Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

РОЗРОБКА МІКРОСЕРВІСНОГО WEB-ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ. ЧАСТИНА 1. МІКРОСЕРВІСИ АВТОРИЗАЦІЇ, АДМІНІСТРУВАННЯ НАПОВНЕННЯ КУРСІВ ТА ЧАТУ В РЕАЛЬНОМУ ЧАСІ

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Федишен Б. В.
(прізвище та ініціали)
Керівник: асистент каф. КН
Денисов І. К.
(прізвище та ініціали)
"07" 06 2024 р.
Рецензент: к.т.н., доцент каф. САІТ
Крижановський Є. М.
(прізвище та ініціали)
"07" 06 2024 р.

Допущено до захисту
Зав. кафедри: проф., д.т.н Лесний А.А.
"07" 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"07" 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Федишену Богдану Вікторовичу

1. Тема роботи: Розробка мікросервісного WEB-додатку освітньої платформи. Частина 1. Мікросервіси авторизації, адміністрування наповнення курсів та чату в реальному часі.
Керівник роботи: асистент Денисов Ігор Костянтинович.
Затверджені наказом вищого навчального закладу "11" 03 2024 року № 80

2. Термін подання студентом роботи 27.05.2024 р.

3. Вихідні дані до роботи :
Мова програмування C#;
Використання об'єктно-орієнтованого програмування;
Середовище розробки Visual Studio 2022.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки додатку освітньої платформи; проектування; проектування додатку освітньої платформи; розробка модулів авторизації, адміністрування наповнення курсів та чату в реальному чаті; тестування розроблених програмних модулів додатку освітньої платформи; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):

схема алгоритму роботи протоколу WebSocket; загальна струк функціональних модулів; UML-діаграми класів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийня
1-4	Асистент кафедри КН Денисов І. К.		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Прим
		початок	закінчення	
1	Обгрунтування доцільності розробки додатку освітньої платформи	06.05.24	10.05.24	
2	Проектування додатку освітньої платформи	11.05.24	15.05.24	
3	Розробка модулів авторизації, адміністрування наповнення курсів та чату в реальному чаті	16.05.24	24.05.24	
4	Тестування розроблених програмних модулів додатку освітньої платформи	27.05.24	29.05.24	
5	Розробка інструкції користувача	30.05.24	03.06.24	
6	Аналіз отриманих результатів та формування висновків	03.06.24	04.06.24	
7	Оформлення бакалаврської дипломної роботи	04.06.24	06.06.24	

Студент (підпис) **Федишен Б** (прізвище та ініціа

Керівник роботи (підпис) **Денисов І.** (прізвище та ініці

АНОТАЦІЯ

Федишен Б. В. Розробка мікросервісного Web-додатку освітньої платформи. Частина 1. Мікросервіси авторизації, адміністрування наповнення курсів та чату в реальному часі. Бакалаврська дипломна робота складається з 70 сторінок формату А4, на яких є 32 рисунків, список використаних джерел містить 29 найменувань.

Мета даної бакалаврської дипломної роботи полягає в покращенні комунікації між учасника платформи. В ході виконання роботи створено гнучку та масштабовану систему, яка сприятиме зручній та ефективній взаємодії користувачів із навчальним матеріалом.

Результатом дипломного дослідження є модулі бізнес-логіки у вигляді мікросервісів, що реалізовані на мові програмування C# у програмному середовищі Visual Studio 2022.

Ключові слова: мікросервіси, C#, ASP .NET, комунікація в реальному часі, шаблони проектування, бізнес-логіка.

ABSTRACT

B. V. Fedyshen Development of a microservice Web-application of an educational platform. Part 1. Authorization microservices, course content administration and chat in real time. Bachelor work consists of 70 pages of A4 format, on which there are 32 drawings, list of used sources contains 29 denominations

The purpose of this bachelor thesis is to improve communication between platform participants. In the course of the work, a flexible and scalable system was created, which will facilitate convenient and effective user interaction with the educational material.

The result of the diploma research is business logic modules in the form of microservices, which are implemented in the C# programming language in the Visual Studio 2022 programming environment.

Keywords: microservices, C#, ASP .NET, real-time communication, design patterns, business logic.

ЗМІСТ

ВСТУП

Актуальність досліджень. У наш час освіта є ключовою складовою для багатьох людей. Вони завжди бажають знань і зазвичай отримують їх у навчальних закладах, таких як університети, коледжі та школи, де процес навчання включає лекції та практичні заняття.

Але через останні події відвідування навчальних закладів стає проблематичним, тому освітні платформи для дистанційного навчання стають дуже популярними. Вони дозволяють студентам отримувати знання незалежно від місця проживання, де єдиною вимогою є доступ до Інтернету, що є нескладною задачею у наш час.

Освітні платформи дозволяють викладачам ділитися своїми знаннями в електронному форматі, створювати завдання для студентів та оцінювати їх виконання.

Важливою частиною навчального процесу є комунікація між викладачем і студентами, оскільки завдяки їй студенти можуть краще засвоювати матеріал та якісніше виконувати завдання.

Для забезпечення цих потреб потрібно використовувати доступний інструментарій та оптимальні підходи для створення відповідного програмного забезпечення. Поєднуючи класичні та сучасні підходи для проектування та реалізації програмного продукту можна забезпечити його комфортне використання користувачами.

Отже, розробка освітньої платформи з компонуванням різних підходів до проектування та програмування є актуальною.

Метою дослідження є покращення комунікації між учасниками освітньої платформи.

Об’єкт дослідження – процес комунікації між учасниками освітньої платформи.

Предмет дослідження – мікросервісний веб-застосунок освітньої платформи дистанційного навчання.

Задачі дослідження:

1. дослідити методи та технології для створення бізнес-логіки освітньої платформи;

2. аргументувати важливість розробки освітньої платформи;

3. розробити архітектуру освітньої платформи та структуру бізнес-модулів;

4. проаналізувати та пояснити вибір інструментів для розробки;

5. реалізувати модулі авторизації, адміністрування наповнення курсів та чату в реальному часі;

6. провести тестування розробленого програмного коду.

Апробація результатів роботи

Результати досліджень було апробовано на LIII науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024) м. Вінниці у 2024 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ

1.1 Аналіз предметної області

Освітні платформи є важливим інструментом у сучасній освіті, оскільки вони відкривають доступ до знань та навчальних ресурсів для широкого кола користувачів незалежно від їхнього місця проживання чи графіку. Створення таких платформ вимагає не лише технічних знань, але й уваги до особливостей навчального процесу та потреб користувачів.

Сучасні учасники освітнього процесу шукають не тільки зручний спосіб доступу до навчального матеріалу, але й інтерактивність та можливості активної взаємодії з платформою. Це означає, що програмні продукти в цій сфері повинні бути гнучкими, забезпечувати можливість інтеграції різних навчальних матеріалів та інтерактивних інструментів, а також надавати засоби для взаємодії та співпраці між учасниками освітнього процесу.

Створення інноваційних освітніх платформ потребує виваженого підходу до архітектури програмного продукту, врахування складності взаємодії з різними типами даних та забезпечення високої захищеності й конфіденційності інформації.

Комунікація між учасниками освітньої платформи є ключовим аспектом, який впливає на ефективність та результативність навчального процесу. Успішне забезпечення цієї комунікації вимагає ретельного планування та розробки зручних інструментів для спілкування.

Навчальна платформа повинна надавати можливості для взаємодії між студентами та викладачами, а також для обміну ідеями та досвідом між учасниками навчального процесу. Це може включати в себе чати, форуми та інші засоби спілкування, які сприяють активному обміну інформацією та співпраці.

Одним з найбільш доцільних методів комунікації можна вважати групові чати курсів в реальному часі, що дозволяють учасникам освітньої платформи швидко взаємодіяти.

Ефективна комунікація на освітній платформі сприяє покращенню розуміння матеріалу, стимулює активну участь студентів у навчальному процесі та сприяє взаєморозумінню між учасниками освітнього середовища.

