна з яких буде містити інформацію про замовників, а інша про замовлення, але тоді виникає питання, де зберігати інформацію про те, яке замовлення належить до якого замовника, і уявимо що нам потрібно мати більше ніж id замовлення та id замовника. Замість того щоб додавати декілька нових стовпців до обох таблиць, можна скористатись foreign key і додати лише по одному стовпцю до кожної з них з цим ключем, який буде працювати як посилання на рядок в іншій таблиці та прив'язаний до primary key іншої таблиці. Таким чином кожен рядок з однієї таблиці буде мати доступ до даних відповідного рядка з іншої (рис. 1.3). Проте альтернативним варіантом може бути створення окремої таблиці яка буде містити два ключі які будуть вказувати на primary key інших таблиць, та за бажанням навіть (або необхідністю) додати до цієї таблиці інші рядки які будуть мітити важливу інформацію для обох таблиць.

Рисунок 1.3 – Приклад таблиці реляційної бази даних

1.3.2 NoSQL бази даних

На відміну від реляційних БД, NoSql (Not Only Sql) працює з іншими форматами великих обсягів неструктурованої інформації, яку можна згрупувати в загальні категорії: документ, ключ-значення, колонка та граф без чіткого зв’язку між собою [10].

NoSql бази даних, на відміну від реляційних, бувають різних типів, залежно від їхньої моделі. NoSql зберігаються дані у структурах подібних до документів, пари ключ значення, колонко орієнтовані або графи. Вони забезпечують гнучкі схеми та легко масштабуються з великими обсягами даних [12].

Сховища типу ключ-значення – це дуже простий тип БД, який має дві колонки для даних (одна для ключа, а інша для інформації). В цьому типі NoSql, кожний елемент зберігається як пара ключ-значення [12].

Колонко орієнтовані БД мають рядки та стовбці для зберігання інформації, але на відміну від реляційних БД, вони не фіксовані в таблиці та мають динамічну схему. Пов'язані стовпці утворюють сімейства стовпців, а база даних зберігає сімейства стовпців окремо [13].

Документно подібна NoSql зберігає дані у форматі bson, json або xml (приклад формату bson зображено на рис.1.4). Інформація зберігається файлах з гнучкою схемою збереження даних [12].

Графова база даних зосереджена на зв'язках між елементами даних. Кожен елемент міститься як вузол. Зв'язки між елементами в базі даних називаються посиланнями або відношеннями. Зв'язки – це елементи першого класу бази даних, що зберігаються безпосередньо [12].

Рисунок 1.4 – Приклад формату bson в якому документно-подібна БД (наприклад MongoDB) зберігає дані

Будь-який тип NoSql, дозволяє зберігати дані і дуже гнучкому форматі, чого не можуть надати реляційні БД.

1.3.3 Вибір бази даних для проєкту

Для того, щоб порівняти реляційні та NoSQL бази даних, варто розглянути які переваги та недоліки вони мають.

Переваги реляційні БД над NoSQL:

- Цілісність даних;
- Кращі для складних запитів (гнучкість запитів);
- Чіткість структури схеми.

Переваги NoSQL над реляційними БД:

- Гнучкість, через відсутність чіткої схеми ;
- Певна перевага в швидкості;
- Складність розробки.

Не зважаючи на деякі переваги NoSql, я все ж більше надаю перевагу реляційним БД за їх структурованість, точність та можливість робити складні запити. Саме тому для роботи на цим проєктом була обрана Sql Server (реляційна база даних від Microsoft).

1.4 HTML та CSS

HTML – це мова розмітки гіпертексту, і це означає, що вона використовується для визначення структури з використанням різних тегів (елементів) [14].

Кожна сторінка в інтернеті створена за допомогою HTML та його тегів. Кожен тег можна представити як блок з певними властивостями, навіть якщо це абзац ({text} текст починається з відступу) або заголовок (<h1>{text}</h1> задає тексту розмір шрифту 2em) (рис.1.5).

Теги можуть бути двох видів: закриті та відкриті. Приклад закритих тегів: , , <body></body>, , <a> та інші. Приклад відкритих тегів: <input />, <img />,
, <video /> та інші.

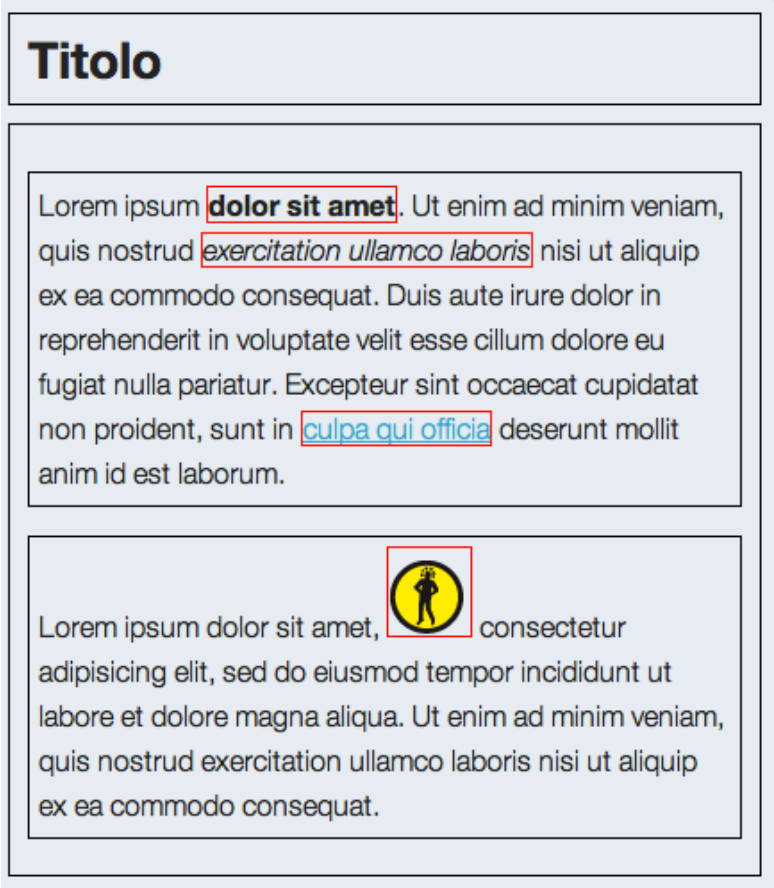

Рисунок 1.5 – Сторінка з різних HTML тегів та виділення цих елементів.

Теги можуть мати атрибути, що містять додаткову інформацію. Наприклад тег <a> має атрибут href=””. Цей атрибут приймає посилання на ресурс. Тег <a> - це тег гіперпосилання, тому Nure створює текст посилання, при натиску на яке, буде виконана переадресація на вказаний в атрибуті href ресурс.

Кожна сторінка має свою власну специфічну структуру (рис.1.6). Все починається з <!DOCTYPE html>, який оголошує тип документа. Тег <html></html> означає початок і кінець усього документу. Тег <head></head> містить інформацію про сторінку. Тег <title></title> вказує назву сторінки яка відображається в рядку заголовка браузера. Тег <meta> задає метадані сторінки. Тег <body></body> містить основний контент який відображається на сторінці браузера [14].

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Рисунок 1.6 – Структура HTML сторінки

CSS (Cascading Style Sheets) – це мова, або таблиці стилів, яка використовується для задання стилів HTML елементам для дизайну веб-сторінок.

Основні компоненти CSS це селектори та оголошення, де селектор це HTML елемент, а оголошення (набір властивостей та значень) (рис.1.7)[14].

```css
селектор {
властивість1: значення;
властивість2: значення;
властивість3: значення;
}

p {
color: #000000;
}

h1 {
font-size: 50px;
}
```

Рисунок 1.7 – CSS Синтаксис

Селектором може бути не тільки назва тегу. Кожен тег HTML може мати атрибут “class” або “id” які допомагають згрупувати елементи під одним класом, або задати свій окремий ідентифікатор для будь-якого елементу. Використовуючи ці атрибути, CSS може задати стилі до елементів які

відносяться до певного класу поставивши крапку перед назвою класу, або конкретному елементу за його id використовуючи “#” символ (рис.1.8).

```
#mioParagrafo {
color: #222222;
}

.miaClasse {
font-size: 18px;
}
```

Рисунок 1.8 - CSS Синтаксис для класів та id

Окрім цього CSS має багато інших можливостей, таких як створення анімацій або додавання стилів при виконанні певної дій, наприклад наведення на елемент.

1.5 Postman

Postman – це платформа для створення та роботи з API, яка позбавляє від труднощів на етапах проєктування та тестування API [15].

Postman має чудовий та інтуїтивно зрозумілий інтерфейс з купою інструментів для створення запитів та транслюванням відповідей від API (рис.1.9).

З допомогою цього застосунку, розробка та тестування стає набагато легше.

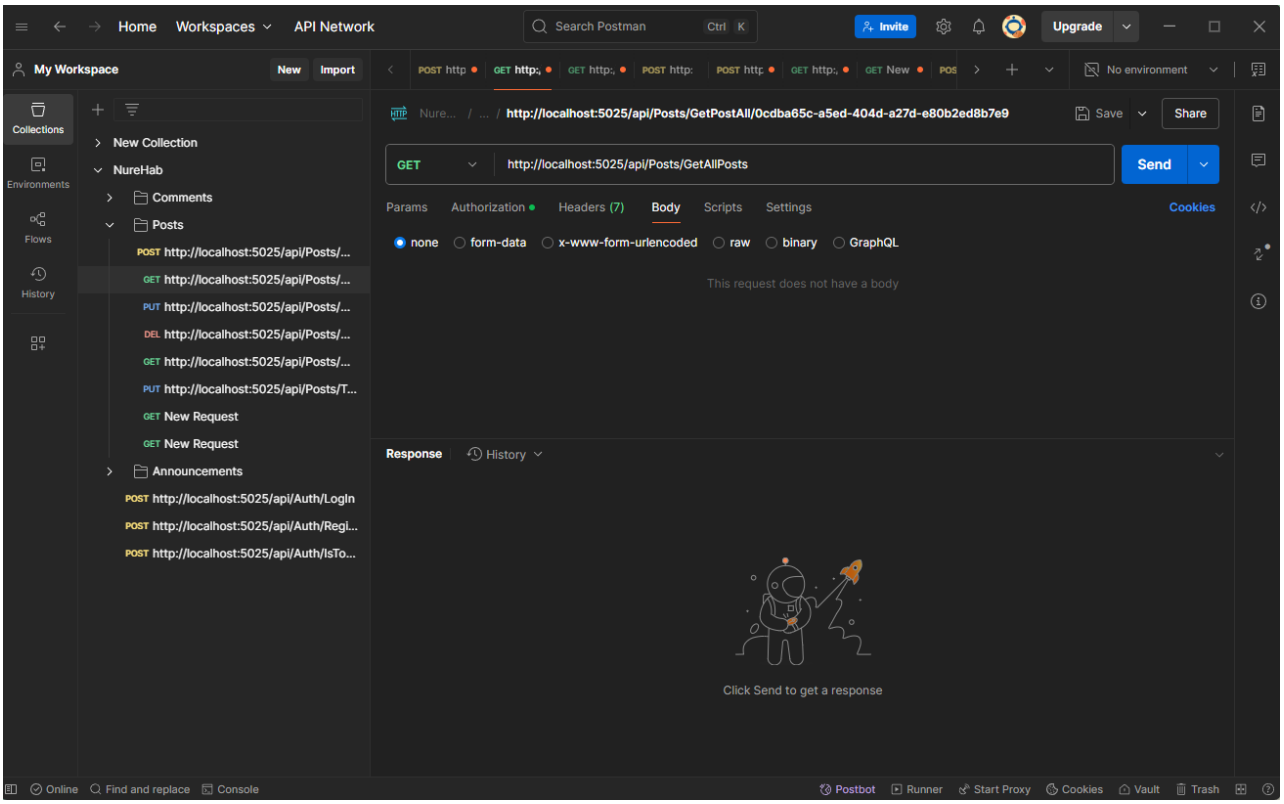

Рисунок 1.9 – Інтерфейс Postman

Postman надає можливості для створення різних типів запитів (get, post, put, delete) з можливістю додавання та налаштування headers та cookies. Окрім можливостей роботи з HTTP протоколом, Postman також підтримує webSocket, GraphQL та інше.

2 РОЗРОБКА ТА РОЗГОРТАННЯ ПРОЄКТУ

2.1 Постановка завдання

Стоїть задача розробити інтуїтивно зрозумілий та функціонально працюючий веб-форум для ХНУРЕ, який має виконувати наступні функції:

- Мати закритий доступ для користувачів інтернету, які не є студентами або викладачами;
- Надавання чи не надавання доступу до певного функціоналу, за наявністю чи відсутністю необхідної ролі;
- Створення оголошень та дописів;
- Відправлення сповіщення про отримане оголошення на пошту з деталями цього оголошення;
- Надання можливості обирати, студенти яких кафедр мають отримати оголошення чи допис;
- Можливість залишати коментарі та реакції під оголошеннями та дописами, а також можливість їх закриття;
- Редагування чи видалення оголошень, дописів та коментарів;
- Пошук з фільтрами для оголошень та дописів;
- Зберігання оголошень, дописів, коментарів та інформації про користувачів у БД;
- Підтвердження пошти при реєстрації та можливість зміни паролю.

2.2 Файлова структура проєкту

Було обрано просту файлову структуру проєкту. В теці проєкту були створені окремо папки для серверної та клієнтської частини. Папка з серверною частиною окрім всіх важливих для запуску і роботи файлів, містить наступні

теки: Controllers (містить код, якій відповідаю за обробку запитів), Data (містить файли, які потрібні для налаштування Entity Framework Core (ORM)), DTOs (містить моделі, які використовуються для приймання та відправлення даних), Extensions (містить файл з додатковими налаштуваннями залежностей), Models (містить моделі), Services (Містить методи з додатковою логікою)(рис.2.1).

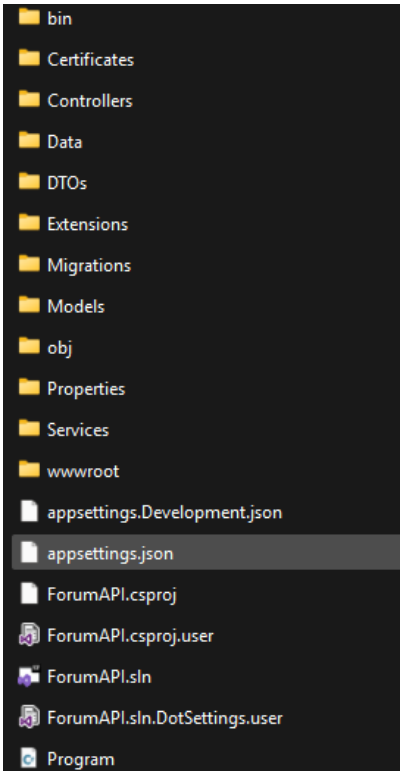

Рисунок 2.1 – Файлова структура серверної частини

Клієнтська сторона містить стандартне для React поділення на теки: node_modules, public та src. Майже вся робота проходить в папці src. Для клієнтської частини проєкту, було обрано наступну файлову структуру: Components, lib (містить код для взаємодії з API), Models, Pages, Styles (рис.2.2).

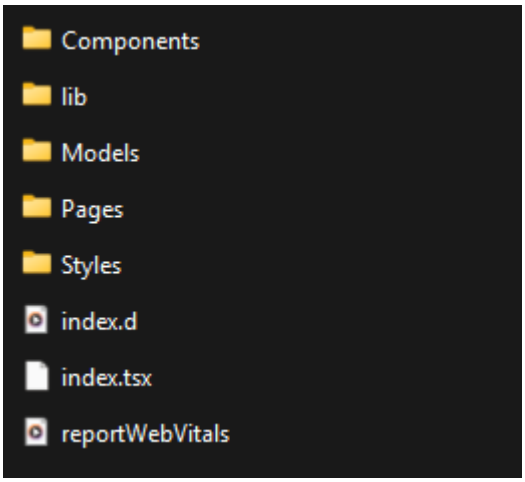

Рисунок 2.2 – Файлова структура клієнтської частини

Після підготовки файлової структури, можна почати розробку сайту.

2.3 Розробка серверної частини

2.3.1 Абстракції та Data Transfer Objects

Розуміння того як будуть представленні дані, які властивості вони будуть мати та як взаємодіяти між собою, може зберегти дуже багато часу під час розробки.

У процесі обдумування абстракцій, було вирішено, що проєкт має містити наступні моделі: користувач, оголошення, допис, коментар та реакція (лайк).

Оголошення має наступні параметри: id (primary key), заголовок, текст (контент), дата і час створення, id автора (foreign key), користувач (навігаційна властивість), чи з коментарями, кількість реакцій, кількість коментарів та група користувачів які отримають оголошення (рис.2.3). Таким чином були підготовлені й ніші моделі для серверної частини (рис.2.4 – 2.7).

```
public class Announcement
{
    [Key]
    2 usages
    public required string Id { get; set; }
    5 usages
    public string? Title { get; set; }
    3 usages
    public string? Text { get; set; }
    5 usages
    public DateTime CreatedAt { get; set; }
    [ForeignKey( name: "Author")]
    2 usages
    public string? AuthorId { get; set; }
    public ApplicationUser? Author { get; set; }
    2 usages
    public int Likes { get; set; }
    2 usages
    public bool WithComments { get; set; } = false;
    5 usages
    public List<string> GroupVisibility { get; set; } = new List<string> { "*" };
    3 usages
    public int AmountOfComments { get; set; }
}
```

Рисунок 2.3 – Модель оголошень

```
public class Post
{
    [Key]
    3 usages
    public required string Id { get; set; }
    4 usages
    public string? Title { get; set; }
    2 usages
    public string? Text { get; set; }
    5 usages
    public DateTime CreatedAt { get; set; }
    [ForeignKey( name: "Author")]
    2 usages
    public string? AuthorId { get; set; }
    public ApplicationUser? Author { get; set; }
    3 usages
    public int Likes { get; set; }
    2 usages
    public bool WithComments { get; set; } = true;
    4 usages
    public List<string> GroupVisibility { get; set; } = new List<string> { "*" };

    2 usages
    public int AmountOfComments { get; set; }
}
```

Рисунок 2.4 – Модель дописів

```
public class ApplicationUser: IdentityUser
{
    6 usages
    public required string Group { get; set; }
    4 usages
    public bool AnnouncementsNotifications { get; set; } = false;
    3 usages
    public DateTime? NextGroupChangingDateAvailable { get; set; }
}
```

Рисунок 2.5 – Модель користувачів

```
public class Comment
{
    [Key]
    4 usages
    public required string Id { get; set; }
    2 usages
    public string? Text { get; set; }
    1 usage
    public DateTime CreatedAt { get; set; }
    [ForeignKey( name: "Post")]
    4 usages
    public string? PostId { get; set; }
    [ForeignKey( name: "Author")]
    1 usage
    public string? AuthorId { get; set; }
    public ApplicationUser? Author { get; set; }
    3 usages
    public int Likes { get; set; }
    [ForeignKey( name: "ReplyTo")]
    4 usages
    public string? ReplyTo { get; set; }
    2 usages
    public List<Comment> RepliesToComment { get; set; } = new List<Comment>();
}
```

Рисунок 2.6 – Модель коментарів

```
public class Like
{
    [Key]
    3 usages
    public required string Id { get; set; }
    [ForeignKey( name: "Author")]
    7 usages
    public required string UserId { get; set; }
    7 usages
    public required string LikeFor { get; set; }
}
```

Рисунок 2.7 – Модель лайків

Окрім цього були також створені абстракції для клієнтської сторони, та моделі для обміну даних між клієнтською та серверною частиною, або DTO (Data Transfer Object) (рис.2.8).

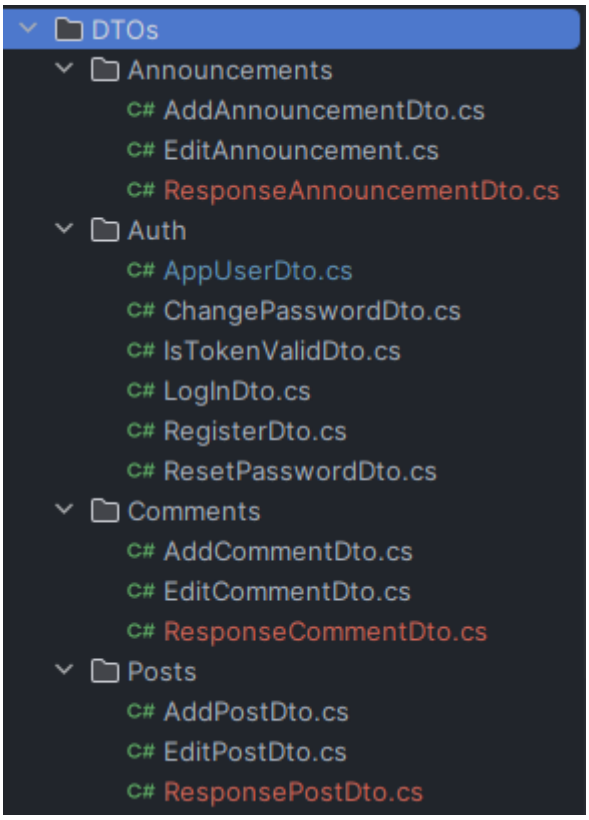

Рисунок 2.8 – Моделі для обміну даних між серверною та клієнтською частиною

Завдяки цим абстракціям, можна легко взаємодіяти з наданими даними з клієнтської сторони, такі як: реєстраційна форма чи заповнена форма зі змінами, які потрібно зробити для допису. А також передавати дані в зручному для клієнтської сторони форматі.

2.3.2 Налаштування політики Cross-Origin Resource Sharing

Для того, щоб дозволити front-end та back-end проєктам взаємодіяти між собою, необхідно налаштувати політику CORS.

CORS (Cross-Origin Resource Sharing) – це механізм на основі HTTP-заголовків, який запобігає несанкціонованим запитам зі сторонніх доменів.

CORS також спирається на механізм, за допомогою якого браузер надсилає запит до сервера, щоб перевірити чи сервер дозволить фактичний запит [16].

За допомогою Asp.Net Core можна дуже легко налаштувати CORS. Для цього необхідно просто додати необхідний сервіс, задати назву налаштуванням політики, та за допомогою методів, вказати домени та те, що їм дозволено (рис.2.9).

```
Services.AddCors(opt :CorsOptions =>
{
    opt.AddPolicy(name: "ForumPolicy",
        configurePolicy: policy =>
        {
            policy.WithOrigins("http://localhost:3000", "https://localhost:3000")
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials();
        });
});
```

Рисунок 2.9 – Налаштування CORS

За допомогою методу WithOrigins(), додаються вказані домени до політики. За допомогою інших методів вказано, що дозволені будь-які headers, methods (get, post, delete та інші) та повноваження (credentials), які нам знадобляться для налаштування авторизації та аутентифікації.

В якості доменів, вказані локальні адреси від React проєкту, з його стандартним хостом (3000).

Для того, щоб CORS почав працювати, необхідно додати CORS middleware та вказати назву створених налаштувань, яку було задано при налаштуванні політики:

– app.UseCors("ForumPolicy");

2.3.3 Підключення та налаштування бази даних

Для того, щоб застосунок мав доступ до бази даних та міг взаємодіяти з нею, веб-серверу необхідний встановити зв’язок за допомогою connection string.

Connection string – це короткий рядок, який містить необхідну інформацію для підключення до БД, таку як: назва БД, сервер бази даних, чи можна під’єднатись без пароля, пароль та інше.

Приклад connection string для локальної БД:

"Server=SOMETHING\\MSSQLSERVER01;Database=NureForumDB;Trusted_Connection=true;TrustServerCertificate=true"

Дані, які краще приховати від чужого ока, краще зберігати в окремому для цього файлі. В Asp.Net Core такий файл створюється автоматично при створенні проєкту та називається appsettings.json. Відкривши файл, додав connection string, у json форматі:

```
"ConnectionStrings":{
    "ForumDataBase":ConnectionString
}
```

Після цього, необхідно створити клас, який буде представляти БД. Але перед цим, необхідно встановити декілька пакетів:

- Microsoft.AspNetCore.Identity.EntityFramworkCore
- Microsoft.EntityFramworkCore.Design
- Microsoft.EntityFramworkCore.Tools
- Microsoft.EntityFramworkCore.SqlServer

Завдяки цим пакетам, можливо взаємодіяти з SqlServer через ORM (Object-Relational Mapping).

ORM – це техніка програмування, за допомогою якої, можна працювати з БД через об’єкти у коді.

Перейшовши в теку Data, створив файл ForumDataContext.cs який буде містити абстракцію БД (рис.2.10).

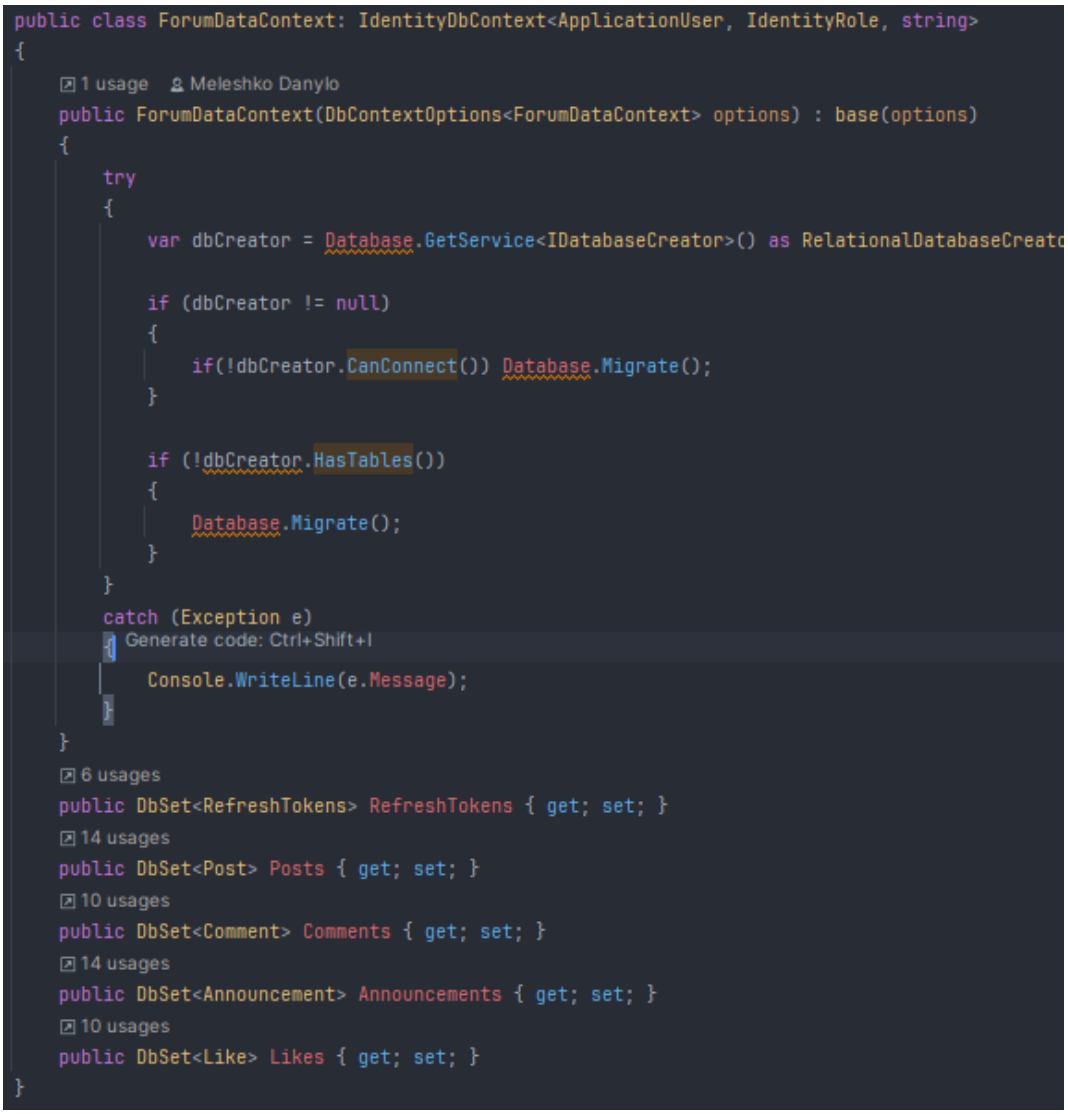

Рисунок 2.10 – Абстракція БД (клас)

Клас наслідує інший клас з пакета Microsoft.AspNetCore.Identity.EntityFramworkCore, використовує базовий конструктор IdentityDbContext, а також містить код який відповідає за створення БД чи необхідних таблиць, якщо при спробі зв’язку з БД, виявилось що вона не створена, або не містить таблиць. Окрім конструктора, клас також містить спеціальні властивості, завдяки яким можна взаємодіяти з таблицями бази даних.

Після створення абстракції БД, необхідно додати цю абстракцію до колекції залежностей та передати наш connection string, який буде використаний для підключення до SqlServer:

```
Services.AddDbContext<ForumDataContext>(opt=>
{
   opt.UseSqlServer(configuration.GetConnectionString("ForumDataBase"));
});
```

Після цього, необхідно скористатись консоллю та створити migrations. Migrations допомагають створювати та вносити зміни до EF моделі БД.

Відкривши консоль, необхідно прописати наступну команду для ініціалізації migrations, де назва задається в кінці, та може бути будь-якою: dotnet ef migrations add InitialCreate.

Після чого, в проєкті буде створена нова папка з файлами, які будуть відповідати за схему БД.

Далі, знов відкривши консоль, потрібно прописати команду яка створить базу даних з EF моделлю: dotnet ef database update.

Після успішного виконання, при підключенні до БД, можна помітити появу бази даних з усіма таблицями (рис.2.11).

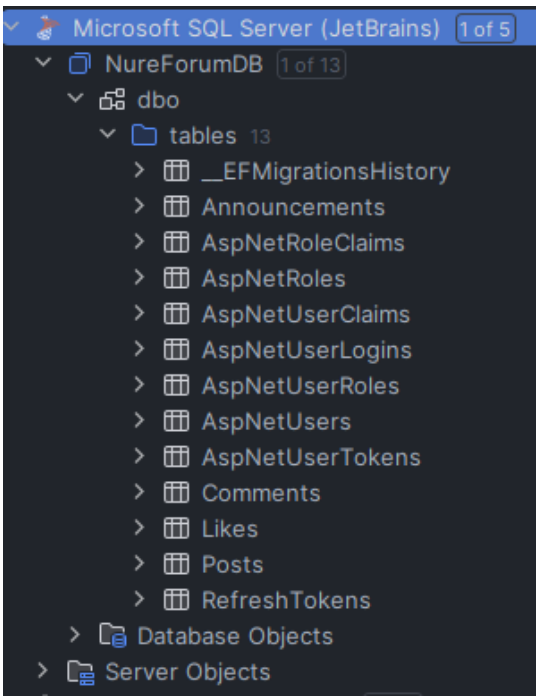

Рисунок 2.11 – Результат створення БД з необхідною схемою за допомогою EF Core

Після цього, створивши об’єкт бази даних в коді, використовуючи функціонал класу IdentityDbContext, можна взаємодіяти з базою даних, таблицями та даними з таблиць.

2.3.4 Налаштування безпеки даних користувачів

При реєстрації, користувач має створити пароль та підтвердити свою електронну пошту. Для підтвердження, на пошту користувача проходить електронний лист з токеном, зі стандартним часовим лімітом для токена (10 хвилин). Користувач має використати отриманий токен на сайті, що підтвердить існування електронної пошти, та належність її до користувача.

Встановлений Microsoft.AspNetCore.Identity.EntityFramworkCore має забезпечити хешування паролів. Таким чином, якщо зловмисник отримав доступ до даних в БД, він не зможе отримати паролі в їх першочерговому вигляді.

При створенні акаунту, користувачі мають отримати роль викладача, старости чи студента. Окрім цього, створення акаунту на сайті має бути доступно лише тим, хто має пошту університету.

Аутентифікація – процедура встановлення належності користувачеві інформації в системі [17].

Налаштувати авторизацію можна в два способи: session та Jwt (JSON Web Token).

Після успішної валідації облікових даних, сервер створює сесію і пов’язує її з унікальним id. Зберігши сесію на диску чи в базі даних, сервер надсилає id сесії через http only cookie до клієнта. Після отримання cookie, клієнт надсилає їх з кожним запитом, а сервер верифікує отриманий id [18].

Http only cookie – це безпечний спосіб спілкування між сервером та клієнтом, бо клієнт не має доступу до них, але клієнт може передавати їх з кожним запитам до сервера.

Jwt складніше та цікавіше ніж сесії. В якості формату в якому Jwt зберігає дані, використовується JSON. При створенні токена, сервер підписує файл,

шифруючи його за допомогою секретного ключа, та встановлює час його валідності. Jwt має два види шифрування: симетричне та асиметричне. В обох варіантах для шифрування використовується секретний ключ, але якщо в симетричному, секретний ключ використовується і для шифрування, і для розшифрування, то в асиметричному, для розшифрування використовується публічний ключ. Публічний ключ створюється при формуванні токена та може бути переданий будь-кому, не хвилюючись за його безпеку.

Переваги асиметричного шифрування:

- Більш безпечний, бо не потрібно передавати секретний ключ;
- Краще підходить для мікросервісної інфраструктури.

Переваги симетричного шифрування:

- Простіше впровадження;
- Більша швидкість, завдяки простішій математиці.

Токен передається у заголовку запиту (Authorization), з припискою “Bearer” перед токеном (Приклад: Authorization: Bearer {token}). Як тільки ключ перестає бути валідним, користувачу необхідно створити новий.

Порівнюючи session та Jwt, session простіші в імплементації, але Jwt краще для масштабування.

Для того, що відкрити кращі можливості для масштабування проєкту. Було обрано Jwt, та для початку симетричний метод шифрування, для простішого впровадження (рис.2.12).

```
Services.AddIdentityCore<ApplicationUser>(opt :IdentityOptions =>
{
    opt.Password.RequireNonAlphanumeric = false;
    opt.Password.RequiredLength = 8;
    opt.Password.RequireUppercase = true;
    opt.Password.RequireLowercase = true;
    opt.Password.RequireDigit = true;
    opt.SignIn.RequireConfirmedEmail = true;
    opt.Tokens.EmailConfirmationTokenProvider = TokenOptions.DefaultEmailProvider; Generate

}).AddEntityFrameworkStores<ForumDataContext>()
  .AddRoles<IdentityRole>()
  .AddDefaultTokenProviders();

Services.AddAuthentication( configureOptions: opt :AuthenticationOptions =>
{
    opt.DefaultAuthenticateScheme =
    opt.DefaultForbidScheme =
    opt.DefaultChallengeScheme =
    opt.DefaultScheme =
    opt.DefaultSignInScheme =
    opt.DefaultSignOutScheme = JwtBearerDefaults.AuthenticationScheme;
})
    .AddJwtBearer(opt :JwtBearerOptions => {
    opt.TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateIssuer = true,
        ValidIssuer = configuration["JWT:Issuer"],
        ValidateAudience = true,
        ValidAudience = configuration["JWT:Audience"],
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            System.Text.Encoding.UTF8.GetBytes(configuration["JWT:SigningKey"])),

        RoleClaimType = ClaimTypes.Role
    };
});
```

Рисунок 2.12 – Впровадження Jwt стандарту

AddIdentityCore<TUser>() метод додає services для реєстрації користувачів, використовуючи UserManager<TUser>, забезпечуючи хешування паролів. Opt задає параметри для валідації. AddEntityFrameworkStores<TStore>(), AddRoles<TRole>() та AddDefaultTokenProviders() додають можливість взаємодіяти з Ef core, підтримку ролей та можливість створювати токени.

AddAuthentication() та AddJwtBearer() задають схему для аутентифікації, а також параметри валідації Jwt токенів.

Після налаштування Jwt, необхідно підготувати метод для створення Jwt токена (рис.2.13).

```
public string CreateToken(ApplicationUser user)
{
    if(user == null) throw new ArgumentNullException();

    var Claims = new List<Claim>
    {
        new Claim( type: ClaimTypes.NameIdentifier, user.Id),
        new Claim( type: ClaimTypes.Email, user.Email!)
    };

    var roles :IList<string> = _userManager.GetRolesAsync(user).Result;
    Claims.AddRange( collection: roles.Select(role :string => new Claim( type: ClaimTypes.Role, role)));

    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JWT:Signingkey"]!));
    var creds = new SigningCredentials(securityKey, algorithm: SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _configuration["JWT:Issuer"],
        audience: _configuration["JWT:Audience"],
        claims: Claims,
        expires: DateTime.UtcNow.AddMinutes(2),
        signingCredentials: creds);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Рисунок 2.13 – Метод для створення Jwt токену

Claims представляють дані, які зберігатимуться в токені, в даному випадку, id користувача, електронна пошта, а також його роль. Після чого, перед тим як створити токен, створюється підпис на основі секретного ключа та алгоритму шифрування.

Головний недолік симетричного методу шифрування – це використання одного ключу для шифрування та розшифрування. Для того, щоб зменшити ризики, варто використати refresh токен.

Завдяки refresh токену, можна створювати новий токен для користувача, без необхідності наново передавати облікові дані. Це дозволить зробити Jwt

токен короткотривалим, без необхідності користувача наново заповнювати форму для входити в систему кожні 2 хвилин.

Refresh токен – це додатковий токен, який створюється при підтвердженні облікових даних користувача, використовується для створення нових токенів та передається через http only cookie.

Короткотривалість Jwt токена важлива, бо на відміну від сесій, заборонити дію токена не можливо, до того як його час дій закінчиться. Як тільки хтось отримав токен іншого користувача, він може використовувати його до того моменту, як він перестає бути дійсним.

Для того, щоб підняти рівень безпеки, Jwt токен має час дії 2 хвилин, та щоб створювати нові Jwt токени, було створено endpoint (рис.2.14).

Endpoint – це url-адреса, яка виступає точкою контакту між клієнтом API та сервером API [19].

```
[HttpPost]
Meleshko Danylo *
public async Task<IActionResult> RefreshToken()
{
    var oldRefreshToken :string? = Request.Cookies["refreshToken"];
    if (oldRefreshToken == null) return Unauthorized("Invalid refresh token");

    var storedRToken :RefreshTokens? = await _dbContext.RefreshTokens // DbSet<RefreshTokens>
        .FirstOrDefaultAsync(t :RefreshTokens => t.Refreshtoken == oldRefreshToken); // Task<RefreshTokens?>

    if (storedRToken == null) return Unauthorized("Invalid refresh token");

    var user = await _userManager.FindByIdAsync(storedRToken.TokenOwner);
    if (user == null) return Unauthorized("This account doesn't exist");
    if (storedRToken.ExpirationTime < DateTime.UtcNow) return Unauthorized("Time to log in");

    _dbContext.RefreshTokens.Remove(storedRToken);

    var newRefreshToken :string = _tokenService.CreateRefreshToken();
    var newTokenEntry = new RefreshTokens
    {
        RefreshTokenId = Guid.NewGuid().ToString(),
        TokenOwner = user.Id,
        Refreshtoken = newRefreshToken,
        ExpirationTime = DateTime.UtcNow.AddDays(14)
    };

    _dbContext.RefreshTokens.Add(newTokenEntry);
    await _dbContext.SaveChangesAsync();

    var newAccessToken :string = _tokenService.CreateToken(user);

    Response.Cookies.Append("refreshToken", newRefreshToken, new CookieOptions
    {
        HttpOnly = true,
        Secure = true,
        SameSite = SameSiteMode.Strict,
        Expires = storedRToken.ExpirationTime
    });

    return Ok(new { Token = newAccessToken, Group = user.Group, Role = user.Role, Id = user.Id, Email = user.Email });
}
```

Рисунок 2.14 – Refresh токен endpoint

RefreshToken() перевіряє запит на наявність refresh токена серед http only cookie, а також те, чи є він в базі даних сервера. Якщо все добре, то створюється нові Jwt та refresh токени, та передаються назад користувачу через http only cookie.

Наступним кроком, реалізація endpoints для реєстрації, входу та виходу з системи (рис.2.15 – 2.17).

Endpoint для реєстрації:

- Приймає облікові дані користувача;
- Перевіряє наявність облікових даних;
- Перевіряє існування користувача в базі даних;
- Створює токен для підтвердження електронної пошти;
- Надсилає токен на пошту користувача;
- Після підтвердженні, перевіряє чи може користувач отримати запитану роль;
- Створює акаунт.

Endpoint для входу в систему:

- Приймає облікові дані користувача;
- Перевіряє наявність облікових даних;
- Перевіряє існування користувача в базі даних;
- Створює новий refresh токен та Jwt токен;
- Додає refresh токен до БД;
- Передає refresh токен та Jwt токен.

Endpoint для виходу з системи:

- Видаляє refresh токен з http only cookie та БД.