При розробці групових чатів для курсів на освітній платформі важливо враховувати кілька ключових факторів:

– масштабованість системи, що означає здатність обробляти великий обсяг одночасних підключень і повідомлень без втрати продуктивності та швидкості реакції;

– безпека та конфіденційність даних, зокрема, шляхом використання шифрування та захисних протоколів передачі інформації;

– можливість створення різних груп чатів для різних курсів або тем, а також налаштування прав доступу до чатів для різних категорій користувачів;

– інтуїтивний і зручний інтерфейс для користувачів, що дозволяє легко взаємодіяти з чатами, обмінюватися файлами та документами, а також отримувати сповіщення про нові повідомлення та події в чатах.

Переваги освітніх платформ:

– глобальний доступ. Освітні платформи дозволяють людям з усього світу мати доступ до інтерактивного онлайн-навчання;

– інтерактивність. Багато освітніх платформ пропонують інтерактивні завдання, що сприяє активному навчанню та підвищує зацікавленість учнів. Це може включати відеоуроки, віртуальні лабораторії, мультимедійний матеріал та інші інтерактивні методи навчання;

– широкий вибір ресурсів. Студенти мають доступ до різноманітних навчальних матеріалів, від підручників та відеоуроків до тестів та курсів з підвищення кваліфікації.

Недоліки освітніх платформ:

– відсутність безпосереднього контакту. Відсутній безпосередній оффлайн контакт між викладачем та студентами, може призвести до менш ефективного сприйняття матеріалу та відчуття відокремленості;

– технічні проблеми. Протягом користування можуть виникати різні проблеми, наприклад, проблеми з підключенням до інтернету, низька швидкість завантаження сторінок або проблеми з програмним забезпеченням;

– необхідність самодисципліни. Без відповідної мотивації та уміння самостійно організовувати свій час, учням та студентам може бути важко досягти успіху в навчанні онлайн.

1.2 Огляд аналогів

JetIQ (рис. 1.1) – освітня платформа ВНТУ, інструмент взаємодії студентів з викладачами та навчальним закладом [2].

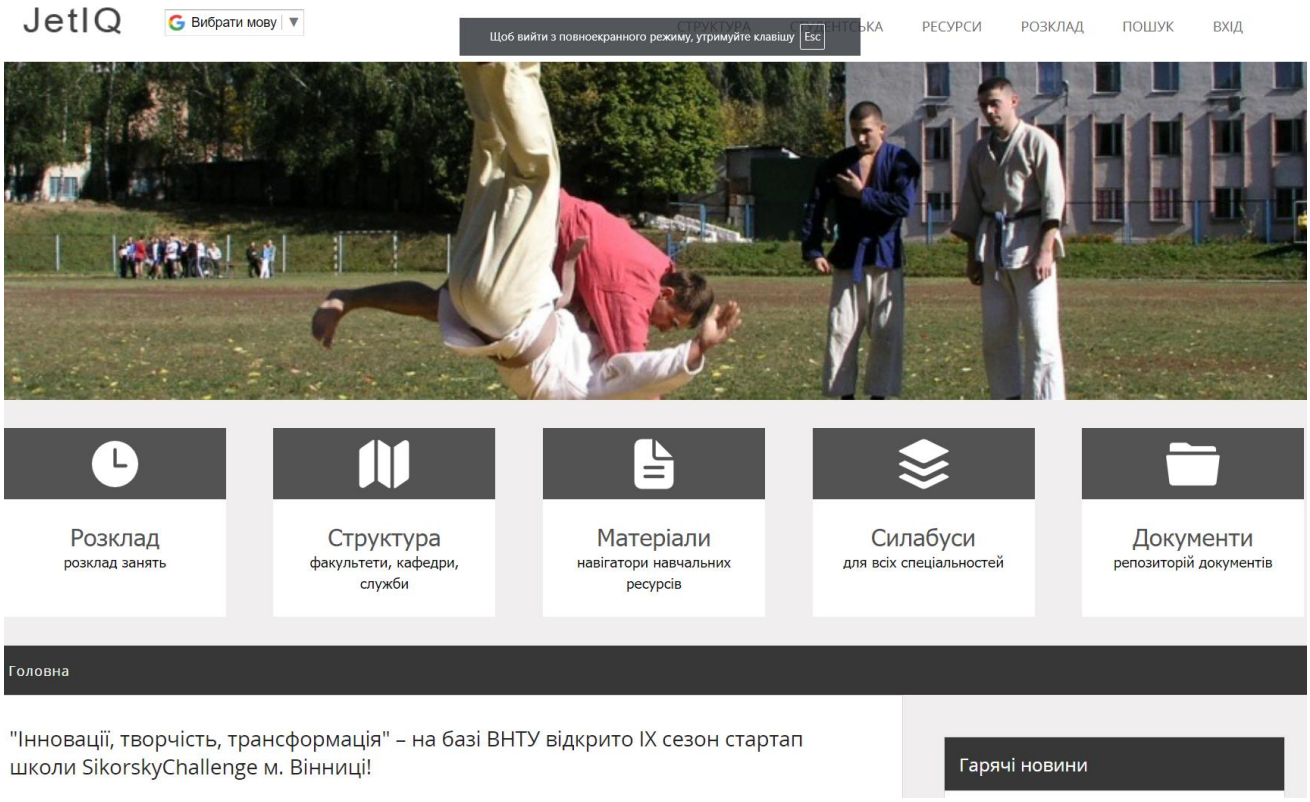

Рисунок 1.1 – Головна сторінка платформи JetIQ

Основні функції JetIQ включають:

1. Створення профілю. Після вступу до університету, кожний студент отримує власний особистий кабінет, який можуть заповнити на свій розсуд.

2. Адміністрування матеріалів. Викладач може адмініструвати наповнення навігатора предмету, який він викладає.

3. Доступ до матеріалів. З особистого кабінету студента можна перейти як до матеріалів своєї спеціальності, так і до матеріалів інших спеціальностей, що дозволяє знайти корисну інформацію.

4. Надсилання виконаних робіт. Завдяки інструменту “Файл-експрес” можна надіслати звіти викладачам на перевірку, та отримати зворотній зв’язок. Також відслідковуються статуси завдань.

5. Новинна стрічка. Завдяки новинній стрічці можна дізнатися про якісь події, або отримати потрібну для навчання інформацію.

6. Доступ до журналу. У студентів є можливість переглянути власну успішність.

7. Ведення журналу. Викладачі мають можливість ведення електронного журналу, який інтегровано в платформу.

8. Доступ до розкладу. Через особистий кабінет студента можна перейти до розкладу певної навчальної групи.

9. Інструмент для проведення тестів. Система має вбудований модуль тестування рівня знань студентів.

10. Наявність форуму. Платформа має вбудований форум, який дозволяє написати викладачеві, студентам або цілим групам.

11. Можливість прикріпити посилання на відеоконференції. В кожному навігаторі є посилання на конференції Google Meet.

Недоліки:

1. Застарілий користувацький інтерфейс. Користувацький інтерфейс платформи JetIQ виглядає застарілим та не інтуїтивним станом на 2024.

2. Відсутність можливості комунікації в реальному часі. Платформа не має групових чатів в реальному часі, що дозволяє максимально швидко комунікувати.

3. Поганий технічний стан. Платформа часто не працює, або працює не належним чином, довгі завантаження сторінок, помилки 500.

4. Погана робота з різними типами файлів. Система не коректно працює з багатьма типами файлів.

Classroom (рис.1.2) – це платформа для навчання, розроблена Google, яка дозволяє вчителям створювати віртуальні класи, де вони можуть надавати завдання, спілкуватися зі студентами та керувати навчальним процесом [3].
Основні функції Classroom включають:

1. Створення класів. Вчителі можуть створювати окремі класи для кожного предмету або групи учнів.

2. Надання завдань. Вчителі можуть додавати завдання, розміщувати матеріали для читання або перегляду, завдання для самостійної роботи тощо.

3. Збір і перевірка завдань. Студенти можуть відправляти завдання через Classroom, і вчителі можуть їх перевіряти й оцінювати.

4. Розклад занять. Classroom дозволяє встановлювати розклад занять, щоб студенти завжди могли знати, коли відбуваються навчальні заходи.

5. Робота з різними типами даних. Вчителі можуть додавати різноманітні матеріали, такі як відео, презентації, файли для завантаження тощо. Завдяки інфраструктурі Google, більшість файлів підтримуються платформою.

6. Можливість прив’язати посилання на відеоконференції. В кожному класі є можливість прикріпити одне посилання на конференції Google Meet, або додати як матеріал декілька.

Недоліки:

1. Відсутність можливості комунікації в реальному часі. Платформа не реалізує чати в реальному часі, а форуми не дають такої гнучкості в комунікації.

2. Дещо перевантажений інтерфейс користувача. Для ефективного використання платформи, необхідно доволі тривалий час на ознайомлення з нею.

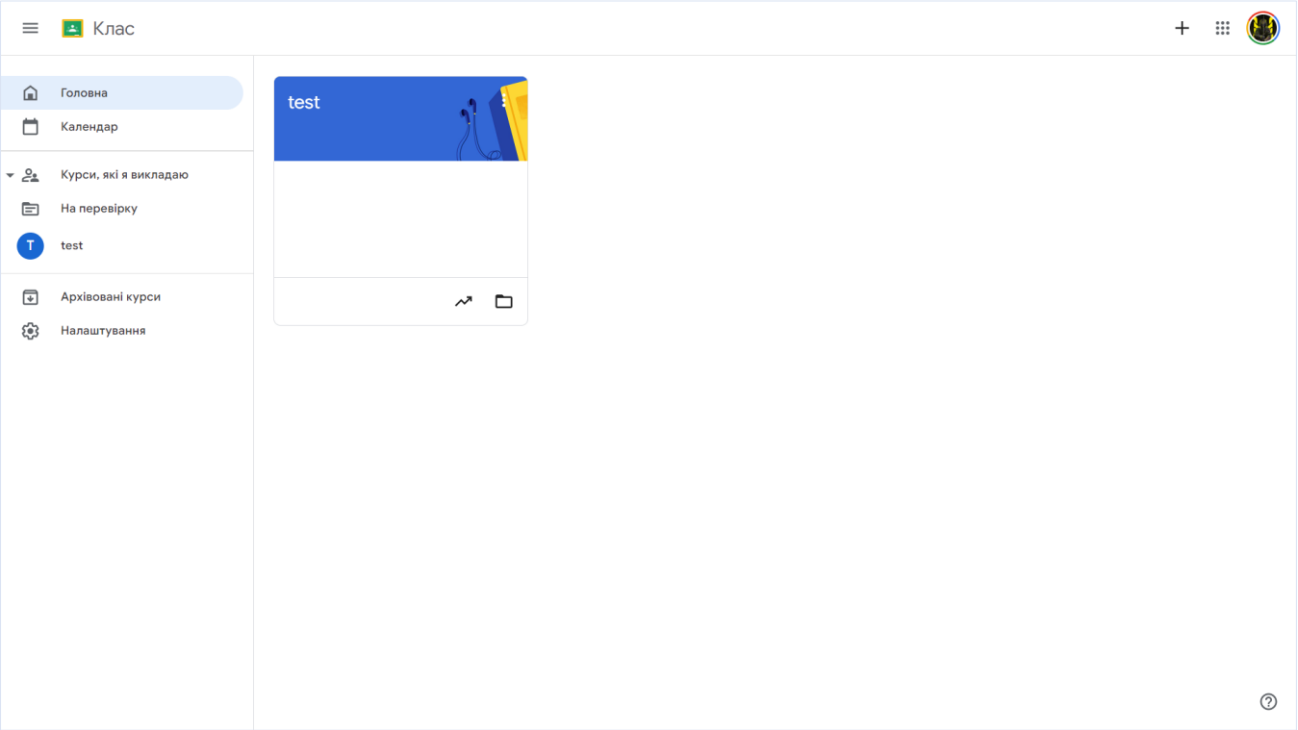

Рисунок 1.2 – Головна сторінка платформи Classroom

1.3 Постановка задачі та формулювання вимог

Освітні платформи є важливим інструментом для навчання та спілкування, особливо в умовах розвитку інформаційних технологій та зростання кількості користувачів. Вони мають на меті створення зручного та ефективного середовища для навчання та спілкування.

У цій роботі було поставлено завдання розробити інтерактивну освітню платформи, спрямовану на забезпечення доступу до якісної освіти для різних категорій користувачів і забезпечення можливості навчання на різних рівнях із використанням сучасних технологій.

Для досягнення цього завдання платформа повинна мати наступні функції:

– Можливість реєстрації та авторизацію користувачів.

– Налаштування профілів користувачів з можливістю редагування особистої інформації.

– Доступ до каталогу курсів з різних предметів або областей, можливість завантаження навчальних матеріалів і взаємодії з ними.

– Інструменти для користувачів, що дозволяють створювати, редагувати та видаляти курси.

– Інструменти для користувачів, що дозволяють адмініструвати наповнення курсів.

– Групові чати в реальному часі для обговорення матеріалів курсів та спілкування між користувачами.

– Інструменти оцінювання з можливістю виставлення оцінок та коментування.

– Зручний, зрозумілий та естетичний дизайн інтерфейсу, що забезпечує доступ до всіх зазначених інструментів, а також надає інформацію користувачеві в разі виникнення помилок чи збоїв на серверній частині.

1.4 Висновок до розділу 1

У даному розділі проведено порівняльний аналіз деяких існуючих освітніх платформ з увагою до їх можливостей та недоліків. Після ознайомлення з різними платформами, визначено, що розробка такої системи доволі важке завдання для виконання якого потрібно враховувати досвід існуючих платформ.

Освітня платформа з функціональним чатом має низку переваг і може бути дуже доцільною в сучасному освітньому середовищі. Зручна комунікація за допомогою чату дозволяє вчителям та учням спілкуватися миттєво, обговорювати певні питання, уточнювати інформацію та надавати додаткові пояснення. Учні можуть використовувати чат для спільної роботи над завданнями й обміну ідеями, що сприяє активному навчанню.

Отже використання чату в реальному часі дозволяє покращити взаємодію користувачів платформи, що дозволить підвищити ефективність навчання.

Виконано постановку задачі та опис призначення розроблюваної освітньої платформи. Після аналізу аналогів визначено напрямки подальших досліджень.

2 ПРОЕКТУВАННЯ ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ

2.1 Вибір архітектури застосунку, структури його бізнес-модулів та способу комунікації

Мікросервісна архітектура [4] є найбільш доцільним вибором для реалізації даного проекту, оскільки вона демонструє переваги у масштабуванні застосунку “в ширину”. Це означає, що з ростом обсягу та складності системи можливо ефективно збільшувати її потужність, додаючи нові модулі та інтегруючи нові функціональні можливості. Крім того, такий тип архітектури сприяє легшій підтримці написаного коду та його тестуванню, оскільки окремі мікросервіси можуть підтримуватись незалежно один від одного. Це забезпечує гнучкість в розробці та впровадженні змін, а також дозволяє швидше виявляти та виправляти помилки.

Тепер розглянемо загальну структуру для кожного бекенд-модуля. При розробці програмного забезпечення варто зважати на те, що написаний код може підтримуватися, допрацьовуватися та тестуватися іншими розробниками. Щоб врахувати цей аспект часто використовуються певні архітектурні шаблони. Одним з таких є шаблон “Clean Architecture” [5]. Він дозволяє організовувати код таким чином, щоб інкапсулювати бізнес-логіку та полегшити орієнтування в проекті.

Отож структура кожного проекту буде такою:

– Domain – у проекті анемічна доменна модель, тобто на цьому рівні будуть тільки бізнес моделі.

– Infrastructure – на цьому рівні будуть класи, що відповідають за роботу з даними.

– Core – на цьому рівні будуть розташовуватися класи, що реалізують бізнес-логіку модуля.

– Web – на цьому рівні будуть розташовуватися контролери, в яких будуть реалізовані ендпоінти, які є однією з конвенцій REST [6].

Для кращого розуміння наведемо рисунок 2.1, для графічного відображення рівнів структури згаданого шаблону.