```
[HttpPost]
Meleshko Danylo *
public async Task<IActionResult> Register([FromBody] RegisterDto register)
{
    if (register == null) return BadRequest();

    var existingUser = await _userManager.FindByEmailAsync(register.Email);

    if (existingUser != null)
    {
        bool isEmailConfirmed = await _userManager.IsEmailConfirmedAsync(existingUser);

        if (!isEmailConfirmed)
        {
            var newToken string = await _userManager.GenerateEmailConfirmationTokenAsync(existingUser);
            await _emailSender.SendEmailAsync(existingUser.Email, subject: "Confirm your email", message: newToken);

            return Ok(new { message = "Please confirm your account. A new confirmation email has been sent." });
        }

        return BadRequest( error: new { message = "User with this email already exists and is confirmed." });
    }

    ApplicationUser newUser = new ApplicationUser
    {
        UserName = register.Email,
        Email = register.Email,
        Group = register.Group!
    };

    if (register.Group == "Teacher") {
        newUser.Role = "Teacher";
        var result = await _userManager.CreateAsync(newUser, register.Password!);
        if (!result.Succeeded) {
            return BadRequest( error: new { message = "Invalid email or password, or user already exists." });
        }

        if (!Teachers.TeacherEmails.Contains(register.Email.Trim()))
            return Forbid();

        await _userManager.AddToRoleAsync(newUser, role: "Teacher");
    }
    else {
        var result = await _userManager.CreateAsync(newUser, register.Password!);
        if (!result.Succeeded) {
            return BadRequest( error: new { message = "Invalid email or password, or user already exists." });
        }
        if(register.Email.Split("@")[1] != "nure.ua") return Forbid();  Generate code: Ctrl+Shift+I

        if(register.IsStarosta) await _userManager.AddToRoleAsync(newUser, role: "Starosta");
        else await _userManager.AddToRoleAsync(newUser, role: "Student");
    }

    var confirmationToken string = await _userManager.GenerateEmailConfirmationTokenAsync(newUser);
    await _emailSender.SendEmailAsync(newUser.Email, subject: "Confirm your email", message: confirmationToken);

    return Ok(new { message = "Please confirm your account" });
}
```

Рисунок 2.15 – Endpoint для реєстрації

```
[HttpPost]
Meleshko Danylo *
public async Task<ActionResult<AppUserDto>> LogIn([FromBody] LogInDto? logIn)
{
    if(logIn == null) return BadRequest();

    var user = await _userManager.FindByEmailAsync(logIn.Email);
    if (user == null || !await _userManager.CheckPasswordAsync(user, logIn.Password)) {
        return Unauthorized(new { message = "Invalid credentials" });
    }

    if (!user.EmailConfirmed) return Unauthorized("This account doesn't exist");

    Response.Cookies.Delete( key: "refreshToken");

    var token :string = _tokenService.CreateToken(user);
    var newRefreshToken :string = _tokenService.CreateRefreshToken();

    var refreshToken = new RefreshTokens
    {
        RefreshTokenId = Guid.NewGuid().ToString(),
        TokenOwner = user.Id,
        Refreshtoken = newRefreshToken,
        ExpirationTime = DateTime.UtcNow.AddDays(14)
    };

    await _dbContext.RefreshTokens.AddAsync(refreshToken);
    await _dbContext.SaveChangesAsync();

    Response.Cookies.Append("refreshToken", newRefreshToken, new CookieOptions
    {
        HttpOnly = true,
        Secure = true,
        SameSite = SameSiteMode.Strict,
        Expires = refreshToken.ExpirationTime
    });

    var response = new AppUserDto()
    {
        Id = user.Id,
        Email = user.Email,
        Token = token,
        Group = user.Group,
        Role = user.Role
    };

    return Ok(response);
}
```

Рисунок 2.16 – Endpoint для входу в систему

```
[HttpPost]
Meleshko Danylo
public async Task<IActionResult> LogOut() {
    var oldRefreshToken :string? = Request.Cookies["refreshToken"];
    if (oldRefreshToken == null) return Unauthorized("Invalid refresh token");

    var storedRToken :RefreshTokens? = await _dbContext.RefreshTokens // DbSet<RefreshTokens>
        .FirstOrDefaultAsync(t :RefreshTokens => t.Refreshtoken == oldRefreshToken); // Task<RefreshTokens?>

    if (storedRToken == null) return Unauthorized("Invalid refresh token");

    var user = await _userManager.FindByIdAsync(storedRToken.TokenOwner);
    if (user == null) return Unauthorized("This account doesn't exist");

    Response.Cookies.Append("refreshToken", oldRefreshToken, new CookieOptions
    {
        HttpOnly = true,
        Secure = true,
        SameSite = SameSiteMode.Strict,
        Expires = DateTime.UtcNow.AddSeconds(-1)
    });

    _dbContext.RefreshTokens.Remove(storedRToken);
    await _dbContext.SaveChangesAsync();

    return Ok();
}
```

Рисунок 2.17 – Endpoint для виходу з системи

2.3.5 Створення основного функціоналу

Створено метод для надсилання листу на електронну пошту (рис.2.18). Для того, щоб надсилання листу на електронну пошту, було створено новий акаунт електронної пошти, з якого будуть надходити листи.

```
public async Task SendEmailAsync(string email, string subject, string message)
{
    using var smtpClient = new SmtpClient(_config["Email:Host"], port: Convert.ToInt32(_config["Email:Port"]))
    {
        UseDefaultCredentials = false,
        EnableSsl = true,
        Credentials = new NetworkCredential(userName: _config["Email:NureHubEmail"], password: _config["Email:EmailPassword"])
    };

    using var mailMessage = new MailMessage
    {
        From = new MailAddress(_config["Email:NureHubEmail"]!, displayName: "NureHub"),
        Subject = subject,
        Body = message,
        IsBodyHtml = true
    };

    mailMessage.To.Add(email);

    await smtpClient.SendMailAsync(mailMessage).ConfigureAwait(false);
}
```

Рисунок 2.18 – Метод для надсилання листів електронною поштою

Створено endpoints для створення нового паролю та підтвердження електронної пошти (рис.2.19).

```
[HttpPost]
Meleshko Danylo
public async Task<ActionResult> ForgotPassword(string? email)
{
    if (email == null) return BadRequest( error: "Invalid email");

    var user = await _userManager.FindByEmailAsync(email);
    if(user == null) return Unauthorized("Couldn't find the user");

    var token :string  = await _userManager.GeneratePasswordResetTokenAsync(user);
    string encodedToken = WebUtility.UrlEncode(token);
    string link = $"{_config["Client"]}/ChangePassword?token={encodedToken}&email={email}";

    await _emailSender.SendEmailAsync(email,  subject: "Reset password link",  message: link);

    return Ok(new { message = "Password reset link sent successfully" });
}

[HttpPost]
Meleshko Danylo
public async Task<ActionResult> ResetPassword([FromBody] ResetPasswordDto changePassword)
{
    var user = await _userManager.FindByEmailAsync(changePassword.Email);
    if (user == null) return Unauthorized("Couldn't find the user");

    var result = await _userManager.ResetPasswordAsync(user, changePassword.ResetToken, changePassword.NewPassword);

    if (!result.Succeeded) {
        var errors :IEnumerable<string>  = result.Errors.Select(e :IdentityError  => e.Description);
        return BadRequest( error: new { errors });
    }
    return Ok(new { message = "Password reset successful" });
}

[HttpPost]
Meleshko Danylo
public async Task<IActionResult> EmailConfirmation(string? email, string? token)
{
    if (email == null || token == null) return BadRequest();

    var user = await _userManager.FindByEmailAsync(email);
    if (user == null) return NotFound("Email could not be found");

    var result = await _userManager.ConfirmEmailAsync(user, token);
    if (result.Succeeded) return Ok(new { message = "Email confirmed" });

    return BadRequest( error: "Email wasn't confirmed");
}
```

Рисунок 2.19 – Endpoints для створення нового паролю та підтвердження електронної пошти

Створено endpoints для отримання, редагування, видалення, додавання оголошень, дописів та коментарів (рис.2.20 – 2.23).

Створено декілька варіантів endpoints для отримання оголошень, дописів чи коментарів:

- Отримання 5 останніх доступних користувачу оголошень/дописів (одна сторінка містить 5 оголошень/дописів), з фільтруванням за автором, заголовком, датою створення та роллю користувача;
- Отримання 5 останніх дописів/оголошень створений користувачем (одна сторінка містить 5 оголошень/дописів). Було також додано фільтрування за заголовком та датою створення;
- Отримання конкретного оголошення/допису (необхідно для отримання детальної інформації про обраний допис чи оголошення);
- Отримання коментарів під дописом чи оголошенням.

Створено endpoints для залишання реакції до оголошення, допису чи коментарю (рис.2.24 – 2.26).

Створено додаткові endpoints:

- Для перевірки, чи залишив користувач реакцію під оголошенням, дописом чи коментарем;
- Для включення та виключення функції отримання сповіщень про нові оголошення;
- Для перевірки, чи включені сповіщення у користувача;
- Зміни групи чи ролі. Ця функція доступна лише раз на тиждень.

```
[Authorize]
[HttpGet]
Meleshko Danylo *
public async Task<IActionResult> GetAllAnnouncements([FromQuery] int page, [FromQuery] string? authorEmail,
                                                     [FromQuery] string? title, [FromQuery] DateTime? dateOfCreation,
                                                     [FromQuery] string? authorRole, CancellationToken ct)
{
    const int pageLimit = 5;
    int announcementsSkipped = (page - 1) * pageLimit;
    var userId :string? = User.FindFirstValue( claimType: ClaimTypes.NameIdentifier);
    if (userId == null) return Forbid();

    var user = await _db.Users // DbSet<ApplicationUser>
        .AsNoTracking() // IQueryable<ApplicationUser>
        .FirstOrDefaultAsync(x :ApplicationUser => x.Id == userId, ct); // Task<ApplicationUser?>
    if (user == null) return BadRequest();

    var query :IQueryable<Announcement> = _db.Announcements.Include( navigationPropertyPath: a :Announcement => a.Author).AsQueryable();

    if(user.Role == "Starosta") query = query.Where(x :Announcement => x.GroupVisibility.Any(g :string => g == "*" || g == user.Group));
    else {
        query = query.Where(x :Announcement => x.GroupVisibility.Any(g :string => g != "Старости"
                                                && (g == "*" || g == user.Group))); // IQueryable<Announcement>
    }

    if(!string.IsNullOrWhiteSpace(authorEmail)) {
        query = query.Where(a :Announcement => a.Author.Email.Contains(authorEmail));
    }
    if(!string.IsNullOrWhiteSpace(title)) {
        query = query.Where(a :Announcement => a.Title.Contains(title));
    }
    if (dateOfCreation.HasValue) {
        query = query.Where(a :Announcement => a.CreatedAt <= dateOfCreation.Value);
    }
    if(!string.IsNullOrWhiteSpace(authorRole)) {
        query = query.Where(a :Announcement => a.Author.Role == authorRole);
    }

    var totalAnnouncements :int = await query.CountAsync(ct);
    var totalPages = (int)Math.Ceiling(totalAnnouncements / (double)pageLimit);

    var announcements :List<Announcement> = await query
        .AsNoTracking() // IQueryable<Announcement>
        .OrderByDescending(x :Announcement => x.CreatedAt) // IOrderedQueryable<Announcement>
        .Skip(announcementsSkipped)
        .Take(pageLimit) // IQueryable<Announcement>
        .ToListAsync(ct); // Task<List<...>>

    return Ok(new
    {
        announcements = MapToResponseAnnouncementsDto(announcements),
        totalPages
    });
}
```

Рисунок 2.20 – Endpoint для отримання всіх оголошень, які доступні користувачу

```
[Authorize]
[HttpGet( template: "{id}")]
2 usages   new *
public async Task<ActionResult<ResponseAnnouncementDto>> GetAnnouncement(string? id, CancellationToken ct)
{
    var announcement = await _db.Announcements // DbSet<Announcement>
        .Include( navigationPropertyPath: a :Announcement => a.Author) // IIncludableQueryable<Announcement,ApplicationUser?>
        .FirstOrDefaultAsync(x :Announcement => x.Id == id); // Task<Announcement?>
    if (announcement == null) return NotFound();

    return Ok(MapToResponseAnnouncementDto(announcement));
}

[Authorize(Roles = "Admin,Teacher")]
[HttpPut( template: "{id}")]
Meleshko Danylo *
public async Task<IActionResult> EditAnnouncement(string? id, [FromBody] EditAnnouncementDto? editedAnnouncement)
{
    if(editedAnnouncement == null) return BadRequest();

    var announcement = await _db.Announcements.FindAsync(id);
    if (announcement == null) return NotFound();

    announcement.Title = editedAnnouncement.Title;
    announcement.Text = editedAnnouncement.Text;
    announcement.WithComments = editedAnnouncement.WithComments;
    announcement.IsEdited = true;
    if (editedAnnouncement.GroupVisibility != null) announcement.GroupVisibility = editedAnnouncement.GroupVisibility;

    _db.Announcements.Update(announcement);
    await _db.SaveChangesAsync();

    return Ok();
}

[Authorize(Roles = "Admin,Teacher")]
[HttpDelete( template: "{id}")]
Meleshko Danylo
public async Task<IActionResult> DeleteAnnouncement(string id)
{
    var announcement = await _db.Announcements.FindAsync(id);
    if (announcement == null) return NotFound();

    _db.Announcements.Remove(announcement);
    await _db.SaveChangesAsync();

    return Ok();
}
```

Рисунок 2.21 – Endpoints для отримання оголошення за його id, редагування оголошення та видалення оголошення

```
[Authorize(Roles = "Admin, Teacher")]
[HttpPost]
Meleshko Danylo *
public async Task<ActionResult<ResponseAnnouncementDto>> AddAnnouncement([FromBody] AddAnnouncementDto? newAnnouncement)
{
    if (newAnnouncement == null) return BadRequest();

    var userId :string? = User.FindFirstValue( claimType: ClaimTypes.NameIdentifier);
    if (userId == null) return Forbid();

    var author :ApplicationUser? = await _db.Users.FindAsync(userId);
    if (author == null) return BadRequest( error: "Author is not exist");

    var announcement = new Announcement
    {
        Id = newAnnouncement.Id!,
        Title = newAnnouncement.Title,
        Text = newAnnouncement.Text,
        CreatedAt = DateTime.UtcNow,
        AuthorId = userId,
        Author = author,
        WithComments = newAnnouncement.WithComments,
        GroupVisibility = newAnnouncement.GroupVisibility,
        AmountOfComments = 0
    };

    List<ApplicationUser> users = new();
    if (announcement.GroupVisibility.Contains("*")) {
        users = await _db.Users.AsNoTracking().Where(user => user.Email != author.Email).ToListAsync();
    }
    else {
        var query :IQueryable<ApplicationUser> = _db.Users // DbSet<ApplicationUser>
            .AsNoTracking()
            .Where(user => announcement.GroupVisibility.Any(group :string => user.Email != author.Email
                                                    && user.AnnouncementsNotifications));
        if (newAnnouncement.IsForStarostas) query = query.Where(user => user.Role == "Starosta"
            && announcement.GroupVisibility.Contains(user.Group)); // IQueryable<ApplicationUser>
        else query = query.Where(user => announcement.GroupVisibility.Contains(user.Group));

        var relevantUsers :List<ApplicationUser> = await query.ToListAsync();
        foreach (var user in relevantUsers)
        {
            if (user.Email != null) users.Add(user);
        }
    }

    var emailTasks :List<Task> = users.Select(user => _emailSender.SendEmailAsync(user.Email, subject: announcement.Title, message: announcement.Text))
        .ToList();
    await Task.WhenAll(emailTasks);

    await _db.Announcements.AddAsync(announcement);
    await _db.SaveChangesAsync();

    return CreatedAtAction(nameof(GetAnnouncement), routeValues: new { id = announcement.Id }, MapToResponseAnnouncementDto(announcement));
}
```

Рисунок 2.22 – Endpoint для створення оголошення, при створенні якого, обрані користувачі отримають оголошення на електронну пошту

```
[Authorize(Roles = "Admin,Teacher")]
[HttpGet]
Meleshko Danylo *
public async Task<IActionResult> GetAllAnnouncementsOfTheUser([FromQuery] int page,
                                    [FromQuery] string? title, [FromQuery] DateTime?
                                    dateOfCreation, CancellationToken ct)
{
    const int pageLimit = 5;
    int announcementsSkiped = (page - 1) * pageLimit;
    var userId :string? = User.FindFirstValue( claimType: ClaimTypes.NameIdentifier);
    if (userId == null) return Forbid();

    var query :IQueryable<Announcement> = _db.Announcements // DbSet<Announcement>
        .Include( navigationPropertyPath: a :Announcement => a.Author).AsQueryable();
    query = query.Where(x :Announcement => x.AuthorId == userId);

    if (!string.IsNullOrWhiteSpace(title)) {
        query = query.Where(a :Announcement => a.Title.Contains(title));
    }

    if (dateOfCreation.HasValue) {
        query = query.Where(a :Announcement => a.CreatedAt <= dateOfCreation.Value);
    }

    var totalAnnouncements :int = await query.CountAsync(ct);
    var totalPages = (int)Math.Ceiling(totalAnnouncements / (double)pageLimit);

    var announcements :List<Announcement> = await query
        .AsNoTracking() // IQueryable<Announcement>
        .OrderByDescending(x :Announcement => x.CreatedAt) // IOrderedQueryable<Announcement>
        .Skip(announcementsSkiped)
        .Take(pageLimit) // IQueryable<Announcement>
        .ToListAsync(ct); // Task<List<...>>

    return Ok(new {
        announcements = MapToResponseAnnouncementsDto(announcements),
        totalPages = totalPages
    });
}
```

Рисунок 2.23 – Endpoint для отримання всіх оголошень зроблених користувачем

```
[HttpPut]
[Authorize]
Meleshko Danylo
public async Task<IActionResult> ToggleAnnouncementNotifications()
{
    var userId:string? = User.FindFirstValue(claimType: ClaimTypes.NameIdentifier);
    if(userId == null) return Forbid();

    var user = await _userManager.Users.FirstOrDefaultAsync(x:ApplicationUser => x.Id == userId);
    if(user == null) return Forbid();

    user.AnnouncementsNotifications = !user.AnnouncementsNotifications;
    await _userManager.UpdateAsync(user);

    return Ok();
}

[HttpGet]
[Authorize]
Meleshko Danylo
public async Task<ActionResult<bool>> IsAnnouncementsNotificationsEnabled()
{
    var userId:string? = User.FindFirstValue(claimType: ClaimTypes.NameIdentifier);
    if(userId == null) return Forbid();

    var user = await _userManager.Users.AsNoTracking().FirstOrDefaultAsync(x:ApplicationUser => x.Id == userId);
    if(user == null) return Forbid();

    return Ok(user.AnnouncementsNotifications);
}
```

Рисунок 2.24 – Endpoints для перевірки, чи включені сповіщення у користувача і включення та виключення сповіщень

```
[Authorize]
[HttpGet(template: "{id}")]
Meleshko Danylo *
public async Task<ActionResult<bool>> IsLiked(string id)
{
    var userId:string? = User.FindFirstValue(claimType: ClaimTypes.NameIdentifier);
    if(userId == null) return Forbid();

    var like = await _db.Likes.AsNoTracking() // IQueryable<Like>
        .FirstOrDefaultAsync(x:Like => x.UserId == userId && x.LikeFor == id); // Task<Like?>
    if(like == null) return Ok(false);

    return Ok(true);
}
```

Рисунок 2.25 – Endpoint для перевірки, чи залишив користувач реакцію під оголошенням, дописом чи коментарем