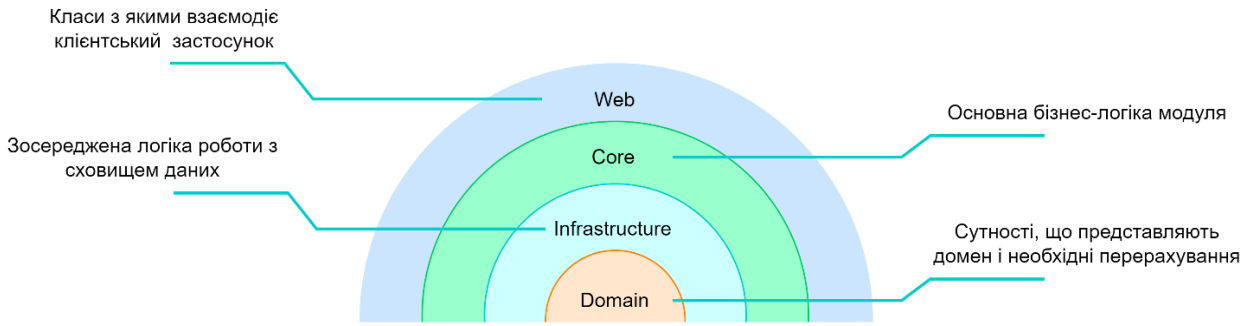

Рисунок 2.1 – Структура коду за шаблоном “Clean Architecture”

Також варто розглянути способи комунікації компонентів системи, оскільки це важливо при розробці програмного забезпечення. Для початку варто зауважити, що буде використано архітектурний стиль REST [6], при якому комунікація відбувається за допомогою HTTP-запитів, таких як GET, POST, PUT, PATCH, DELETE та деяких інших. Це є класичний та загальновідомий спосіб комунікації.

Ще одним способом комунікації компонентів застосунку є комунікація за допомогою протоколу WebSocket (рис. 2.2), який забезпечує спосіб обміну даними між браузером і сервером через спілкування в реальному часі. Дані можна передавати в обох напрямках у вигляді “пакетів”, без розриву з’єднання та додаткових HTTP-запитів [7]. Такий стиль комунікації використовується в програмах, які мають переваги від швидкого спілкування в реальному часі, наприклад у програмах для чату, інформаційної панелі та ігор.

У випадку цього проекту HTTP-запити використовуватимуться для взаємодії клієнта з модулями ідентифікації, адміністрування курсів, адміністрування контенту курсів, здачі робіт.

Модуль чату використовуватиме протокол WebSocket, для забезпечення комунікації в реальному часі.

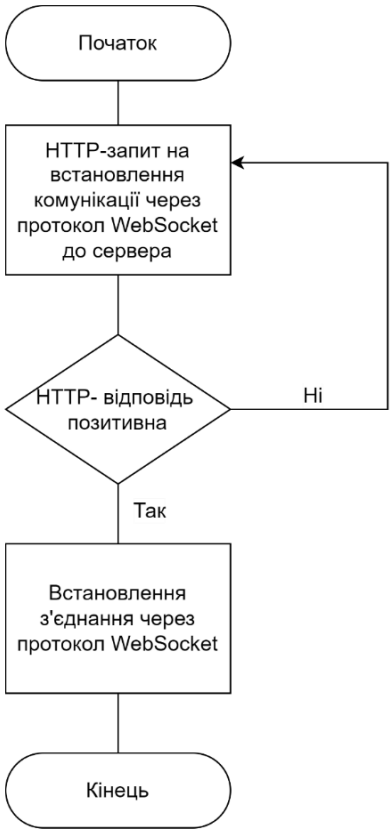

Рисунок 2.2 – Алгоритм встановлення з’єднання в протоколі WebSocket

2.2 Структурне моделювання додатку освітньої платформи

При проектуванні масштабних програмних застосунків варто звернути увагу на предметно-орієнтоване проектування [8]. Це метод моделювання складного об’єктно-орієнтованого програмного забезпечення, який відзначається концентрацією уваги на предметній області та розробкою бізнес-моделей цієї області з її глибоким розумінням [8].

Представимо загальний домен спроектованого додатку освітньої платформи у вигляді діаграми класів на рисунку 2.2. Отож предметна область містить п’ятнадцять сутностей, між якими є такі типи зв’язків:

– 1 до 0, …, n.
– 1 до 1, …, n.
– 1 до n.

Варто зауважити, що домен побудовано таким чином, щоб забезпечити легкість масштабування.

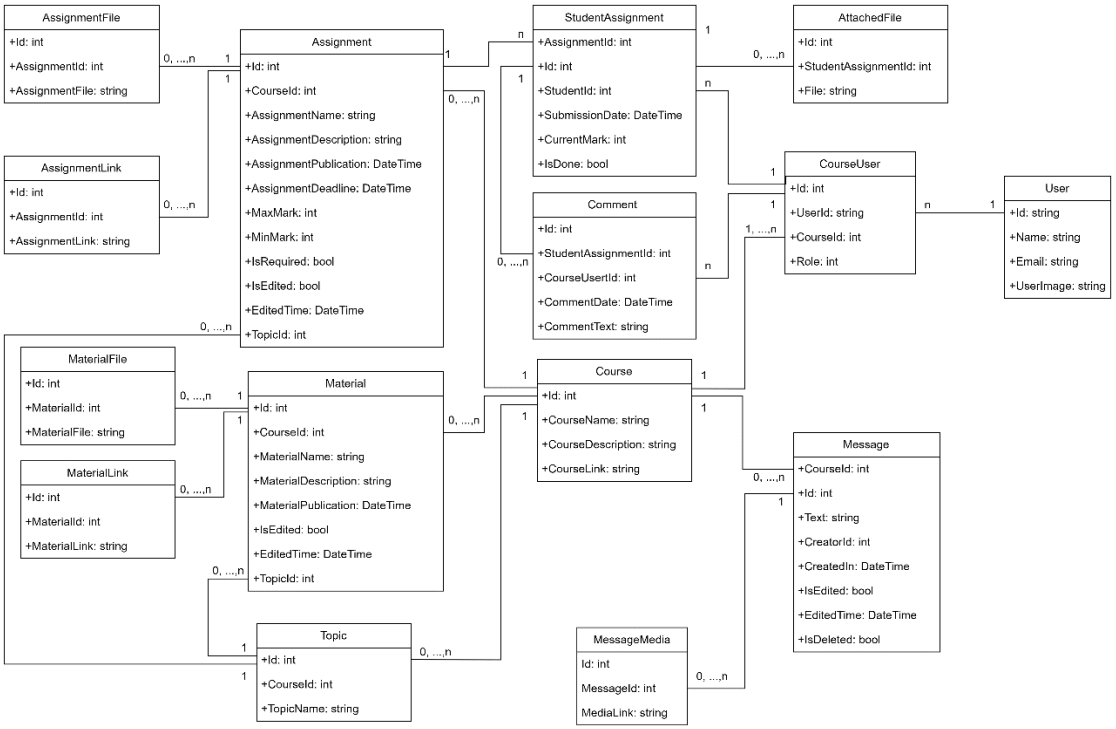

Рисунок 2.3 – Діаграма класів, що представляє домен

Розглянемо основні бізнес-моделі:

– User – представляє користувача системи.

– Course – представляє курс.

– CourseUser – представляє зв’язок між користувачем та курсом.

– Assignment – представляє завдання, яке було призначено студентам.

– Material – представляє опубліковані для студентів матеріали.

– Topic – представляє тему, до якої належать матеріали чи завдання.

– StudentAssignment – представляє завдання, яке було призначено конкретному студенту.

– Comment – представляє приватний коментар для зданого завдання.

– AssignmentFile –представляє файли прикріплені до завдання.

– AssignmentLink –представляє посилання прикріплені до завдання.

– MaterialFile – представляє файли із навчальним матеріалом.

– MaterialLink – представляє посилання прикріплені до матеріалів.

– AttachedFile – представляє прикріплені студентом до завдання файли.

– Message – представляє повідомлення, надіслане користувачем у чаті.

– MessageMedia – представляє медіа прикріплене до повідомлення.

На цьому етапі розробка бізнес-моделей закінчена, домени для реалізації бізнес-логіки застосунку сформований.

Опираючись на домен предметної області сформуємо основні модулі додатку (рис. 2.4):

– Модуль ідентифікації – призначений для обробки даних користувача, реєстрації, автентифікації та авторизації.

– Модуль адміністрування курсів – призначений для виконання основних операцій з курсами.

– Модуль адміністрування контенту курсів – призначений для виконання основних операцій з наповненням курсів.

– Чат – призначений для забезпечення комунікації між учасниками чату в реальному часі.

– Модуль здачі робіт – призначений для обліку успішності учасників курсу.

– Модуль взаємодії з користувачем – клієнтський застосунок.