```
[Authorize]
[HttpPost]
Meleshko Danylo *
public async Task<IActionResult> ChangeGroup([FromQuery] string newGroup, [FromQuery] string isStarosta)
{
    var userId :string? = User.FindFirstValue( claimType: ClaimTypes.NameIdentifier);
    if(userId == null) return Unauthorized();

    var user = await _userManager.Users.FirstOrDefaultAsync(x :ApplicationUser => x.Id == userId);
    if(user == null) return NotFound();

    if (user.NextGroupChangingDateAvailable.HasValue && user.NextGroupChangingDateAvailable.Value > DateTime.Now) {
        return BadRequest();
    }

    var userRoles :IList<string> = await _userManager.GetRolesAsync(user);
    if (newGroup == "Teacher")
    {
        if (!Teachers.TeacherEmails.Contains(user.Email.Trim()))
            return Forbid();

        user.Role = "Teacher";
        await _userManager.RemoveFromRoleAsync(user, userRoles[0]);
        await _userManager.AddToRoleAsync(user, role: "Teacher");
    }
    else if (newGroup != "Teacher" && isStarosta == "true") {
        user.Role = "Starosta";
        await _userManager.RemoveFromRoleAsync(user, userRoles[0]);
        await _userManager.AddToRoleAsync(user, role: "Starosta");
    }
    else {
        if(user.Email.Split("@")[1] != "nure.ua") return Forbid();
        user.Role = "Student";
        await _userManager.RemoveFromRoleAsync(user, userRoles[0]);
        await _userManager.AddToRoleAsync(user, role: "Student");
    }

    user.NextGroupChangingDateAvailable = DateTime.UtcNow.AddDays(7);
    await _userManager.UpdateAsync(user);
    await _db.SaveChangesAsync();

    return Ok();
}
```

Рисунок 2.26 – Endpoint для зміни групи чи ролі користувача

2.4 Розробка клієнтської частини

2.4.1 API запити

Для отримання даних, браузер має надавати запити до сервера, використовуючи endpoints. Щоб створити запити, можна скористатись функцією JavaScript (fetch). Вказавши ip адресу, порт, метод який використовується запитом (get, put, post, delete), заголовки та можливо дані які передаються через запит, браузер створить та направить запит на вказану ip адресу.

В цьому проєкті, для спрощеного створення запитів, було встановлено популярний пакет Axios. Окрім того, що Axios простіше за стандартний fetch, цей пакет також дозволяє створити екземпляр Axios. Створивши екземпляр Axios, можна задати базовий URL (Uniform Resource Locator), заголовки, чи будуть використовуватись http only cookie, що робити коли сервер відправляє код помилки (рис.2.27). Користуючись створеним екземпляром для відправлення запитів до сервера, кожен запит буде мати вказані налаштування (шлях до серверу, Authorization заголовок з Jwt токеном, відправлення http only cookie разом з запитом).

```
const axiosInstance : AxiosInstance = axios.create({
    baseURL: process.env.REACT_APP_API_PATH,
    withCredentials: true,
    headers: { "Authorization": `Bearer ${token}`}
});

axiosInstance.interceptors.response.use(
    Doc Comment  Unit Tests
    (response : AxiosResponse<any, any> ) : AxiosResponse<any, any> => response,
    Doc Comment  Unit Tests
    async (error : any ) : Promise<AxiosResponse<any, ... => {
        if (error.response?.status === 401 && !error.config._retry) {
            error.config._retry = true;
            setLoading(true);
            try {
                const response : AxiosResponse<any, any> = await axios.post(`${process.env.REACT_APP_API_PATH}/Auth/RefreshToken`, null, {
                    withCredentials: true,
                });
                error.config.headers["Authorization"] = `Bearer ${response.data.token}`;
                setToken(response.data.token);
                return axiosInstance(error.config); // Retry original request
            } catch(error) {
                window.location.assign("/LogIn")
                setToken(null);
                setUser({
                    id: "",
                    email: "",
                    group: "",
                    role: ""
                });
                console.error(error);

                return Promise.reject(error); // Refresh failed, reject request
            }
            finally {
                setLoading(false);
            }
        }
        return Promise.reject(error);
    }
);
```

Рисунок 2.27 – Створення та налаштування екземпляру Axios

Скориставшись створеним екземпляром, було створено методі для взаємодії з API endpoints (рис.2.28 – 2.31).

```
async function getAnnouncements(filters?: Filters) :Promise<void> {
    // Get announcements from API
    try {
        filters = filters || {title: "", authorEmail: "", createdAt: new Date(), authorRole: ""};
        const queryParams = new URLSearchParams({
            page: page.toString(),
            ...(filters.title && { title: filters.title }),
            ...(filters.authorEmail && { authorEmail: filters.authorEmail }),
            ...(filters.createdAt && { dateOfCreation: filters.createdAt.toISOString() }),
            ...(filters.authorRole && { authorRole: filters.authorRole })
        });

        let response :any = await axiosInstance.get(`/Announcements/GetAllAnnouncements?${queryParams}`);

        setAnnouncements(response.data.announcements);
        setTotalPages(response.data.totalPages);
    }
    catch(error) {
        console.error("Faild to get announcements", error);
    }
}
```

Рисунок 2.28 – Метод для отримання оголошень

Для створення query параметрів, які необхідні для фільтрації даних, було використано метод URLSearchParams(). В якості параметрів, використовуються дані які отримуються з фільтрів, та передаються до методу getAnnouncements() в якості параметра filters типу Filters.

Після отримання відповіді від серверу, відбувається оновлення даних в змінних. Для створення цих змінних було використано React hook, useState(). useState() дозволяє асинхронно оновлювати змінні, а також проводить повторний рендер сторінки. Для оновлення значення змінною, що використовує useState(), використовується set метод (setAnnouncements, setTotalPages).

```
async function getAnnouncement(announcementId: string) : Promise<void> {
    // Get announcement from API
    try {
        let response : any  = await axiosInstance.get(`/Announcements/GetAnnouncement/${announcementId}`);

        setAnnouncement(response.data);
    }
    catch (error) {
        console.error("Faild to get announcement", error);
    }
}

Doc Comment  Unit Tests  Show usages  Meleshko Danylo *
async function addAnnouncement(announcement: AnnouncementAddDto) : Promise<void>  {
    // Add announcement to API
    try {
        let response : any  = await axiosInstance.post(`/Announcements/AddAnnouncement`, announcement);

        setAnnouncements(prevAnnouncements : Announcement[]  => [...prevAnnouncements, response.data]);
        toast.success("Announcement added successfully.");
    }
    catch(error) {
        toast.error("Fail to add announcement");
        console.error("Failed to add announcement", error);
    }
}
```

Рисунок 2.29 – Методи для отримання оголошення та створення нового оголошення

```
async function editAnnouncement(announcementId: string, updatedAnnouncement: AnnouncementEditDto) : Promise<void>  {
    // Edit announcement in API
    try {
        let response : any  = await axiosInstance.put(`/Announcements/EditAnnouncement/${announcementId}`, updatedAnnouncement,
        );

        setAnnouncements(announcements.map(p : Announcement  => p.id === announcementId ? response.data : p));
        toast.success("Announcement edited successfully.");
    } catch (error) {
        toast.error("Fail to edit announcement");
        console.error("Failed to edit announcement", error);
    }
}

Doc Comment  Unit Tests  Show usages  Meleshko Danylo
async function deleteAnnouncement(announcementId: string) : Promise<void>  {
    // Delete announcement from API
    try {
        await axiosInstance.delete(`/Announcements/DeleteAnnouncement/${announcementId}`,

        );

        setAnnouncements(prevAnnouncements : Announcement[]  => prevAnnouncements.filter(p : Announcement  => p.id !== announcementId));
        setAnnouncementsOfTheUser(prev => prev.filter(p => p.id !== announcementId));
        toast.success("Announcement deleted successfully.");
    }
    catch (error) {
        toast.error("Fail to delete announcement");
        console.error("Failed to delete announcement", error);
    }
}
```

Рисунок 2.30 – Методи для редагування оголошення та видалення оголошення

```
async function allAnnouncementsOfTheUser(filters?: Filters) {
    try {
        filters = filters || {title: "", authorEmail: "", createdAt: new Date(), authorRole: ""};
        const queryParams = new URLSearchParams({
            page: page.toString(),
            ...(filters.title && { title: filters.title }),
            ...(filters.createdAt && { dateOfCreation: filters.createdAt.toISOString() }),
        });

        let response = await axiosInstance.get(`/Announcements/GetAllAnnouncementsOfTheUser?${queryParams}`, {

        });

        setAnnouncementsOfTheUser(response.data.announcements);
        setTotalPages(response.data.totalPages);
    }
    catch (e) {
        console.error("Failed to get your announcements", e);
    }
}

Doc Comment  Unit Tests  Show usages  Meleshko Danylo
async function toggleLike(announId: string) {
    try {
        let response = await axiosInstance.put(`/Announcements/ToggleLike/${announId}/Like`, null,)

        setAnnouncements(prev => prev.map(a => a.id === announId ? response.data : a));
        setAnnouncement(response.data);
    }
    catch(error) {
        console.error("Failed to toggle like", error);
    }
}
```

Рисунок 2.31 – Методи для отримання оголошень створених користувачем та додавання та прибирання реакції з оголошення

Такі ж самі методи були створені для дописів та коментарів.

Окрім методів отримання, додавання, редагування та видалення оголошень, дописів та коментарів, були також створено методи для перевірки активації сповіщень на пошту, активація та деактивація сповіщень, а також метод для запиту на створення нового Jwt токена (рис.2.32 – 2.33).

```
async function isNotificationsEnabled(){
    try {
        const response = await axiosInstance.get(`/Features/IsAnnouncementsNotificationsEnabled`);

        setIsNotificationsActive(response.data);
    }
    catch (e) {
        console.error("Failed to get notifications enabled", e);
    }
}

Doc Comment  Unit Tests  Show usages  Meleshko Danylo
async function toggleNotifications(){
    try {
        await axiosInstance.put(`/Features/ToggleAnnouncementNotifications`, null);

        setIsNotificationsActive(prev => !prev);
    }
    catch (error) {
        console.error("Failed to toggle notifications", error);
    }
}
```

Рисунок 2.32 – Методи для перевірки активації сповіщень та активації та деактивації сповіщень

```
async function refreshToken() : Promise<void> {
    try {
        const response : AxiosResponse<any, any> = await axios.post(`${process.env.REACT_APP_API_PATH}/Auth/RefreshToken`, null,
            {withCredentials: true});
        setToken(response.data.token);
        setUser({
            id: response.data.id,
            email: response.data.email,
            group: response.data.group,
            role: response.data.role ?? ""
        });
        setRole(response.data.role);
    } catch (error) {
        setToken(null);
    } finally {
        setLoading(false);
    }
}
```

Рисунок 2.33 – Методи для отримання оголошень створених користувачем та додавання та прибирання реакції з оголошення

Були створені методи для надсилання запитів для реєстрації нового користувача, входу та виходу з системи, отримання токена для зміни пароля та зміни пароля (рис.2.34 – 2.37).

```
async function handleLogout() : Promise<void> {
    try {
        await axios.post(`${process.env.REACT_APP_API_PATH}/Auth/LogOut`, null,
            {withCredentials: true});
        setToken(null);
        setUser({
            id: "",
            email: "",
            group: "",
            role: ""
        })
    }
    catch(error) {
        console.error(error);
    }
}
```

Рисунок 2.34 – Метод для виходу з системи

```
async function handleForgotPassword(event: React.FormEvent) : Promise<void> {
    event.preventDefault();
    setErrorMessage(e : ErrorMessage => ( { ...e, isError: false }))

    try {
        const response : AxiosResponse<any, any> = await axios
            .post(`${process.env.REACT_APP_API_PATH}/Auth/ForgotPassword?email=${email}`);

        if(response.status === 200) {
            window.location.assign(`/LogIn`);
        }
    }
    catch (error) {
        setErrorMessage(e : ErrorMessage => ( { ...e, isError: true }))
        console.error(error);
    }
}
```

Рисунок 2.35 – Метод для отримання токену на зміну пароля

```
async function handleLogin(event: React.FormEvent) : Promise<void> {
    event.preventDefault();

    try {
        const credentials = {
            "email": email,
            "password": password
        };

        const response : any = await axiosInstance.post(`/Auth/LogIn`, credentials,
            {withCredentials: true});

        const userInfo = {
            id: response.data.id,
            email: response.data.email,
            group: response.data.group,
            role: response.data.role
        }

        if(response.status === 200) {
            setToken(response.data.token);
            setUser(userInfo);
            window.location.assign(`Announcements`);
        }
    } catch (err) {
        setErrorResult(e : ErrorMessage => ({
            ...e,
            isError: true,
        }))
        console.error("Login failed:", err);
    }
}
```

Рисунок 2.36 – Метод для входу в систему

```
async function handleResetPassword(event: React.FormEvent) : Promise<void> {
    event.preventDefault();
    setErrorResult(e : ErrorMessage => ( { ...e, isError: false }))

    if(form.confirmPassword !== form.password) {
        setErrorPassword(e : ErrorMessage => ({
            ...e,
            isError: true,
        }))
        return;
    }
    else setErrorPassword(e : ErrorMessage => ({...e, isError: false}));

    try {
        const credentials = {
            "email": searchParams.get("email"),
            "newPassword": form.password,
            "resetToken": decodeURIComponent(searchParams.get("token") || "")
        };

        const response : AxiosResponse<any, any> = await axios.post(`${process.env.REACT_APP_API_PATH}/Auth/ResetPassword`,
            credentials);

        if(response.status === 200) {
            window.location.assign(`/LogIn`);
        }
    } catch (err) {
        setErrorResult(e : ErrorMessage => ({
            ...e,
            isError: true
        }))
        console.error("Login failed:", err);
    }
}
```

Рисунок 2.37 – Метод для зміни пароля

Метод для реєстрації нових користувачів:

```
async function handleRegistration(e: React.FormEvent) {
e.preventDefault();
if(!emailRegex.test(form.email) && passwordRegex.test(form.password)){
  setErrors(prev => ({
    ...prev,
    email: true,
    password: false,
    isError: false
  }));
```

```
    return;
}

if(!passwordRegex.test(form.password) && emailRegex.test(form.email)){
    setErrors({
        ...errors,
        password: true,
        email: false,
        isError: false
    });
    return;
}

if(!emailRegex.test(form.email) && !passwordRegex.test(form.password)){
    setErrors({
        ...errors,
        password: true,
        email: true,
        isError: false
    });
    return;
}

setErrors(prev => ({
    ...prev,
    password: false,
    email: false,
    isError: false
}));
```

```
try {
    let credentials;

    if(teacherBox.isPressed){
        credentials = {
            "Email": form.email,
            "Password": form.password,
            "Group": teacherBox.group
        };
    }
    else if(starostaBox.isPressed){
        credentials = {
            "Email": form.email,
            "Password": form.password,
            "Group": form.group,
            "IsStarosta": true
        };
    }
    else {
        credentials = {
            "Email": form.email,
            "Password": form.password,
            "Group": form.group
        };
    }

    await axios.post(`${process.env.REACT_APP_API_PATH}/Auth/Register`,
credentials);

    if(registerRef.current) registerRef.current.classList.add("no-display");
```

```
        if(confirmRef.current) confirmRef.current.classList.remove("no-display");
    }
    catch(e: any) {
        if(e.response.status === 403){
            setErrors(prev => ({
                ...prev,
                message: "Ви не можете обрати цей факультет",
                isError: true
            }));
        }
        else{
            setErrors(prev => ({
                ...prev,
                message: "Пошта чи пароль неправильні, або користувач з цими даними вже існує",
                isError: true
            }));
        }
        console.error("Login failed:", e);
    }
}
```

Метод проводить валідацію облікових даних перед тим як надіслати їх до сервера. Якщо дані відповідають очікуванням, то потім, в залежності від прапорців, готується Json об’єкт з необхідними даними, та відправляється до сервера.

Для взаємодії з даними, отриманими з сервера, були створені абстракції (рис.2.38 – 2.40).

```
export type Announcement = {
    id: string,
    title: string,
    text: string,
    createdAt: Date,
    authorId: string,
    authorEmail: string,
    authorImage: string,
    authorRole: string,
    likes: number,
    amountOfComments: number,
    isEdited: boolean,
    withComments: boolean,
    groupVisibility: string[]
}

Doc Comment  Unit Tests
export type AnnouncementAddDto = {
    id: string,
    title: string,
    text: string,
    authorId: string,
    withComments: boolean,
    groupVisibility: string[],
    isForStarostas: boolean
}

Doc Comment  Unit Tests
export type AnnouncementEditDto = {
    title?: string,
    text?: string,
    withComments?: boolean,
    groupVisibility?: string[]
}
```

Рисунок 2.38 – Абстракції для оголошень

```
export type Post = {
    id: string,
    title: string,
    text: string,
    createdAt: Date,
    authorId: string,
    authorEmail: string,
    authorImage: string,
    authorRole: string,
    likes: number,
    amountOfComments: number,
    isEdited: boolean,
    withComments: boolean,
    groupVisibility: string[]
}

Doc Comment  Unit Tests
export type PostAddDto = {
    id: string,
    title: string,
    text: string,
    authorId: string,
    withComments: boolean,
    groupVisibility: string[]
}

Doc Comment  Unit Tests
export type PostEditDto = {
    title?: string,
    text?: string,
    withComments?: boolean,
    groupVisibility?: string[]
}
```

Рисунок 2.39 – Абстракції для дописів

```
export type Comment = {
    id: string,
    text: string,
    createdAt: Date,
    postId: string,
    authorId: string,
    authorEmail: string,
    authorImage: string,
    authorRole: string,
    likes: number,
    replyTo: string,
    repliesToComment: Comment[],
}

Doc Comment  Unit Tests
export type CommentAddDto = {
    id: string;
    text: string;
    authorId: string;
    postId: string;
    replyTo?: string;
}
```

Рисунок 2.40 – Абстракції для коментарів

2.4.2 Налаштування навігації між сторінками

Для налаштування навігації між сторінками, було використано router пакет, який дозволяє легко виконати цю задачу.

Після встановлення пакета, необхідно перейти в index.tsx файл, імпортувати необхідні компоненти:

import {Route, BrowserRouter as Router, Routes} from 'react-router-dom';

Скориставшись імпортованими компонентами, було створено навігацію між сторінками (рис.2.41).

```
<Router>
    <Routes>
        <Route path='/ForgotPassword' element={<ForgotPassword/>}/>
        <Route path='/ChangePassword' element={<ChangePassword/>}/>
        <Route path='/Register' element={<Register/>}/>
        <Route path='/' element={<AuthValidation><Announcements/></AuthValidation>}/>
        <Route path='/Announcements' element={<Announcements/>}/>
        <Route path='/Announcements/EditAnnouncement/:announId' element={<EditAnnouncement/>}/>
        <Route path='/LogIn' element={<LogIn/>}/>
        <Route path='/Forum' element={<Forum/>}/>
        <Route path={`/Announcement/:announcementId`} element={<Announcement/>}/>
        <Route path='/OwnPosts' element={<OwnPosts/>}/>
        <Route path='/OwnPosts/EditPost/:postId' element={<EditPost/>}/>
        <Route path='/AddPost' element={<AddPost/>}/>
        <Route path='/Forum/Post/:postId' element={<Post/>}/>
        <Route path='OwnAnnouncements' element={<OwnAnnouncements/>}/>
        <Route path='AddAnnouncement' element={<AddAnnouncement/>}/>
    </Routes>
</Router>
```

Рисунок 2.41 – Створена навігація між сторінками за допомогою пакету router

Ключовим компонентом для налаштування навігації – є Route. Route має два ключові атрибути: path та element. Path задає шлях за яким браузер буде проводити рендер компонента, який заданий в атрибуті element.

Рендер – створення зображення або відео з опису, за допомогою комп’ютерної програми [20].

2.4.3 Створення компонентів та сторінок

Дизайн сайту для кожної сторінки можна поділити на декілька компонентів, за виключенням сторінок для реєстрації, входу в систему та зміни паролю (мають лише форму для заповнення та задній фон). Кожна сторінка містить header (блок для навігації на сторінці, розміщується зверху), footer (блок з посиланнями на сторонні ресурси та сторінки сайту, розміщується знизу) та блок, наповнення якого змінюється в залежності від сторінки (сторінка з оголошеннями, сторінка для додавання дописів, сторінка з деталями допису).

Першим кроком було створено header та footer, бо обидва компоненти використовуються частіше за інші, та не змінні незалежно від сторінки (рис.2.42 – 2.43).