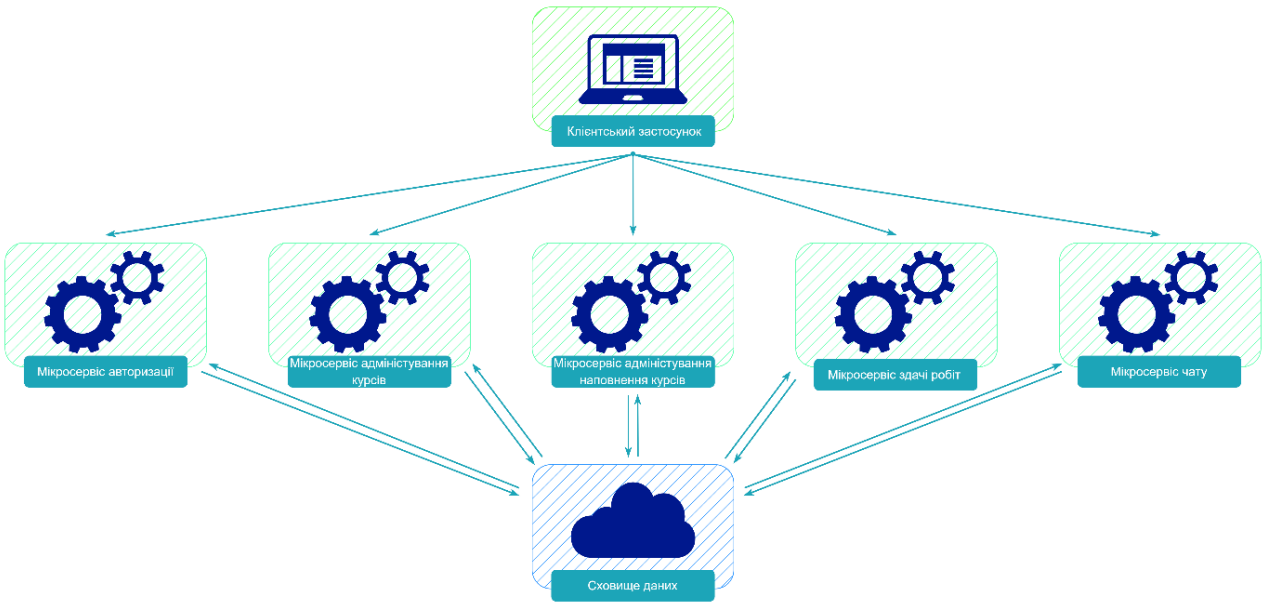

Рисунок 2.4 – Структурна схема взаємодії модулів

2.3 Висновок до розділу 2

У цьому розділі було пояснено вибір архітектури додатку та загальну структуру кожного модуля.

Основою для вибору мікросервісної архітектури є її переваги, такі як гнучкість та масштабованість, шляхом розподілу функцій між окремими сервісами. Цей підхід передбачає використання невеликих, автономних компонентів, які можуть бути розроблені, протестовані та масштабовані окремо один від одного.

Вибір структури кожного модуля базується на загальновідомих архітектурних шаблонах, які забезпечують розробку добре структурованого коду.

Вибір способів комунікації базується на конкретних задачах, які необхідно вирішити.

Також наведено діаграму класів основних бізнес-моделей предметної області. Це дозволило візуалізувати структуру системи та взаємозв’язки між її компонентами. Визначення класів та їх атрибутів дозволяє краще розуміти функціональні можливості кожного компонента та планувати подальші кроки у розробці. Наприклад, з врахуванням цих діаграм можна буде ефективно планувати реалізацію функцій, їх тестування та інтеграцію сервісів для створення повноцінної освітньої платформи.

3 РОЗРОБКА МОДУЛІВ АВТОРИЗАЦІЇ, АДМІНІСТРУВАННЯ НАПОВНЕННЯ КУРСІВ ТА ЧАТУ В РЕАЛЬНОМУ ЧАТІ

3.1 Аналіз і обґрунтування вибору засобів для реалізації

Для розробки серверної частини необхідно вибрати мову програмування. Розглянемо три найпопулярніші варіанти.

C# – це мова програмування загального призначення, розроблена компанією Microsoft [9]. Вона є об’єктно-орієнтованою, керованою, статично типізованою мовою, яка використовується для розробки різноманітних програмних рішень, включаючи:

– Веб-застосунки. Для розробки бізнес-логіки веб-застосунку існує фреймворк ASP.NET.

– Настільні програми. C# можна використовувати для створення настільних програм для Windows за допомогою програмних каркасів таких як: WinForms, WPF, UWP.

– Ігри. Unity – це популярний ігровий рушій, який підтримує C#.

– Мобільні програми. Xamarin – це платформа крос-платформної розробки мобільних програм, яка використовує C#.

JavaScript – це мова програмування сценаріїв, яка використовується для додавання динамічності та інтерактивності веб-сторінкам [10]. Вона є інтерпретованою мовою, яка виконується в браузері, і не потребує компіляції. JavaScript використовується для:

– Створення динамічних веб-сторінок. JavaScript можна використовувати для зміни вмісту веб-сторінки, реагуючи на дії користувача, такі як кліки миші або натискання клавіш.

– Розробка бізнес-логіки веб-застосунків. Фреймворк node.js використовується для розробки бізнес-логіки.

– Розробка ігор. JavaScript використовується для створення браузерних ігор.

– Розробка мобільних програм. JavaScript можна використовувати для розробки мобільних програм за допомогою фреймворків, таких як React Native і Cordova.

Python – це мова програмування загального призначення, розроблена з акцентом на читабельність коду [11]. Вона є динамічно типізованою, інтерпретованою мовою, яка використовується для:

– Розробки веб-застосунків. Python використовується для розробки веб-застосунків за допомогою фреймворків Django та Flask.

– Наукові обчислення. Python використовується для наукових обчислень і аналітики даних завдяки своїм бібліотекам NumPy, Pandas та SciPy.

– Машинне навчання. Python використовується для розробки моделей машинного навчання задопомогою бібліотек TensorFlow та scikit-learn.

– Автоматизація. Python використовується для автоматизації завдань, завдяки своїй простоті та можливостям сценаріїв.

Для розробки бекенд частини обрано мову програмування C#, оскільки вона є універсальною мовою програмування, яка пропонує низку переваг, до яких можна віднести як особливості мови C# так і особливості платформи .NET. Розглянемо основні з них:

– Статична типізація. Типи даних змінних та виразів перевіряються на етапі компіляції. Це допомагає виявити помилки типу ще до того, як програма буде запущена, що може заощадити час і зусилля на налагоджені. Варто зауважити, що в C# є ключове слово “dynamic”, що дозволяє використовувати динамічну типізацію, де це необхідно.

– Підтримка ООП. Повноцінно підтримує об’єктно-орієнтоване програмування, включаючи всі його основні принципи: абстракцію, наслідування, поліморфізм та інкапсуляцію.

– Продуктивність. C# є компільованою мовою, що означає, що код перетворюється на машинний код перед виконанням. Тому виконавча швидкість коду C# дозволяє додати її до найшвидших високорівневих мов програмування.

– Набір інструментів для бекенд-розробки. До них відносяться, наприклад, такі інструменти: ASP.NET Core – програмний каркас з високою продуктивністю, Entity Framework Core – об’єктно-реляційне відображення для роботи з даними.

Отже підсумовуючи вищесказане, C# є оптимальним вибором для розробки бізнес-логіки цього програмного застосунку.

3.2 Розробка програмного модуля авторизації

Для реалізації цього модуля використаємо AWS Cognito. Amazon Cognito – це ідентифікаційна платформа для веб- і мобільних додатків. Він містить каталог користувачів, сервер автентифікації та службу авторизації для токенів доступу OAuth 2.0 і облікових даних AWS [12]. За допомогою Amazon Cognito можна реалізувати автентифікацію та авторизацію користувачів із вбудованого каталогу користувачів. Також він підтримує інших постачальників ідентифікаційних даних, таких як Google і Facebook. [13]

Варто зауважити, що для реалізації методів авторизації та ідентифікації за допомогою Cognito, необхідно використати IAmazonCognitoIdentityProvider з набору бібліотек AWS SDK.