```
export default function Navbar() : Element  {
    const {handleLogout, role} = useAuthContext();

    return (
        <>
            <header className="Header">
                <div className="Header-left">
                    <Link id="NureIconBottom" to="/"><img alt="" src="/images/LogoHeader.png"/></Link>
                </div>
                <div className="Header-center">
                    <Link to="/Announcements">Оголошення</Link>
                    <Link to="/Forum">Форум</Link>
                    <Link to="/OwnPosts">Власні Пости</Link>
                </div>
                <div className="Header-right">
                    <Link id="LogInButton" onClick={handleLogout} to="/Login">Вийти</Link>
                    <Link id="RegisterButton" to="/Register">Реєстрація</Link>
                </div>
            </header>
        </>
    )
}
```

Рисунок 2.42 – Header сайту

```
export default function Footer() : Element  {
    const {role} = useAuthContext();

    return (
        <div className="Footer">
            <div className="FooterBody">
                <div className="FooterUp">
                    <div className="FooterLeft">
                        <Link id="NureIconBottom" to="/*"><img alt="" src="/images/LogoFooter.png"/></Link>
                    </div>
                    <div className="FooterRightUp">
                        <Link to="/Announcements">Оголошення</Link>
                        <Link to="/Forum">Форум</Link>
                        <Link to="/OwnPosts">Власні Пости</Link>
                    </div>
                </div>

                <div className="FooterDown">
                    <div className="FooterDownLeft">
                        <a className="Footer-animated-button" href="https://www.facebook.com/nureKharkiv/">
                            <img alt="" src="/images/IconF.png"/></a>
                        <a className="Footer-animated-button" href="https://www.instagram.com/khnure_official/">
                            <img alt="" src="/images/IconI.png"/></a>
                        <a className="Footer-animated-button" href="https://www.youtube.com/user/nuretv">
                            <img alt="" src="/images/IconY.png"/></a>
                        <a className="Footer-animated-button" href="https://www.linkedin.com/school/kharkiv-national-university-of-radioelectronics/">
                            <img alt="" src="/images/IconL.png"/></a>
                        <a className="Footer-animated-button" href="https://twitter.com/PressNURE">
                            <img alt="" src="/images/IconT.png"/></a>
                    </div>
                    <div className="FooterDownRight">
                        <div className="DivMonImgDown">
                            <a href="http://mon.gov.ua/"><img alt="" src="/images/LogoUAMON.png"/></a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Рисунок 2.43 – Footer сайту

Якщо не рахувати сторінки для реєстрації, входу в систему та зміни пароля, то всього є 12 сторінок: створити оголошення/допис, відредагувати оголошення/допис, стрічка з оголошеннями/дописами, власні оголошення/дописи та деталі про оголошення/допис.

Сторінки які відповідають за зображення списку оголошень або дописів, можна поділити на чотири сегменти: сегмент для заголовка сторінки, список оголошень/дописів, набір фільтрів, а також опція зміни ролі/групи, секція для кнопки виключення та включення сповіщень (присутня лише на сторінках пов'язаними з оголошеннями). Список оголошень/дописів, містить список блоків, які відповідають за їх зображення в компактному вигляді. Тому, цю сторінку можна розбити на наступні компоненти: список оголошень/дописів, набір фільтрів та блок для репрезентації оголошення/допису.

Для формування списку оголошень/дописів, робиться запит до сервера. Після отримання даних з сервера, відбувається фільтрація по отриманому списку. Для кожного оголошення/допису створюється окремий компонент, який відповідає за зображення кожного окремого оголошення/допису в списку на сторінці (рис.2.44 – 2.45).

```
{announcements.map(announcement : Announcement => (
    <div key={announcement.id}>
        <AnnouncementPostBox item={announcement} type={"Announcement"} />
    </div>
))}
```

Рисунок 2.44 – Зображення отриманих з серверу оголошень

```
export default function AnnouncementPostBox({item, type}: {item: Post | Announcement, type: string}) :Element {

    return(
        <>
            <div className="AnnounBody">
                <div className="AnnounHeader">
                    <div className="AnnounHeaderAuthor">
                        <div className="AnnounAuthorEmail">
                            <p>{item.authorEmail}</p>
                        </div>
                        <div className="AnnounAuthorRole">
                            <p id="AnnounAuthorRole">{item.authorRole}</p>
                        </div>
                    </div>
                    <div className="AnnounHeaderCommLikes">
                        <div>{dateFormat.format(new Date(item.createdAt))}</div>
                        <div>
                            <div><img alt="" src="/images/LikeIcon.png"/>{item.likes}</div>
                            <div><img alt="" src="/images/CommentIcon.png"/>{item.amountOfComments}</div>
                        </div>
                    </div>
                </div>
                <svg xmlns="http://www.w3.org/2000/svg" width="860" height="2" viewBox="0 0 860 2"
                     fill="none">
                <path d="M0 1H860" stroke="#1DD6F3" strokeLinecap="round" strokeLinejoin="round"
                          strokeDasharray="6 6"/>
                </svg>
                <div className="AnnounTextBody">
                    <h4>{item.title}</h4>
                    <div className="AnnounContent">
                        <p>{item.text}</p>
                    </div>
                </div>
                <div className="AnnounFooter">
                    <Link className="AnnounAndPostBox-LinkAnimation" to= {type === "Post" ? `/Forum/Post/${item.id}` : `/Announcement/${item.id}`}
                          state={{post: item}}>Детальніше...</Link>
                </div>
            </div>
        </>
    )
}
```

Рисунок 2.45 – Компонент для зображення оголошення/допису у вигляді компактного блоку

Під створеним списком розміщені кнопки для перегортання сторінок (рис.2.46).

```
<div className="PagesButtonsWrapper">
    <button className="NextAndBackPage-Button BigButtonHoverAnimation" onClick={() :void => {
        if(page !== 1) setPage(prev :number => prev - 1)
    }}>Back</button>
    <span>{page} з {totalPages}</span>
    <button className="NextAndBackPage-Button BigButtonHoverAnimation" onClick={() :void => {
        if(page !== totalPages) setPage(prev :number => prev + 1)
    }}>Next</button>
</div>
```

Рисунок 2.46 – Сегмент з кнопками для перегортання сторінок

По праву сторону від списку з оголошеннями/дописами, розміщений сегмент з фільтрами (рис.2.47). Доступне фільтрування за заголовком, поштовою адресою автора, датою створення та роллю автора.

```
<div className="AnnounAndPostFilterBox">
    <div className="InputAndIconFilter">
        <img alt="" className="InputIcon" src="/images/SearchIcon.png"/>
        <input type="text" placeholder="Пошук" className="AnnounAndPostFilterInput"
               onChange={(e : ChangeEvent<HTMLInputElement... ) : void  => SearchByTitle(e)}/>
    </div>
</div>
<svg xmlns="http://www.w3.org/2000/svg" width="420" height="2" viewBox="0 0 420 2" fill="none">
    <path d="M0 1H420" stroke="#1DD6F3" strokeLinecap="round" strokeLinejoin="round"
          strokeDasharray="6 6"/>
</svg>
{!ownAnnouns &&
    <div className="AnnounAndPostFilterBox">
        <label htmlFor="Автор">Автор</label><br/>
        <div className="InputAndIconFilter">
            <img alt="" className="InputIcon" src="/images/SearchIcon.png"/>
            <input id="Автор" type="text" placeholder="Викладач" className="AnnounAndPostFilterInput"
                   onChange={(e : ChangeEvent<HTMLInputElement... ) : void  => SearchByAuthor(e)} />
        </div>
    </div>
}
<div className="AnnounAndPostFilterBox">
    <label htmlFor="ДатаСтворення">Дата створення</label>
    <select id="ДатаСтворення" className="AnnounAndPostFilterInput"
            onChange={(e : ChangeEvent<HTMLSelectElemen... ) : void  => SearchByDate(e.target.value)}>
        <option value="Будь-коли">Будь-коли</option>
        <option value="3 Вчора">3 Вчора</option>
        <option value="3 дні тому і більше">3 дні тому і більше</option>
        <option value="Тиждень тому і більше">Тиждень тому і більше</option>
        <option value="Місяць тому і більше">Місяць тому і більше</option>
        <option value="Рік тому і більше">Рік тому і більше</option>
        <option value="3 роки тому і більше">3 роки тому і більше</option>
    </select>
</div>
{!ownAnnouns &&
    <div className="AnnounAndPostFilterBox">
        <label htmlFor="АвториПостів">Автори постів</label>
        <select onChange={(e : ChangeEvent<HTMLSelectElemen... ) : void  => SearchByRole(e.target.value)} id="АвториПостів"
                className="AnnounAndPostFilterInput">
            <option value="Будь-хто">Будь-хто</option>
            <option value="Вчитель">Вчитель</option>
            <option value="Студент">Студент</option>
        </select>
    </div>
}
```

Рисунок 2.47 – Сегмент з фільтрами для списку оголошень/дописів

Під фільтрами знаходяться дві кнопки. Перша кнопка переводить на сторінку зі своїми оголошеннями, де також можна перейти на сторінки для створення та редагування оголошень, але ця кнопка відображається лише користувачам, які мають роль вчителя. Друга кнопка, відповідає за можливість змінити роль/групу (рис.2.48).

```
<div className="AddPostOrAnnounButtonBox">
    <h2 className="AddPostOrAnnounButtonH2">Змінити факультет/роль (раз на тиждень)</h2>
    <svg xmlns="http://www.w3.org/2000/svg" width="420" height="2" viewBox="0 0 420 2"
         fill="none">
        <path d="M0 1H420" stroke="#1DD6F3" strokeLinecap="round" strokeLinejoin="round"
              strokeDasharray="6 6"/>
    </svg>
    {error.message && (
        <span className="AuthErrorSpan" style={{maxWidth: "85%", alignSelf: "center"}}>{error.message}</span>
    )}
    <div style={{
        width: "70%", display: "flex", justifyContent: "center",
        alignItems: "center", flexDirection: "column", gap: "10px"
    }}>
        <select name="group" className="GroupChangeSelect"
                value={group}
                onChange={(e : ChangeEvent<HTMLSelectElemen... ) : void => setGroup(e.target.value)}>
            {GroupVisibilityArray.map(g : GroupVisibility => (
                <option key={g.id} value={g.group}>
                    {g.group}
                </option>
            ))}
            <option value={"Teacher"}>Викладач</option>
        </select>
        <input style={{width: "15px"}} name="starostaBox" type="checkbox"
               onClick={() : void => {
                   setStarostaBox(prev : { isPressed: boolean; group.... => ({...prev, isPressed: !prev.isPressed}));
               }}/>
        <label htmlFor="starostaBox">Староста</label>
    </div>
    <button className="AddPostOrAnnounButton BigButtonHoverAnimation"
            style={{width: "70%", alignSelf: "center"}}
            onClick={() : Promise<void> => handleClick()}
    >Підтвердити</button>
</div>
```

Рисунок 2.48 – Кнопка для зміни ролі/групи

Праворуч від фільтрів, знаходиться кнопка для включення та виключення сповіщень. Об’єднавши все до копи та додавши CSS стилі, React згенерує сторінки (рис.2.49 – 2.51).

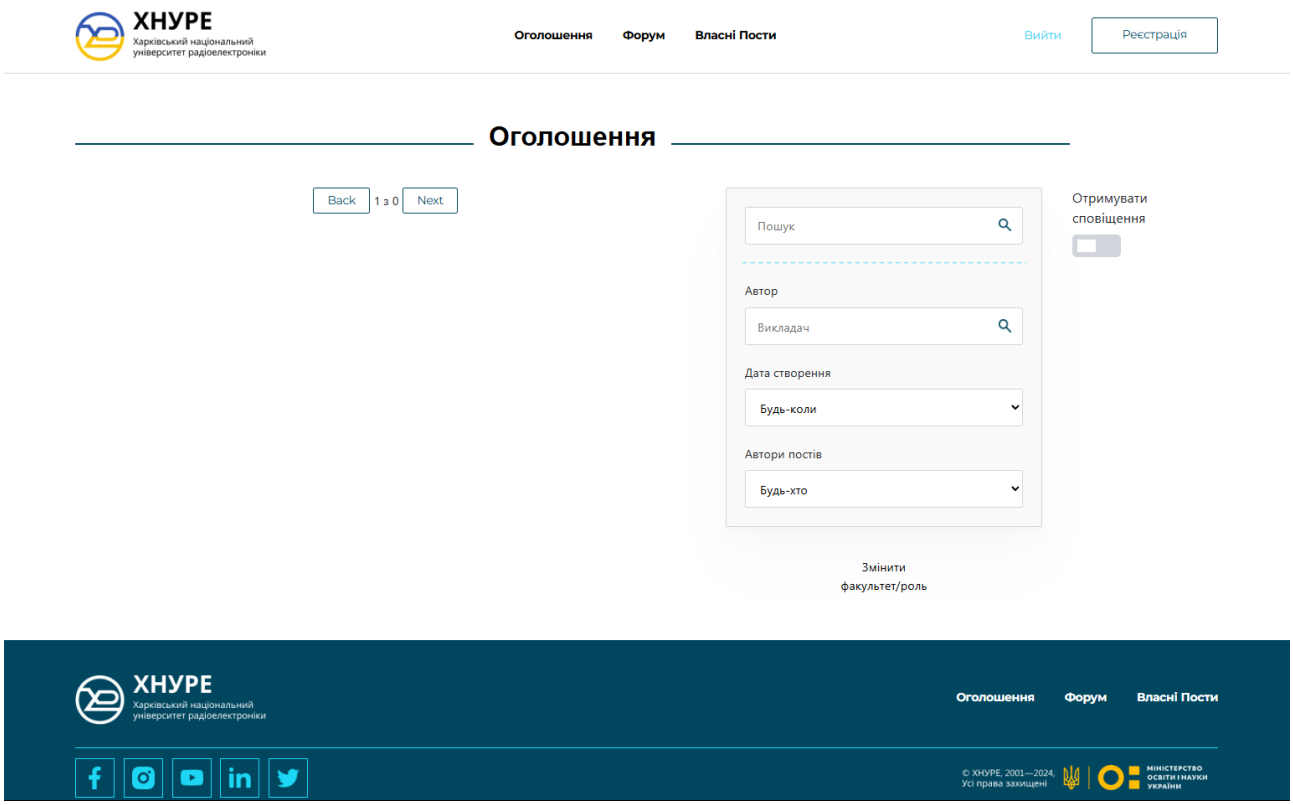

Рисунок 2.49 – Сторінка з списком оголошень

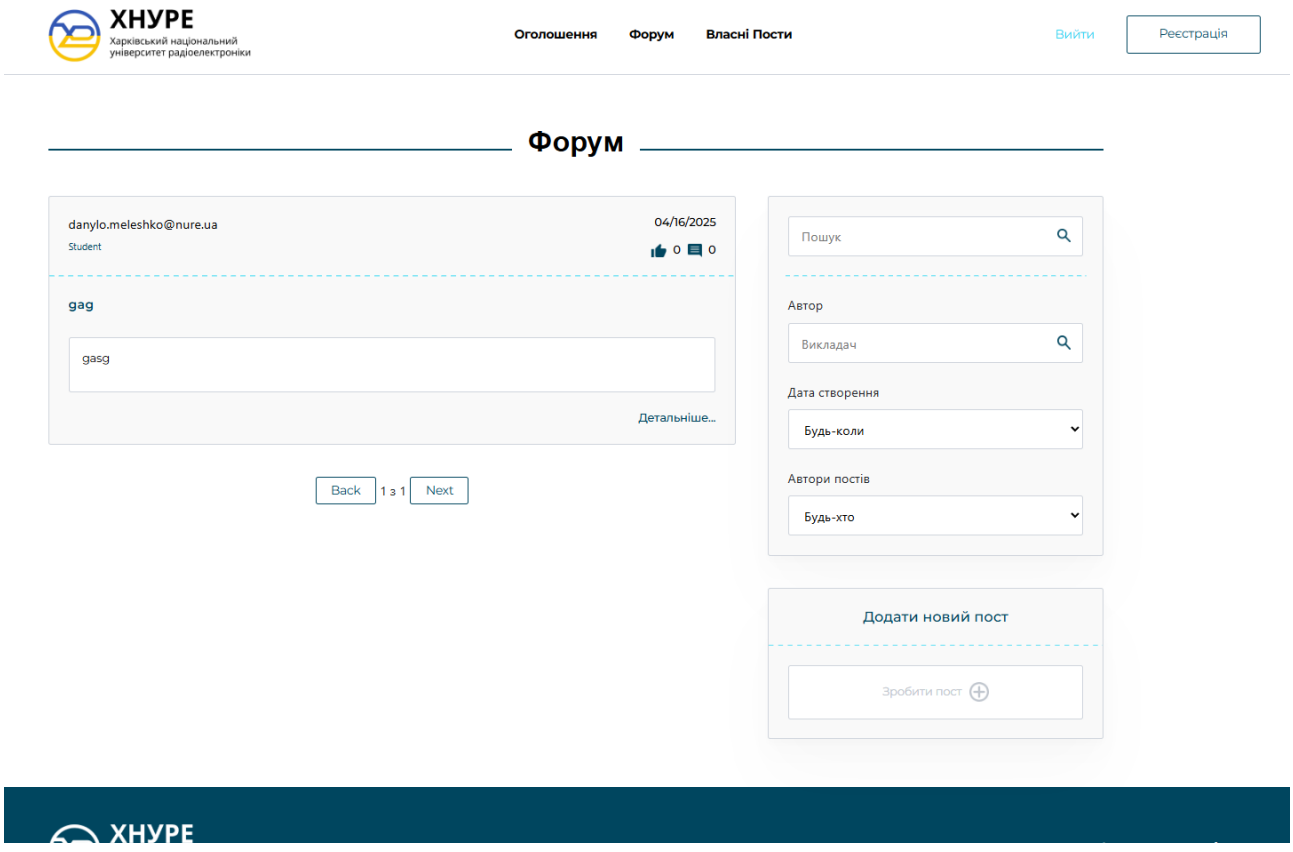

Рисунок 2.50 – Сторінка з списком дописів

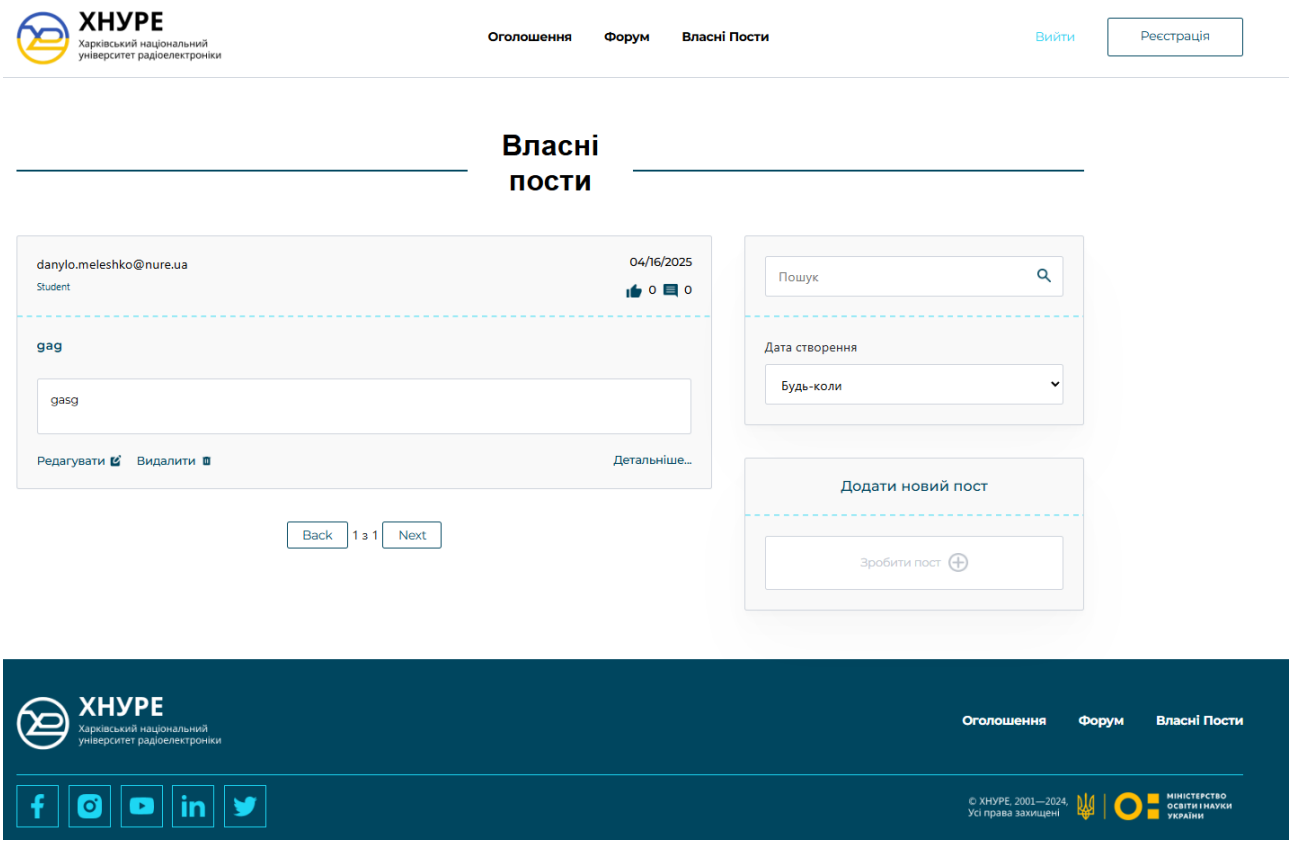

Рисунок 2.51 – Сторінка з списком власних дописів

Сторінки, які відповідають за створення нових оголошень/дописів, та редагування існуючих, простіші: вони мають лише сегмент для заголовка, та сегмент для форми (створення або редагування).