Розглянемо загальну структуру модуля, яка наведена на рисунку 3.1. Отож як було сказано в розділі проектування, модуль ділиться на чотири рівні, які ієрархічно залежать один від одного в такому порядку:

– Domain – не має залежностей.
– Infrastructure – залежить від Domain.
– Core – залежать від Domain та Infrastructure.
– Web – залежить від Core та містить основну точку входу в модуль.

Розглянемо взаємодію класів кожного рівня, окрім Domain, оскільки в цьому модулі не потрібно великої кількості бізнес-моделей. Необхідною для реалізації мікросервісу є тільки модель User, яка відображає бізнес-користувача.

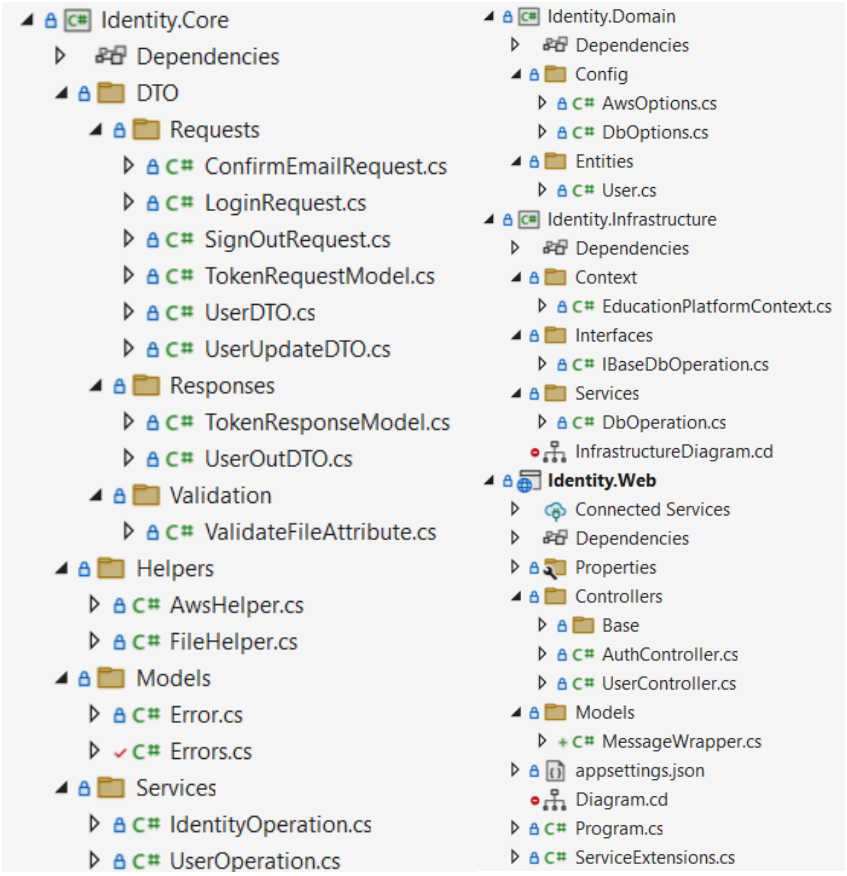

Рисунок 3.1 – Файлова структура модуля авторизації

Рівень інфраструктури, який представлений на рисунку 3.2, містить всього два класи EducationPlatformContext, DbOperation та інтерфейс IBaseDbOperation. Розглянемо детальніше:

– EducationPlatformContext – це підклас DbContext з Microsoft Entity Framework Core, що представляє контекст сховища даних для освітньої платформи.

– IBaseDbOperation – це інтерфейс, який визначає базові операції над даними для будь-якого типу T, який є класом. Методи включають додавання, оновлення, видалення та отримання за ідентифікатором.

– DbOperation – це реалізація інтерфейсу IBaseDbOperation<T> для типу User.

Розглянемо рівень Core (рис. 3.3), на якому реалізована основна логіка роботи модуля авторизації. Через велику кількість класів, розглянемо їх функції через розташування в конкретних директоріях:

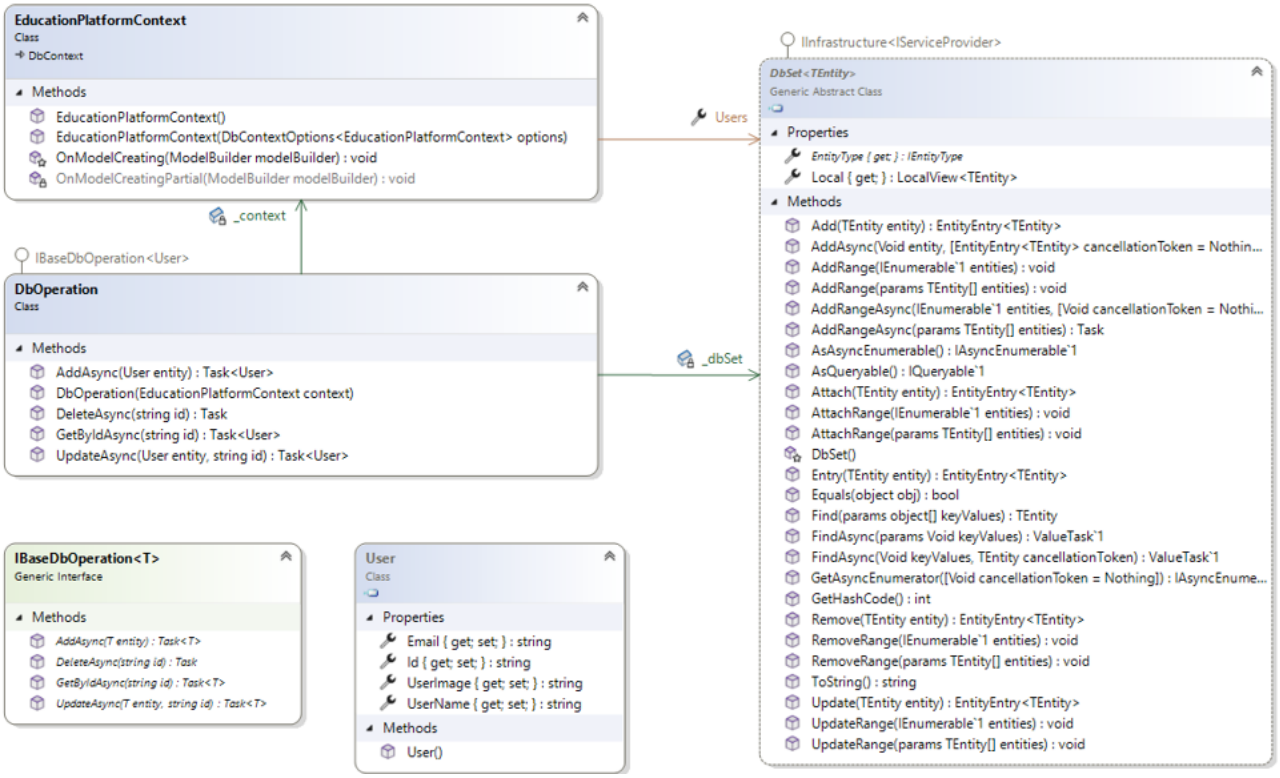

Рисунок 3.2 – Взаємодія класів рівня Infrastructure

– DTO (Data Transfer Objects) – це об’єкт, який визначає спосіб надсилання даних через мережу [14]. Тому в цій директорії містяться вхідні та вихідні моделі даних, а також додаткова валідація вхідних даних. В свою чергу тут містяться піддиректорії: Requests, Responses, Validation, для покращення структурованості.

– Helpers – класи для загальних завдань, наприклад, робота з файлами чи з AWS, як в наведеному прикладі. Практично це методи, які можуть використовуватися кількома класами в програмі.

– Services – класи для реалізації складної логіки, яку використовують інші компоненти програми. Сюди відноситься бізнес-логіка, наприклад, реалізація методів авторизації та взаємодії з даними користувача.

– Models – директива для класів, що реалізують додаткові моделі необхідні для реалізації методів на рівні Core та Web.