Форма для створення та редагування містить наступні пункти: заголовок, опис, чи можна додавати коментарі, обмеження доступності цього оголошення/допису за роллю/групою (кафедрою), а також кнопку для відправлення даних з форми до сервера.

Було побудовано необхідні компоненти та додано CSS стилі. В результаті отримані наступні сторінки: створити оголошення/допис та відредагувати оголошення/допис (рис.2.52 – 2.53).

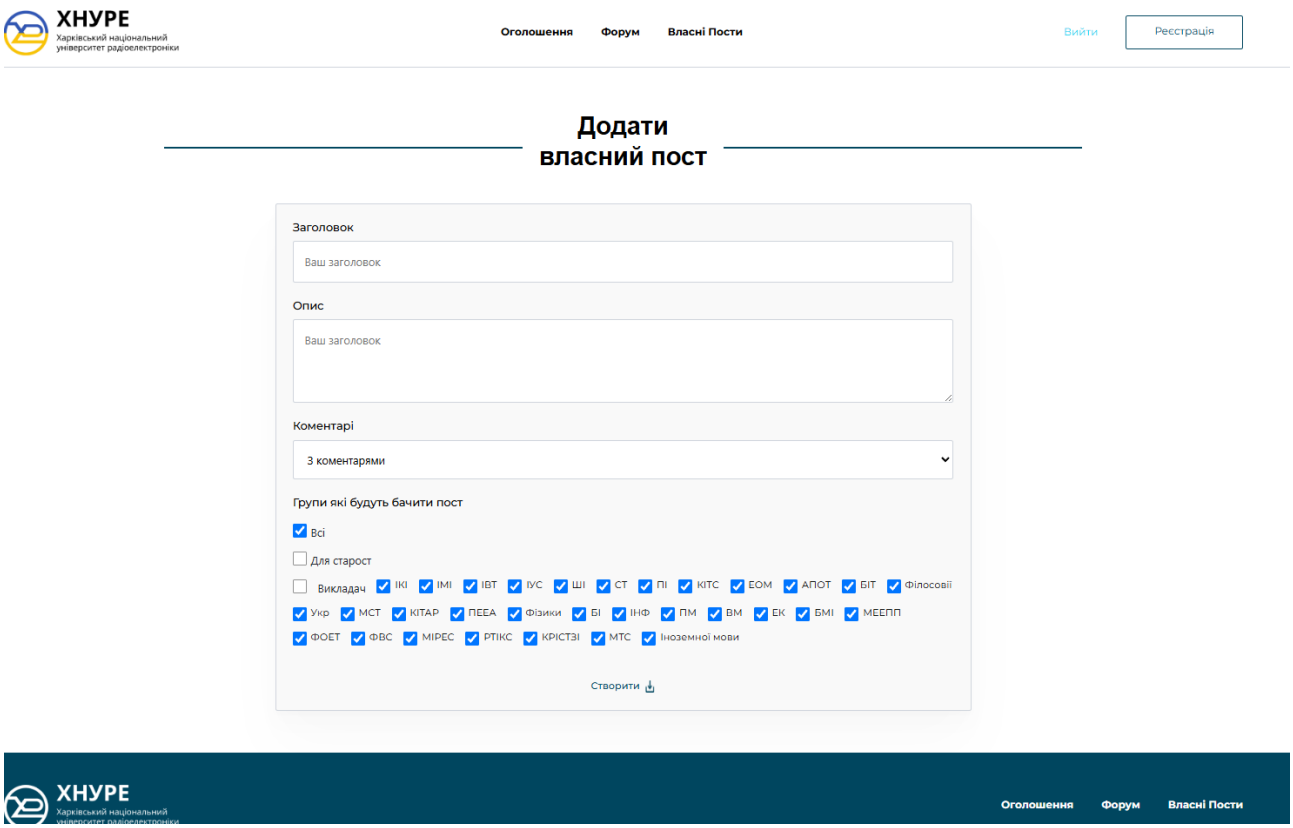

Рисунок 2.52 – Сторінка для створення допису

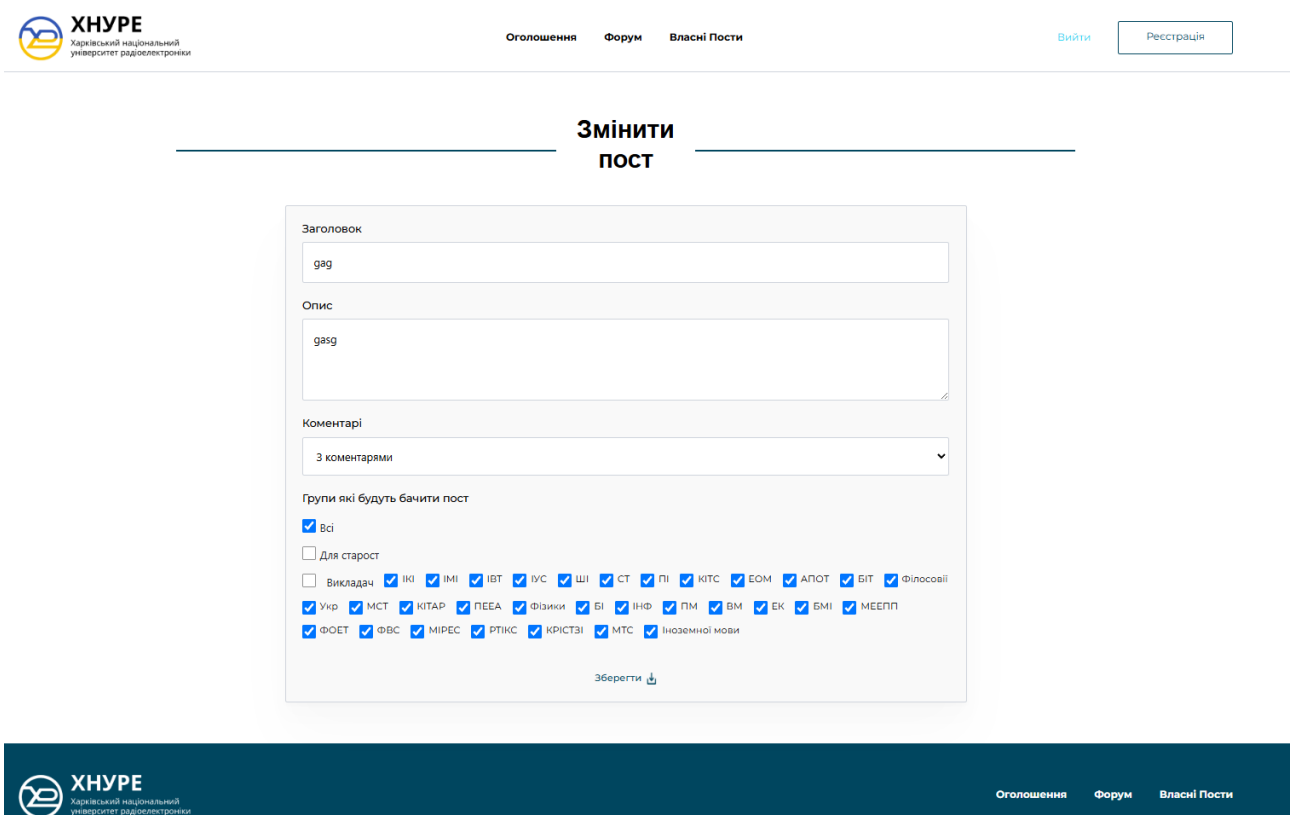

Рисунок 2.53 – Сторінка для редагування допису

Сторінка з детальною інформацією про оголошення/допис ділиться на три секції:

- Заголовок (оголошення/допису);
- Блок з повним змістом оголошення та кнопкою для залишення реакції;
- Коментарі

Блок з повним змістом відрізняється від блоку, який був використаний для сторінок з переліком оголошень/сповіщень. Блок з повним змістом, окрім того, що не обмежений в розмірах, також містить кнопку для залишення реакції (рис.2.54).

Компонент для коментарів містить перелік коментарів, залишених під оголошенням/дописом, можливі коментарі відповіді до інших коментарів, а також блок для додавання коментарю (рис.2.55). У середині компонента, після отримання даних з сервера, проходить зображення кожного коментарю. Використовується інший компонент, який зображає окремий коментар, а також можливі коментарі під ним.

Компонент, який відповідає за зображення коментаря, дозволяє видалити або відредагувати зміст коментарю, якщо автор коментарю, та обліковий запис, з якого зайшов користувач, співпадають (рис.2.56).

Після додавання CSS стилів, отримано сторінку для зображення деталей оголошення/допису (рис.2.57).

```
const handleLike : () => void  = () : void  => {
    setIsPostLiked(isPostLiked);
    toggleLike(post.id);

    setPost(prev : Post  => ({...prev, likes: isPostLiked ? post.likes - 1 : post.likes + 1}));
};

return (
    <div className="AnnounBody">
        <div className="AnnounHeader">
            <div className="AnnounHeaderAuthor">
                <div className="AnnounAuthorEmail">
                    <p>{post.authorEmail}</p>
                </div>
                <div className="AnnounAuthorRole">
                    <p id="AnnounAuthorRole">{post.authorRole}</p>
                </div>
            </div>
            <div className="AnnounHeaderCommLikes">
                <div>{dateFormat.format(new Date(post.createdAt))}</div>
                <div>
                    <div><img alt="" src="/images/LikeIcon.png"/>{post.likes}</div>
                    <div><img alt="" src="/images/CommentIcon.png"/>{post.amountOfComments}</div>
                </div>
            </div>
        </div>
        <svg xmlns="http://www.w3.org/2000/svg" width="1000" height="2" viewBox="0 0 1000 2" fill="none">
            <path d="M0 1H1000" stroke="#1DD6F3" strokeLinecap="round" strokeLinejoin="round"
                  strokeDasharray="6 6"/>
        </svg>
        <div className="AnnounTextBody">
            <div style={{overflow: 'visible', maxHeight: 'none'}} className="AnnounContent">
                <p style={{margin: '0'}}>{post.text}</p>
            </div>
        </div>
        <div className="AnnounFooter">
            <button className={`LikeButton ${isPostLiked ? 'liked' : ''}`} onClick={handleLike}>
                <img style={{scale: '1.3'}} src="/images/LikeIcon.png" alt=""
                     className={isPostLiked ? 'liked-image' : 'default-image'}/>
            </button>
        </div>
    </div>
)
```

Рисунок 2.54 – Компонент для блоку з повним змістом оголошення/допису

```
<div style={{width: '100%', paddingRight: '24px', paddingLeft: '24px', paddingTop: '24px'}} className="AnnouncementsAndPosts-UpperBlock">
    <div style={{paddingTop: '0'}}></div>
    <h1 style={{fontSize: '18px', fontWeight: '600'}} className="AnnouncementsAndPosts-H1">Коментарі</h1>
    <div style={{paddingTop: '0'}}></div>
</div>
{(role === "Admin" || role === "Teacher" || role === "Student") && (
    <div className="CommentInputWrapper">
    <textarea value={newComment}
              onKeyUp={(e : KeyboardEvent<HTMLTextAreaE....) : void => addNewComment(e)}
              onChange={e : ChangeEvent<HTMLTextAreaEle.... => commentInputOnFocus(e)}
              ref={commentInputRef} className="AddCommentInput" rows={1}
              placeholder='Enter text'
              onFocus={() : void => {
                  commentOnFocusButtonsRef.current.style.display = 'flex';
              }}></textarea>
        <div ref={commentOnFocusButtonsRef} className="OnFocusCommentInputButtons"
             style={{display: 'none'}}>
            <button onClick={() : void => {
                setNewComment("");
                commentOnFocusButtonsRef.current.style.display = 'none';
            }} className="CommentInputCancel">Відмінити
            </button>
            <button onClick={(e : MouseEvent<HTMLButtonElemen.... ) : void => {
                e.preventDefault();
                if (newComment.trim() === '') return;
                const comment: CommentAddDto = {
                    id: uuid(),
                    text: newComment,
                    authorId: user.id,
                    postId: postId,
                    replyTo: ""
                }
                addComment(comment, postId);
                post.amountOfComments = post.amountOfComments + 1;
                setPost(post);

                setNewComment("");
                commentInputRef.current.style.maxHeight = '59px'
            }} className="CommentInputSubmit">Додати
            </button>
        </div>
    </div>
)}

<div className="CommentsBody" style={comments.length < 1 ? {display: 'none'} : {}}>
    {comments.map((comment : Comment ) : false | Element => (
        comment.replyTo === null &&
        (
            <div style={{display: "flex", flexDirection: "column", gap: "12px"}} key={comment.id}>
                <CommentBox postId={postId} comment={comment}/>
            </div>
        )
    ))}
</div>
```

Рисунок 2.55 – Компонент для блоку з коментарями

```jsx
{user.id === comment.authorId && (
    <div className="OwnAnnounPostFooter">
        <div className="OwnPostAnnounButtons">
            <button className="OwnPostDeleteButton" onClick={handleToggleEdit}>
                {editPressed ? "Скасувати" : "Редагувати"}
                <img src="/images/IconEdit.png" alt="Edit"/>
            </button>
            <button className="OwnPostDeleteButton" onClick={handleDeleteComment}>
                Видалити
                <img src="/images/IconDelete.png" alt="Delete"/>
            </button>
        </div>
    </div>
)}
```

Рисунок 2.56 – Кнопки редагування та видалення власних коментарів

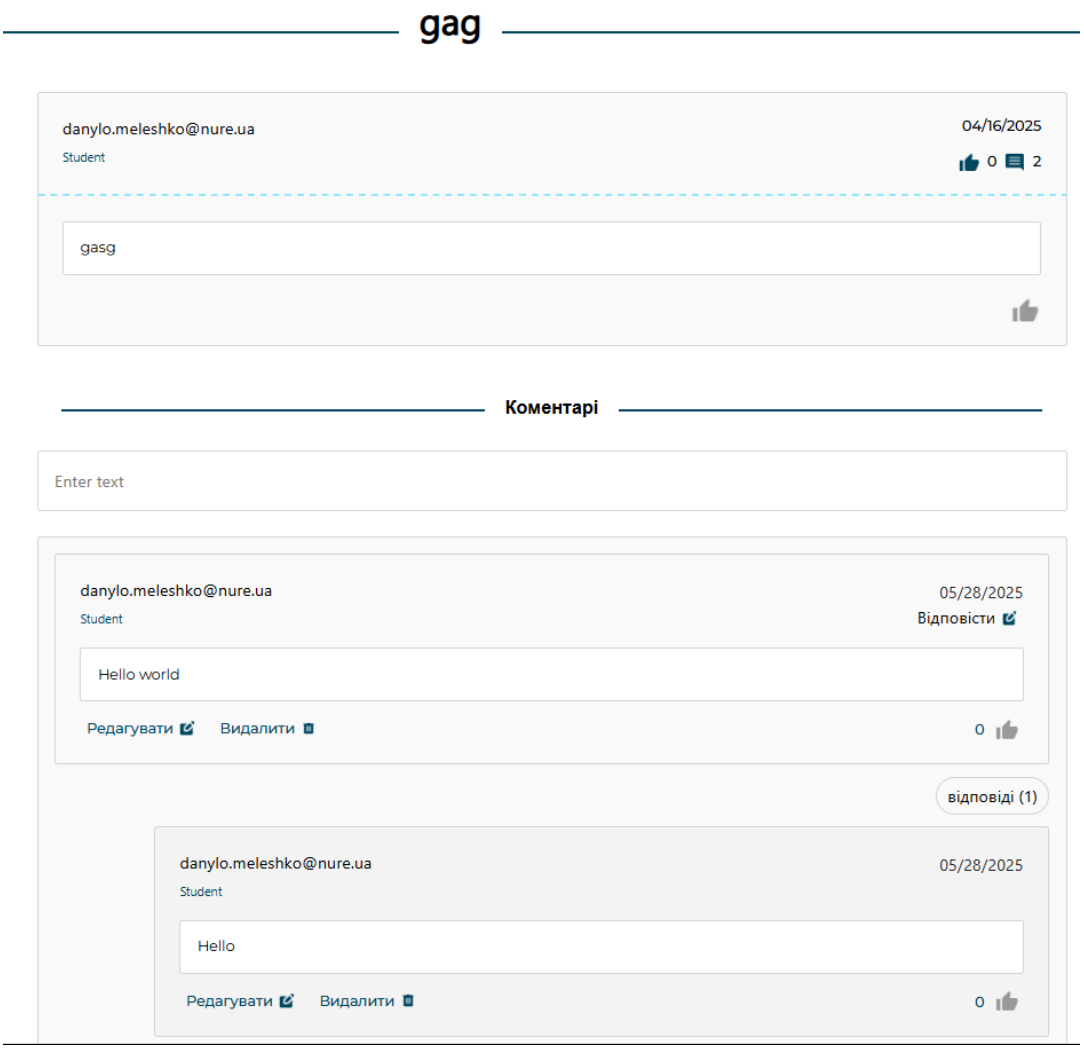

Рисунок 2.57 – Сторінка з деталями про оголошення/пост

Сторінки для реєстрації, входу в систему та зміни пароля, мають лише компонент з формою для заповнення облікових даних, та простий набір CSS стилів.

Форма для реєстрації містить наступні поля: поле для електронної пошти, поли для пароля, блок з варіантами кафедр для вибору, прапорці для отримання ролі викладача чи старости (можна отримати лише одну з ролей, або залишитись зі стандартною - Student), а також кнопку для відправлення облікових даних (рис.2.58).

Сторінка для входу в систему та зміни пароля, такі ж самі як і сторінка для реєстрації, але інша форма для заповнення.

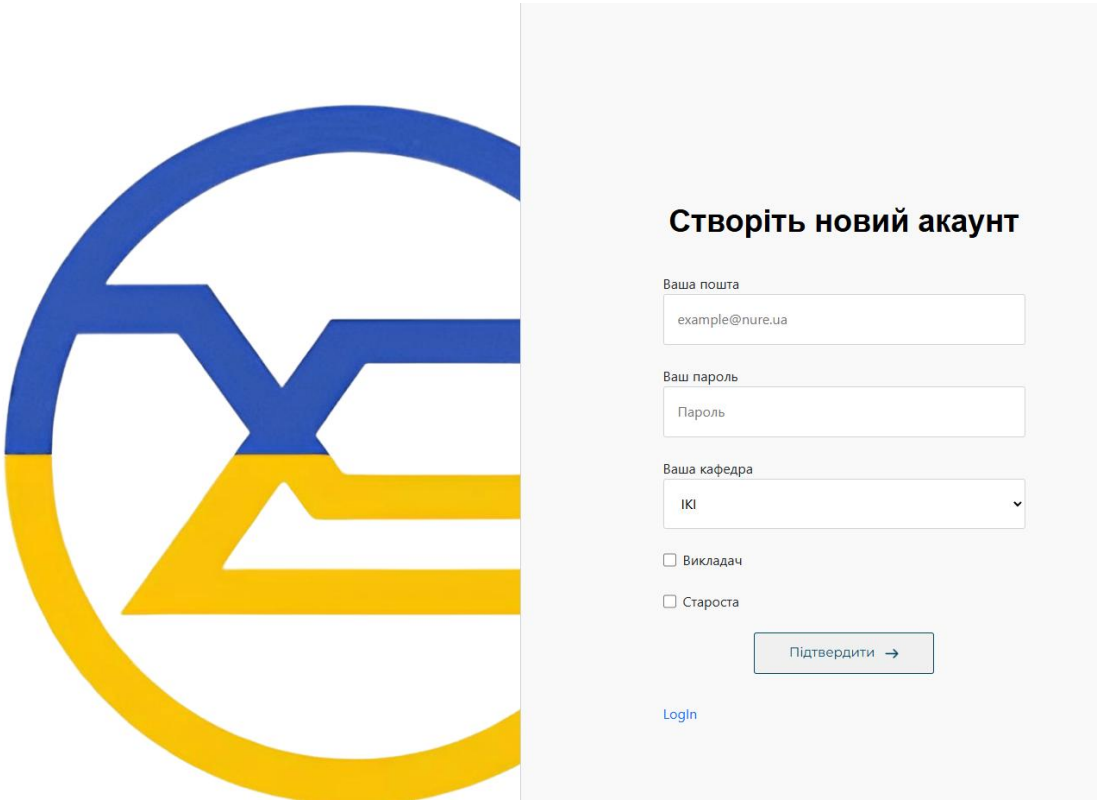

Рисунок 2.58 – Сторінка з реєстраційною формою

2.5 Розгортання веб-сайту

Перед завершенням розробки проєкту, необхідно провести тестування API endpoints (рис.2.59), а також перевірку того, чи всі відповіді з сервера приходять, та приходять в очікуваному стані, до браузера.

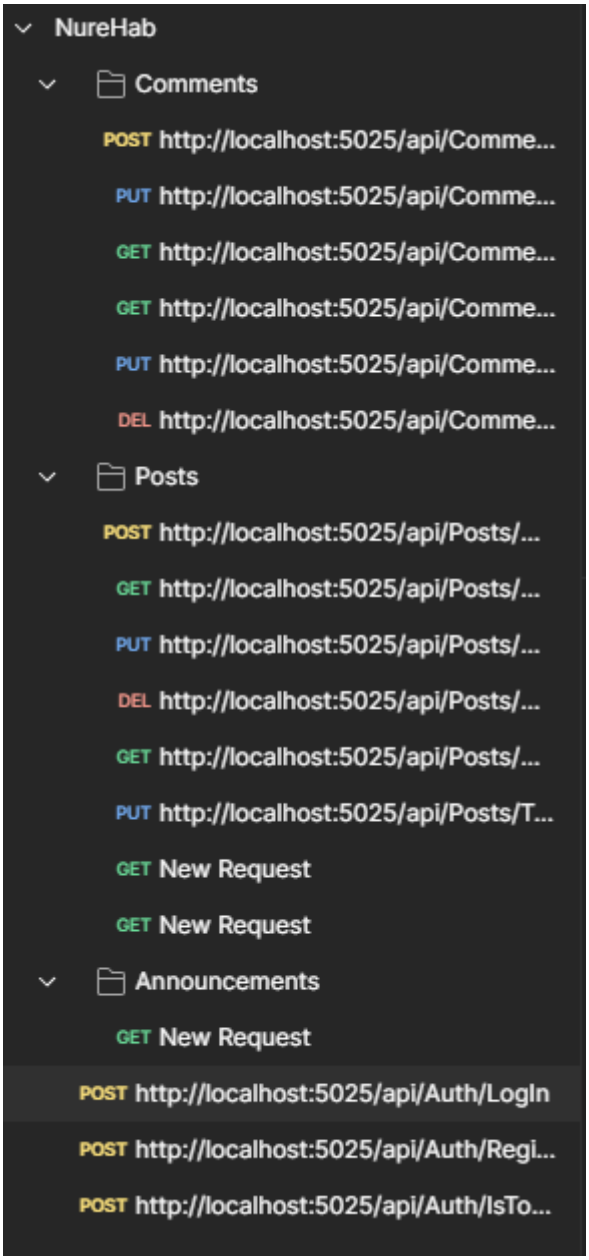

Рисунок 2.59 – Список тестових запитів, зроблених для тестування API endpoints, зроблених за допомогою Postman

Після завершення розробки проєкту, та проведення тестування, необхідно розмістити сайт в інтернет. Розміщення сайту в інтернет, ще часто називають розгортання (deployment) проєкту.

Для того, щоб розгорнути full-stack проєкт, необхідно створити та налаштувати віртуальну машину (ВМ), на якій буде розгортатись проєкт. Після підготування віртуальної машини (встановлення необхідної операційної системи, виділення оперативної пам'яті, налаштування доступу до ВМ), на ВМ необхідно встановити проєкт, а також все необхідне для його запуску (необхідні утіліти, мови програмування, база даних та необхідні пакети для запуску проєкту).

По завершенню підготовки ВМ, необхідно запустити проєкт на публічному порту, 80 (стандартний порт для http протоколу) або 443. 443 стандартний порт для https протоколу, що означає, що для використання цього порту, необхідно також підготувати SSL/TSL сертифікат.

Окрім цього, можна піти далі, та використати веб сервер, наприклад Apache або Kestrel, в якості проксі сервера, який буде слухати порт 443, а API буде слухати порт 8080 (використовується коли 80 або 443 зайняті). Таким чином, запити спочатку будуть йти на порт 443, а Apache буде перенаправляти запити на порт 8080, який слухає наш API.

Для того, щоб розмістити клієнтську частину, можна створити теку в проєкті з API, та розмістити в ній спакований React проєкт. Для того, щоб спакувати React проєкт, необхідно скористатись командою: npm run build. Ця команда створить нову теку Build з готовим до запуску проєктом, без всього зайвого. Після отримання спакованої версії проєкту, необхідно перейти до API проєкту, та перенести його в створену під нього теку (рис.2.60).

Для того, щоб Asp.Net Core проєкт міг працювати зі статичними файлами, та зробити так, щоб коли браузер звертається до API, використовуючи неіснуючий endpoint, він повертав вказаний HTML файл (рис.2.61). React проєкт представляє собою набір статичних файлів.

Останнім кроком буде отримання доменного імені. Придбати доменне ім’я можна на одному з багатьох сайтів. Після отримання домену, його необхідно прив’язати до IP адреси сервера.

Full-stack проєкт – це про проєкт, який має і клієнтську (frontend), і серверну частину проєкту (backend).

Віртуальна машина - модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації. Віртуальна машина, на відміну від програми емуляції конкретного пристрою, забезпечує повну емуляцію фізичної машини чи середовища виконання [21].

Secure Sockets Layer (SSL) – це протокол зв’язку або набір правил, який створює безпечне з’єднання між двома пристроями або програмами через мережу. SSL – це технологія, яку ваші програми або браузери могли використовувати для створення безпечного та зашифрованого каналу зв’язку через будь-яку мережу. Transport Layer Security (TLS) – це оновлена версія SSL, яка виправляє існуючі вразливості SSL. TLS виконує автентифікацію ефективніше та продовжує підтримувати зашифровані канали зв’язку [22].

Порт — це програмно-визначений номер, пов'язаний з мережевим протоколом, який приймає або передає дані для певної служби [23].

Проксі сервер – це сервер, який виступає як посередник між браузером (клієнтом), та сервером (API). Всі запити не направляються прямо до API, а спочатку мають пройти через проксі, який потів перенаправляє запити до сервера.

Головна ціль використання проксі – це покращення безпеки серверу. Проксі можна використовувати для обмеження кількісті запитів за проміжок часу, що дозволяє захищати сервер від ддос атак.

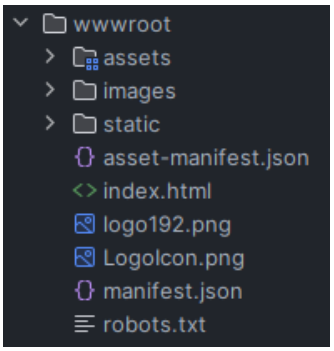

Рисунок 2.60 – Готовий до запуску React проєкт, який було перенесено до теки в API

```
app.UseStaticFiles();
app.MapFallbackToFile("index.html");
```

Рисунок 2.61 – Код необхідний до використання React проєкту з API

Для того, щоб спростити процес розгортання проєкту, та не виконувати кожен крок самостійно, було використано Azure.

Azure – це хмарна платформа, розроблена Microsoft, яка надає послуги для розробки, розгортання та керування проєктами. Azure може надавати всі три види послуг: SaaS (Software as a service), PaaS (Platform as a service), IaaS (Infrastructure as a service) [24].

SaaS – це модель хмарних обчислень, де постачальник пропонує клієнту користування прикладним програмним забезпеченням та керує всіма необхідними фізичними та програмними ресурсами [25].

PaaS – це модель хмарних обчислювальних послуг, де користувачі надають, створюють екземпляри, запускають та керують модульним пакетом обчислювальної платформи та застосунків без складності побудови та підтримки інфраструктури, пов'язаної з розробкою та запуском застосунків [26].

IaaS – це модель хмарних обчислень, де постачальник хмарних послуг надає обчислювальні ресурси, такі як сховище, мережа, сервери та віртуалізація [27].

Хмарні технології дають можливість зберігати файли не на девайсі користувача, а на віддаленому, добре захищеному комп'ютері. Для компанії це

важливо, оскільки можна не купувати дорогі сервери, платити за регулярні оновлення, технічну підтримку. Крім того, за наявності пароля і логіна доступ можливий з будь-якої точки світу, де є інтернет [28].

Незважаючи на те, що використання хмарних ресурсів не є безкоштовним, для нових користувачів, Azure дозволяє придбати безкоштовний сервер на 12 місяців, а також базу даних на місяць.

Першим кроком було створено ресурс групу, в якій будуть зберігатись всі необхідні сервіси та ресурси проєкту (рис.2.62).

Після створення ресурс групи, було обрано підходящій для розгортання сервіс, та заповнена форма для отримання послуги (рис.2.63). Обираючи регіон, необхідно розуміти, що в залежності від відстані до серверу, буде різна швидкість передачі запитів. Завдяки тому, що Azure підтримує мову програмування C#, було обрано опцію публікації використовуючи код проєкту.

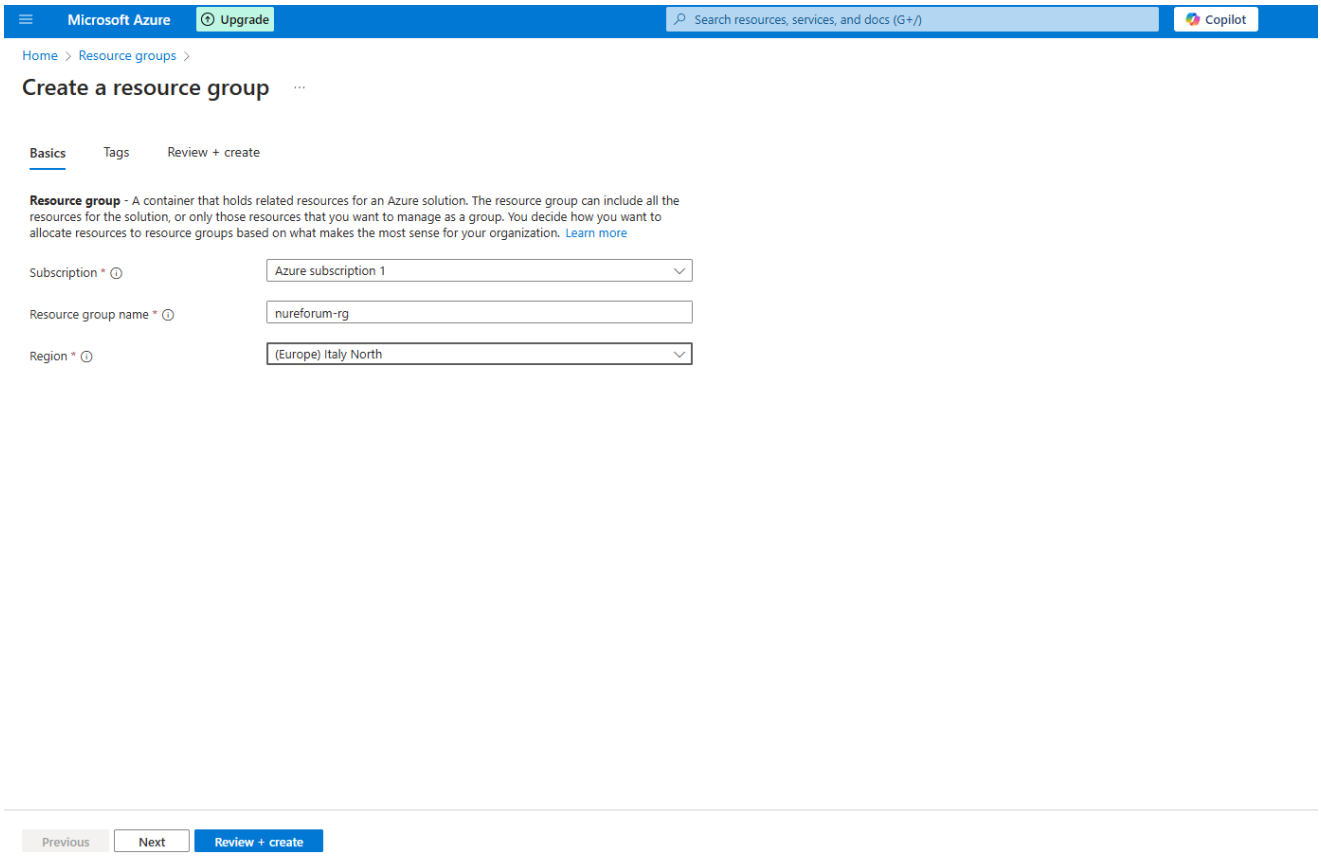

Рисунок 2.62 – Форма для створення ресурс групи

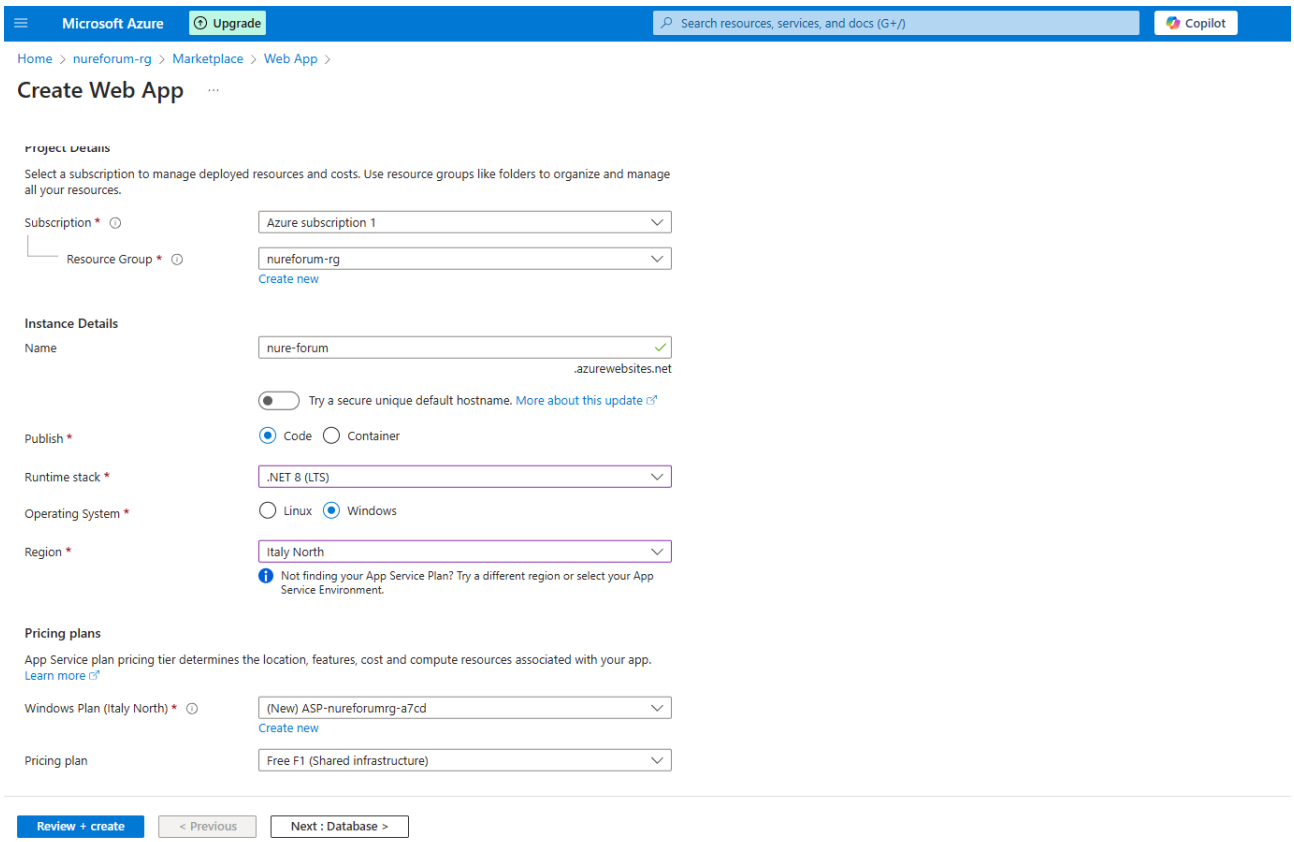

Рисунок 2.63 – Форма для створення Web App сервісу

Після створення Web App сервісу, було створено Sql Database Server (рис.2.64). Важливо обрати регіон, який був обраний для Web App, щоб комунікація між БД та сервером була настільки швидка, наскільки можливо. Орім цього, важливо налаштувати автентифікацію бази даних.