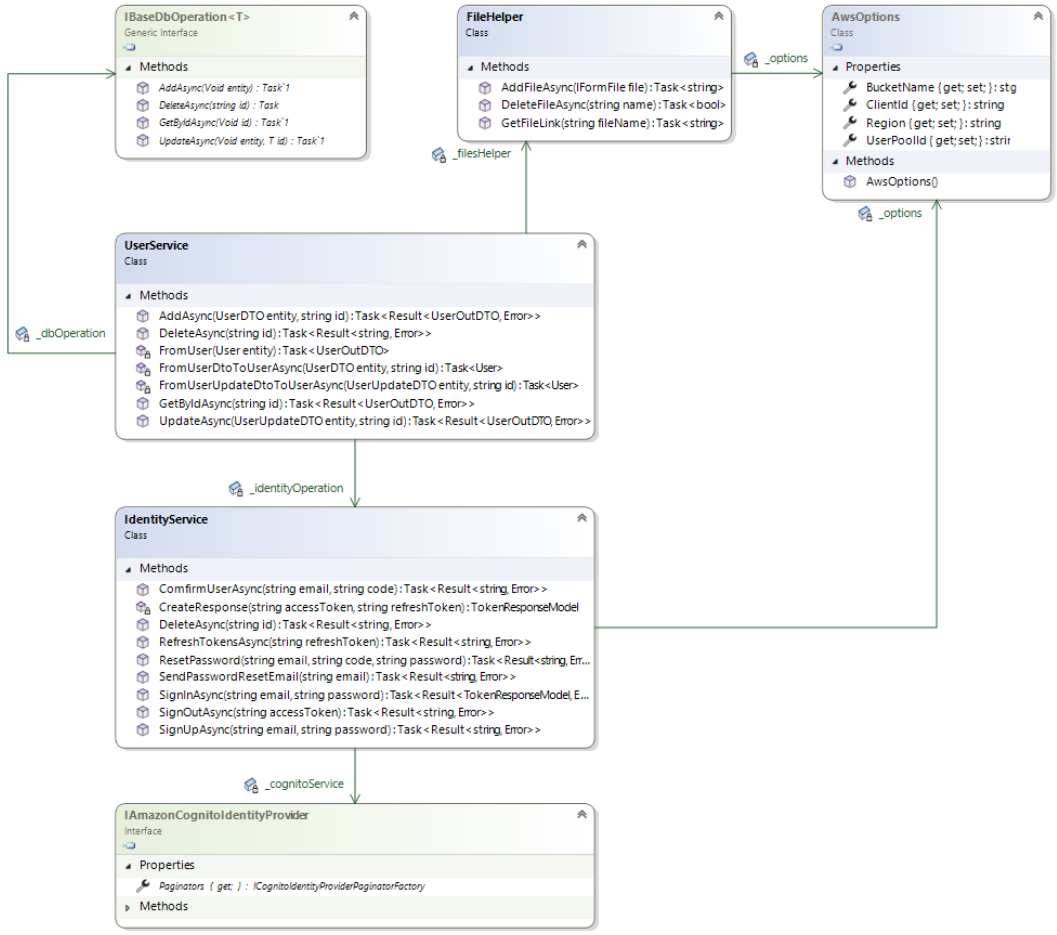

Рисунок 3.3 – Взаємодія класів рівня Core

Варто зауважити, що всі проекти цього модуля, окрім Identity.Web відповідають типу “Бібліотека класів”, тобто вони не містять точки входу, але реалізовані в них методи виконуються після виклику.

Identity.Web містить точку входу – статичний метод Main(), який розташований у класі Program. Варто зауважити, що в вищезгаданому методі реєструються необхідні сервіси в контейнер впровадження залежності [15]. Розглянемо інші класи цього рівня, взаємодія яких наведена на рисунку 3.4:

– BaseController – це клас контролера, що розширює набір методів стандартного базового класу Controller для всіх інших контролерів у програмі.

– AuthController – це клас, що використовується для обробки запитів, пов’язаних з автентифікацією користувачів.

– UserController: – це клас, що використовується для обробки запитів, пов’язаних з бізнес-даними користувачів.

– MessageWrapper – клас, де описана модель відповіді API.

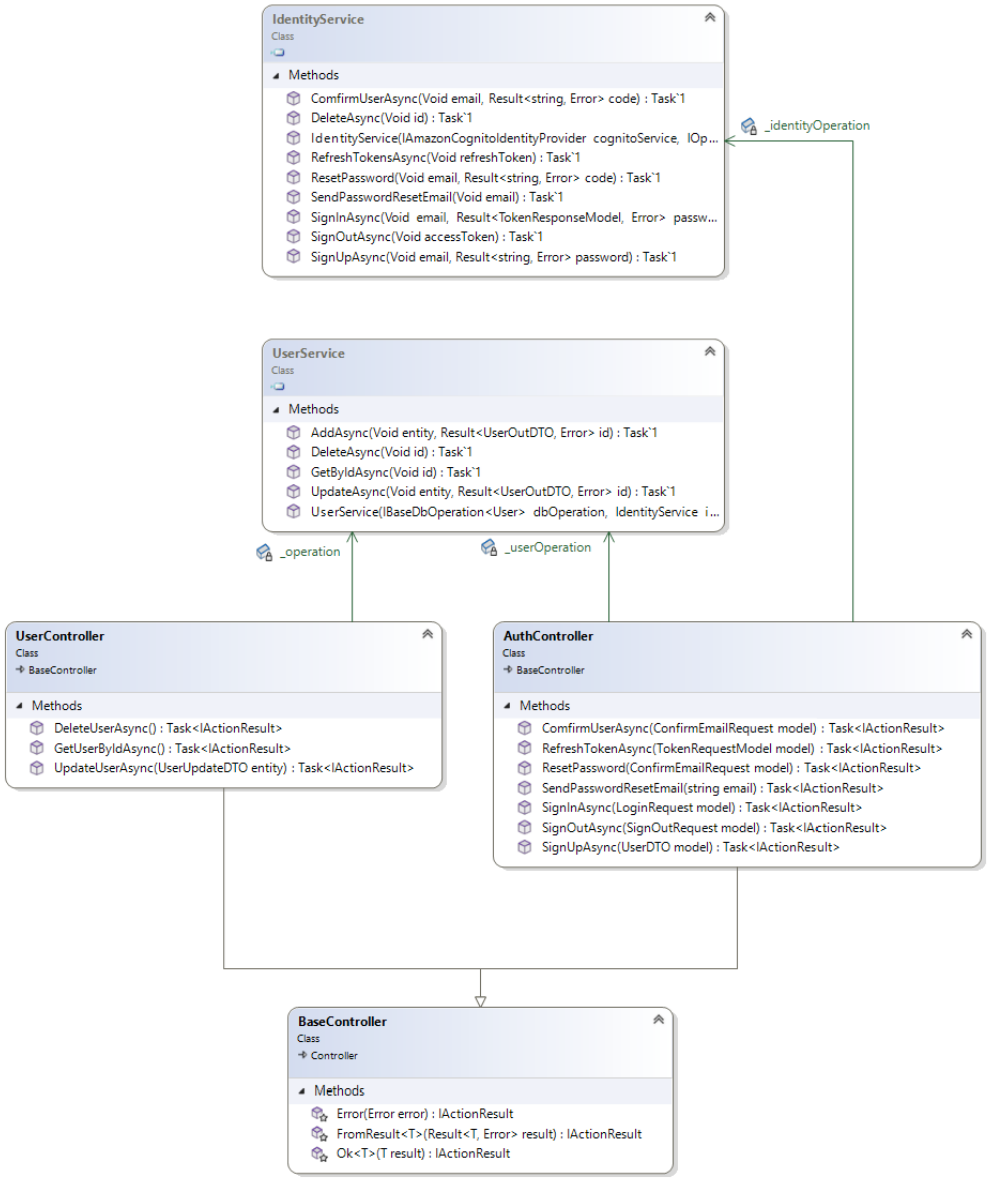

Рисунок 3.4 – Взаємодія класів рівня Web

На цьому розробку модуля авторизації, можна вважати завершеною.

3.3 Розробка програмного модуля адміністрування наповнення курсів

Як можна побачити на рисунку 3.5, файлова структура цього модуля не відрізняється від попереднього. Для того щоб не повторюватися, розглянемо тільки те, що відрізняється від попереднього проекту.