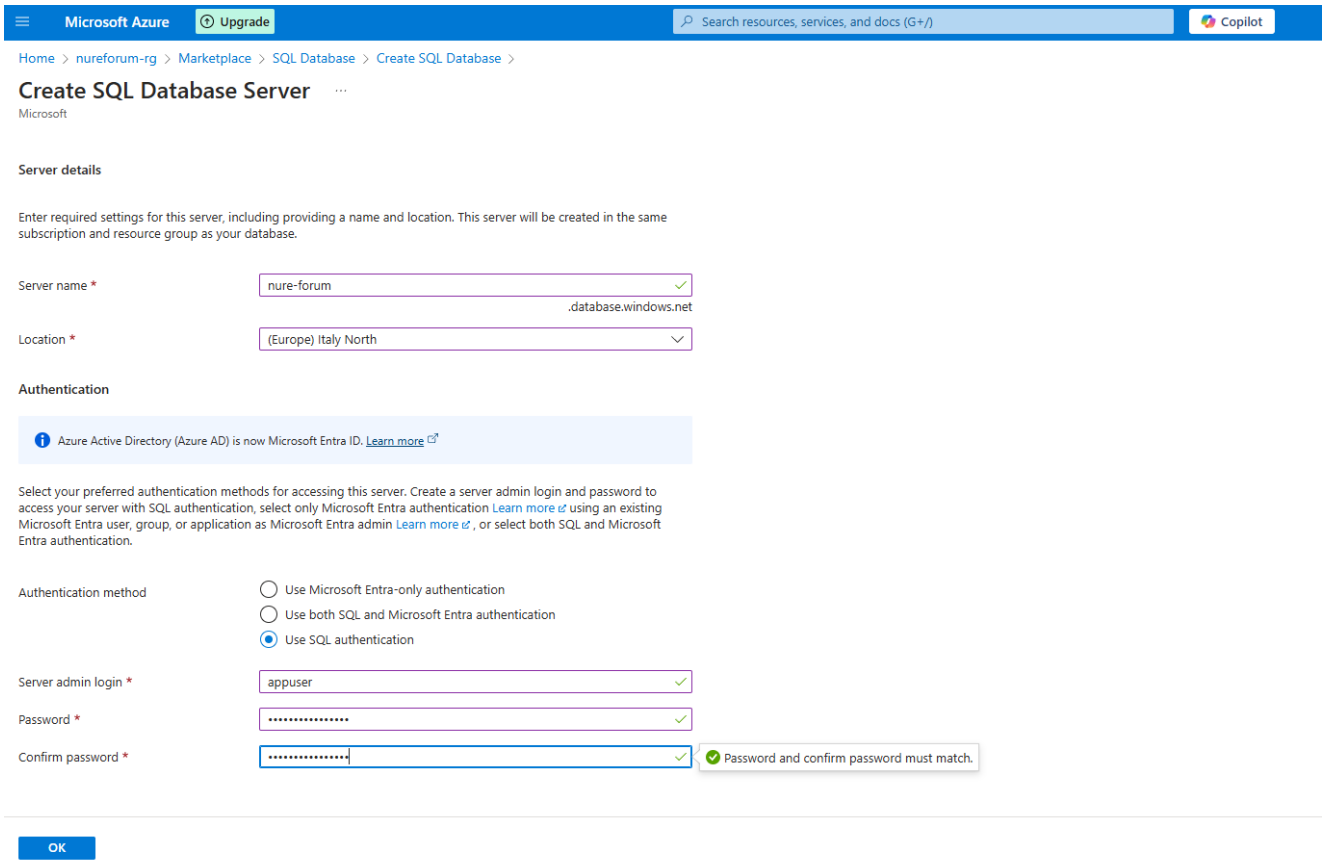

Рисунок 2.64 – Форма для створення серверу з базою даних

Після Створення серверу з БД, необхідно перейти до Web App, та в налаштуваннях знайти Environment variables. Перейшовши туди, було створено нову змінну для connection string (рис.2.65). Знайти connection string можна в налаштування серверу БД.

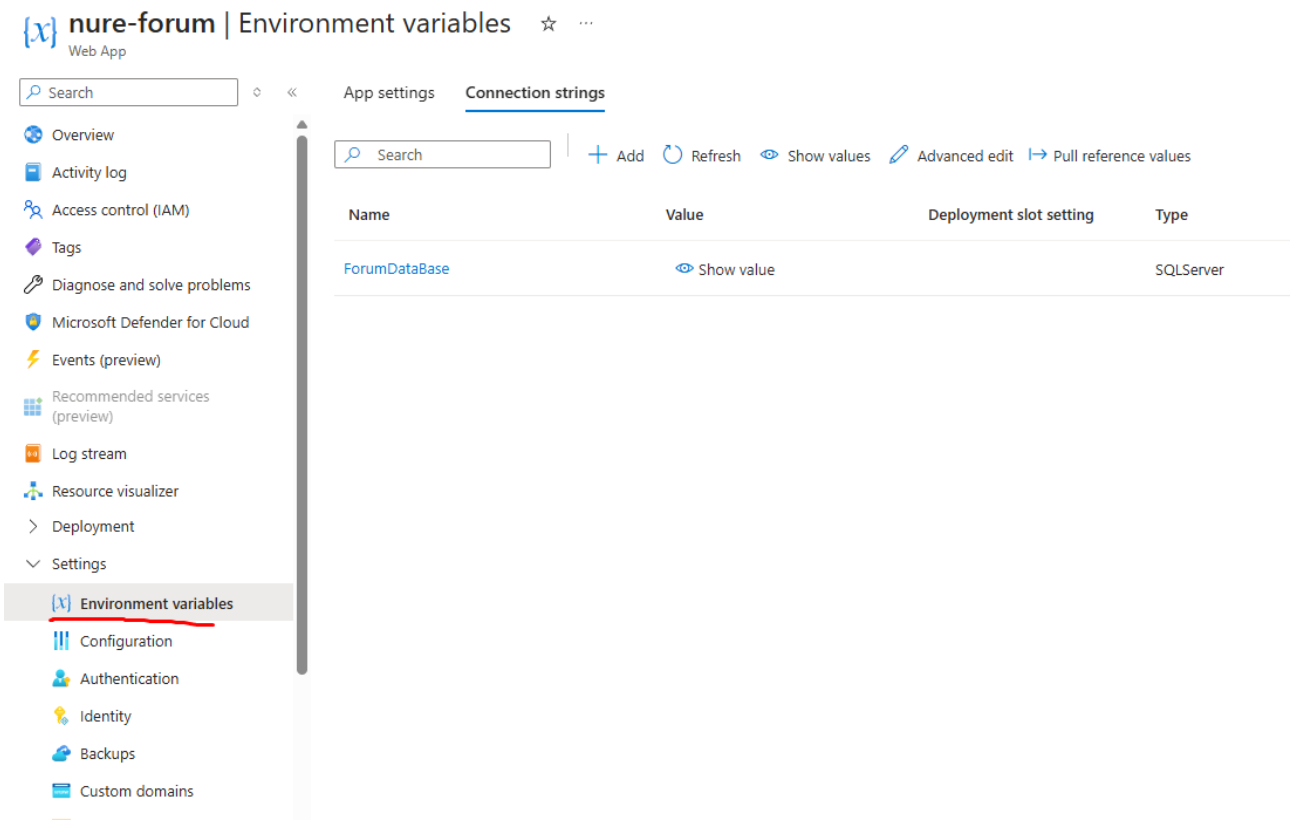

Рисунок 2.65 – Створення змінної простору для connection string до бази даних

Змінні простору, дозволяють маніпулювати важливими для публікації та запуску змінними, без необхідності робити зміни в коді вручну.

Після того, як всі сервіси налаштовані та готові до записку, необхідно відкрити редактор коду (в якому написано проєкт), встановити зв’язок з акаунтом Azure та розгорнути готовий до публікації проєкт.

Для того щоб отримати готову до публікації версії .Net проєкту, необхідно скористатись наступними командами: dotnet build та dotnet publish -c Release -o /bin/Publish.

Скориставшись допомогою редактора коду, готовий проєкт було розгорнути на сервері (рис.2.66).

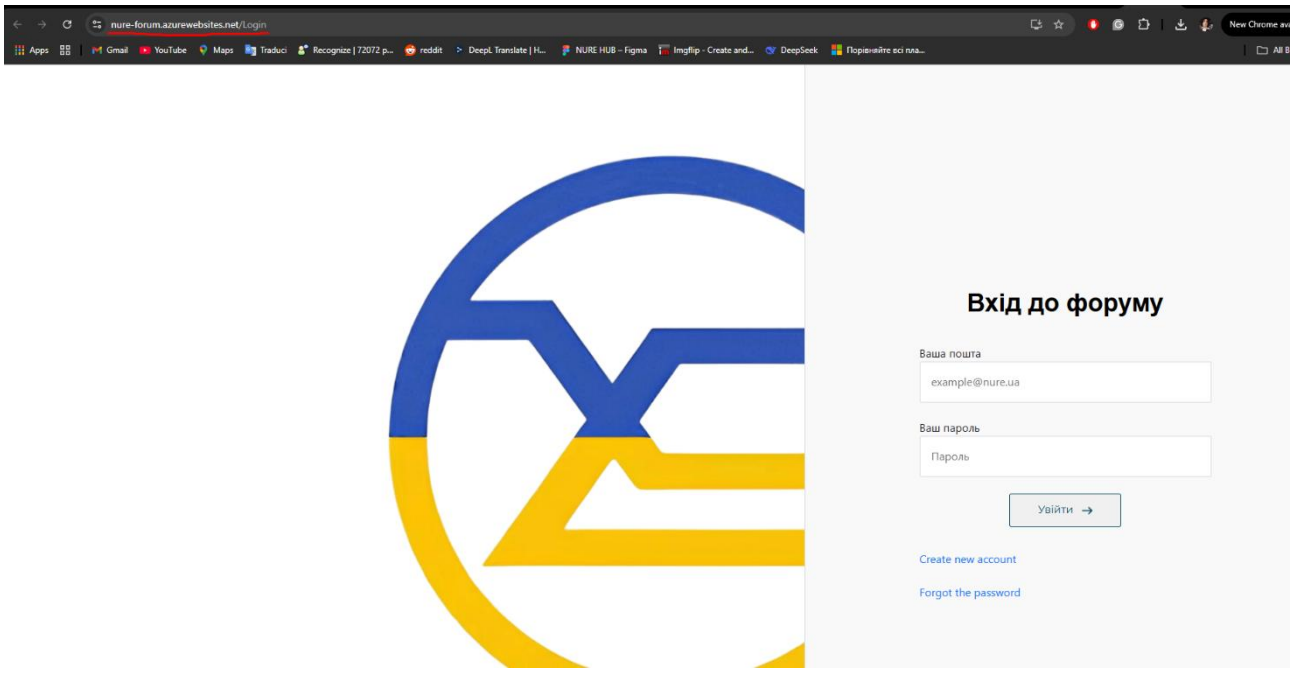

Рисунок 2.66 – Результат розгортання проєкту

В якості доменного імені сайту, було використано створений на Azure домен. Однак при бажанні, можна придбати та використати інший домен, створений на сторонньому ресурсі. Приклад іншого домену: nure-forum.com або nure-forum.ua.

Сервер, що було використано для розгортання проєкту, стандартно, також має налаштований для захисту від ддос атак проксі сервер.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ ПОКРАЩЕННЯ

3.1 Можливі покращення серверної сторони

Якщо кількість активних користувачів стане завеликою для серверу, що призведе до зменшення швидкості отримання даних з API, непоганим рішенням для цього може бути додавання кешування.

Кешування допомагає прибрати необхідність робити запити які часто повторюються. Зберігаючи результати запиту з присвоєним ключем. У випадку якщо такий самий запит було зроблено декілька хвилин тому або менше, сервер одразу повертає результат.

При існуванні запитів які часто повторюються, додавання кешу може сильно зменшити навантаження на сервер та збільшити швидкість отримання даних.

Для кращого масштабування та збільшення гнучкості проєкту, гарним рішенням може буте перехід на мікросервісну архітектуру.

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, при якому проєкт поділяється на більш дрібні сервіси, де кожен сервіс відповідає за виконання однієї функції. Ця архітектура дуже гарно підходить проєктам які постійно ростуть та розвиваються.

Переваги мікросервісної архітектуру:

- Масштабованість. Роздільність проєкта на різні незалежні частини, робить цю архітектуру зручною для горизонтального масштабування (додати більше інстансів для навантажених сервісів).
- Гнучкість при внесенні змін. Данна архітектура не тільки гарно підходить для підтримки проєкту командою розробників, а також дозволяє оновлювати окремі частини проєкту без впливу на всю систему.
- Стійкість при помилках. Збій одного сервісу, на валить всю систему.

3.2 Можливі покращення клієнтської сторони

С точки зору виконання поставленого завдання, все необхідно виконано та працює як і очікувалось. Однак замість того, щоб робити запити до API просто використовуючи Axios, кращім рішенням буде використання пакету React Query.

Основна перевага React Query – це його вбудоване автоматичне кешування даних. Вбудоване кешування даних може значно підвищити рівень продуктивності веб-сайту.

Окрім кешування, React Query також має вбудований refetching (автоматичне оновлення даних). Завдяки цьому, можна позбутись необхідності в використанні useEffect() для оновлення даних.

useEffect() – це хук, який дозволяє робити rerender сторінки та допомагає синхронізувати контент сторінки з поточним станом даних. Часто працює в парі з useState(). Першим параметром передається функція з діями які мають бути виконанні при зміні значення другого параметру. Другий параметр – це масив з значеннями, зміна яких активує функцію, яка була вказана в першому параметрі (рис.3.1).

```
useEffect(() : void => {
    if(role === "Teacher" || role === "Admin"){
        announAddRef.current.style.display = "flex";
    }
    getAnnouncements();
}, [page]);
```

Рисунок 3.1 – Приклад використання useEffect(). При зміні сторінки, відбувається перевірка ролі користувача, а також робиться запит до API на отримання нових оголошень

Для зменшення повторення коду, що робить запити до API та опрацьовує дані отримані з серверу, було активно використано useContext() хук. useContext() дозволяє використовувати методи та змінні з інших компонентів. Використавши

useContext() та передавши контекст іншого компонента, можна отримати його функціонал:

const {getAnnouncements, announcements, setPage, page, totalPages} = useContext(AnnouncementsContext);

Однак кращім рішенням було б використання кастомних хуків. Перевага кастомних хуків над useContext(), в цьому випадку - це те, що для отримання методів та змінних, не потрібно, щоб компонент був огорнутий в компонент-провайдер іншого компонента (рис.3.2).

Щоб передати функціонал компоненту, необхідно використати метод createContext() для створення контексту. Після створення контексту, необхідно повернути компонент-провайдер та вказати в атрибуті value функціонал, який ми хочемо передавати іншим компонентам (рис.3.3).

```
<AnnouncementsProvider>
    <CommentsProvider>
        <AnnouncementPageBody announcementId={announcementId || ""}/>
    </CommentsProvider>
</AnnouncementsProvider>
```

Рисунок 3.2 – Приклад передачі контексту. AnnouncementPageBody компонент може використовувати функціонал AnnouncementsProvider компоненту

```
<AnnouncementsContext.Provider value={ {addAnnouncement, editAnnouncement, deleteAnnouncement, allAnnouncementsOfTheUser,
                        announcement, announcements, toggleLike, getAnnouncement, getAnnouncements, isLiked,
                         setAnnouncement, setAnnouncements, announcementsOfTheUser, page, setPage, totalPages, setTotalPages,
                         setAnnouncementsOfTheUser, isAnnouncementLiked, setIsAnnouncementLiked,
                         isNotificationsActive, setIsNotificationsActive, toggleNotifications, isNotificationsEnabled,} } >
    {children}
</AnnouncementsContext.Provider>
```

Рисунок 3.3 – Використання компонент-провайдера для передачі функціоналу

На відміну від useContext(), кастомні хуки, можна використовувати в будь-якому компоненті, без обмежень, які має useContext(). Ця перевага кастомних хуків, робить їх більш підходящими для комунікації з API. Кастомні хуки, зробили б код простішим та зрозумілішим.

ВИСНОВКИ

В даній кваліфікаційній роботі було описано процес розробки та розгортання веб-сайту, а саме веб-форуму для університету ХНУРЕ. Окрім опису процесу розробки та розгортання сайту в Інтернет, було також проведено аналіз можливого вдосконалення проєкту.

Визначено та опрацьовано основані етапи для виконання даної задачі:

- Описані технології, що були застосовані для розробки проєкту;
- Прописано хід написання коду серверної та клієнтської сторони
- Описано процес створення та підключення бази даних до веб-серверу;
- Розглянуто різні варіанти впровадження авторизації та аутентифікації, а також описано та проаналізовано процес впровадження стандарту JWT;
- Продемонстровані всі endpoints створеного API;
- Описано налаштування навігації між сторінками;
- Описано процес створення сторінок веб-сайту;
- Описано процес розгортання веб-сайту з використанням Azure сервісів;
- Проведено аналіз можливого вдосконалення проєкту.

Даний проєкт створено для того, щоб замінити сторонні ресурси, що використовуються для комунікації між викладачами, студентами та іншими співробітниками університету, та має перевагу в наданні масової інформації.

При використанні даного проєкту, університет отримає власну централізовану базу даних з оголошеннями. Наявність такого ресурсу, може спростити пошук необхідних оголошень, як для студентів, так і для викладачів.

Як видно даний проєкт здатний приносити велику користь, а також має високий потенціал для розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Programming language [Електронний ресурс] // wikipedia. – 2024. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Programming_language.

2. What is Just-In-Time(JIT) Compiler in .NET [Електронний ресурс] // geeksforgeeks. – 2021. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/.

3. A tour of the C# language [Електронний ресурс] // learn.microsoft. – 2025. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview.

4. What is a Framework? Why We Use Software Frameworks [Електронний ресурс] // Brian O'Grady. – 2015. – Режим доступу до ресурсу: https://codeinstitute.net/global/blog/what-is-a-framework/#:~:text=A%20framework%2C%20or%20software%20framework,selectively%20modified%20by%20adding%20code.

5. What React Is and Why It Matters by Eric Baer [Електронний ресурс] // oreilly. – 2018. – Режим доступу до ресурсу: https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html.

6. Built-in React Hooks [Електронний ресурс] // react. – 2023. – Режим доступу до ресурсу: https://react.dev/reference/react/hooks.

7. Introduction to .NET [Електронний ресурс] // learn.microsoft. – 2024. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/introduction.

8. What is an API (Application Programming Interface) [Електронний ресурс] // geeksforgeeks. – 2025. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-an-api/.

9. Яку роль відіграє база даних у розробці веб-сайтів. [Електронний ресурс] // jakub. – 2021. – Режим доступу до ресурсу: https://desigo.eu/uk/яку-роль-відіграє-база-даних-у-розробц/.

10. Як бізнесу працювати з реляційними та нереляційними базами даних? [Електронний ресурс] // Ганна Сухорукова. – 2023. – Режим доступу до ресурсу: https://hub.kyivstar.ua/articles/yak-biznesu-praczyuvaty-z-relyaczijnymy-ta-nerelyaczijnymy-bazamy-danyh.

11.Cos'è SQL (Structured Query Language)? [Електронний ресурс] // aws.amazon. – 2024. – Режим доступу до ресурсу: https://aws.amazon.com/it/what-is/sql/.

12. Types of Databases. [Електронний ресурс] // mongodb. – 2024. – Режим доступу до ресурсу: https://www.mongodb.com/resources/basics/databases/types.

13. NoSQL database types explained: Column-oriented databases. [Електронний ресурс] // Donald Farmer. – 2024. – Режим доступу до ресурсу: https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Column-oriented-databases.

14. HTML e CSS – La guida completa per principianti [Електронний ресурс] // Andrea Di Rocco. – 2025. – Режим доступу до ресурсу: https://sos-wp.it/tutorial/guida-html-css/.

15. What is Postman? [Електронний ресурс] // postman. – 2021. – Режим доступу до ресурсу: https://www.postman.com/product/what-is-postman/.

16. Cross-Origin Resource Sharing (CORS) [Електронний ресурс] // postman. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS.

17. Автентифікація [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Автентифікація.

18. Session-Based Authentication: A Detailed Guide [2024] [Електронний ресурс] // Darko Bozhinovski. – 2024. – Режим доступу до ресурсу: https://supertokens.com/blog/session-based-authentication.

19.What is an API endpoint? [Електронний ресурс] // postman. – 2023. – Режим доступу до ресурсу: https://blog.postman.com/what-is-an-api-endpoint/.

20. Rendering (computer graphics) [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Rendering_(computer_graphics).

21. Віртуальна машина [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Віртуальна_машина.

22. Qual è la differenza tra SSL e TLS? [Електронний ресурс] // aws.amazon. – 2024. – Режим доступу до ресурсу: https://aws.amazon.com/it/compare/the-difference-between-ssl-and-tls/.

23. port [Електронний ресурс] // Gavin Wright. – 2021. – Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/port.

24. Microsoft Azure [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Microsoft_Azure.

25. Software as a service [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_as_a_service.

26. Platform as a service [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Platform_as_a_service.

27. Infrastructure as a service [Електронний ресурс] // wikipedia. – 2025. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Infrastructure_as_a_service.

28. Що таке хмарні технології: визначення та застосування [Електронний ресурс] // Михайло Колмиков. – 2024. – Режим доступу до ресурсу: https://megasite.ua/ua/chto-takoe-oblachnie-tehnologii-opredelenie-i-primenenie.