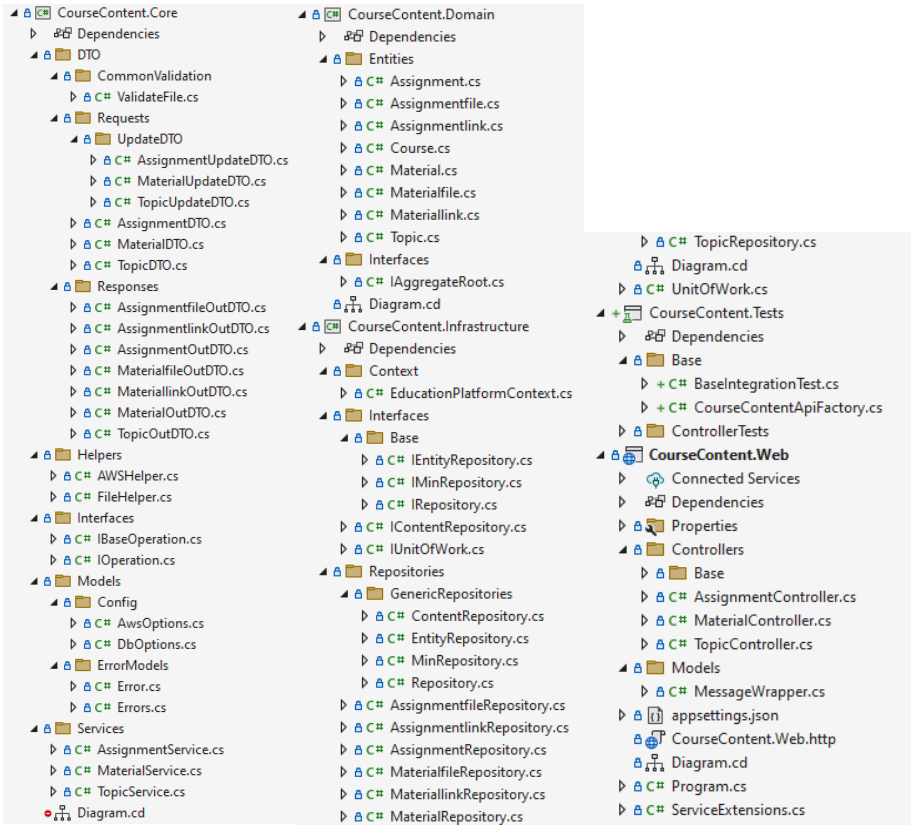

Рисунок 3.5 – Файлова структура модуля адміністрування контенту

Для подальшого розуміння особливостей взаємодій певних класів, розглянемо сутності, які необхідні для реалізації функцій цього модуля. На рисунку 3.6 зображену взаємодію основних сутностей.

Із особливостей тут присутнє використання маркерного інтерфейсу. Його можна використовувати для виконання дизайнерських рішень в коді. Наприклад, можна змінити інтерфейс сховища, щоб він працював лише з типами IAggregateRoot, а не з будь-якими типами Entity або BaseEntity, що значно полегшить дотримання правила, згідно з яким класи-нащадки в агрегатах ніколи не зберігаються окремо, а лише як частина цілого агрегата.

Отож тепер перейдемо до розгляду рівня інфраструктури в цьому модулі. Розпочнемо з того, що для взаємодії із сховищем реалізовано ще один шар абстракції у вигляді інкапсуляції роботи з даними за допомогою шаблонів проектування “Unit of work” та “Repository”. Впровадження цих шаблонів може допомогти ізолювати програму від змін у сховищі даних і полегшити автоматизоване модульне тестування або тестову розробку

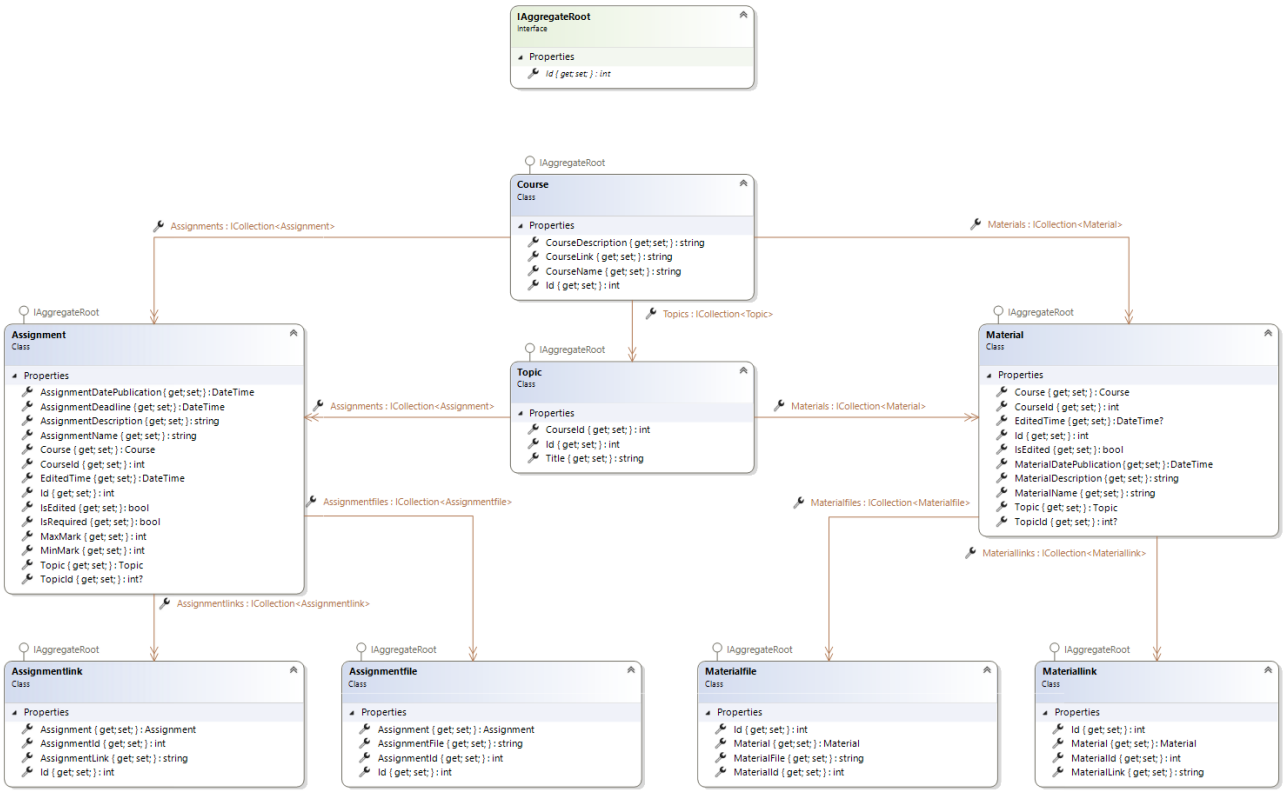

Рисунок 3.6 – Взаємодія класів рівня Domain

Шаблон “Unit of work” використовується для керування транзакціями та забезпечення того, що кілька операцій розглядаються як одна логічна одиниця. Він надає спосіб групувати операції з сховищем даних разом, гарантуючи, що вони або успішні, або невдалі в цілому. Ключовий принцип шаблону одиниці роботи полягає в підтримці узгодженості та цілісності даних шляхом узгодженого внесення або відміни змін [16].

Шаблон “Repository” – шаблон розробки програмного забезпечення, який діє як посередник між бізнес-логікою та сховищем даних. Він забезпечує узгоджений інтерфейс для виконання CRUD-операцій (створення, читання, оновлення, видалення) над об’єктами даних, одночасно інкапсулюючи складність доступу до даних. Це розділення покращує модульність, можливість повторного використання коду, полегшує розробку та підтримку програмного коду [17].

Розглянемо на рисунку 3.7 взаємодію класів, що реалізовують взаємодію зі сховищем даних, з використанням вище описаних шаблонів.

Як видно на діаграмі, для реалізації репозиторіїв використано чотири інтерфейси. Це зроблено для дотримання одного з принципів SOLID [18], а саме принципу “Interface Segregation Principle”, який каже, що нащадки не повинні залежати від методів, які вони не використовують. Загалом IEntityRepository, IMinRepository та IRepository реалізовують базові репозиторії, а IContentRepository – специфічний. Специфічний репозиторій – це репозиторій для певної сутності, де визначається її додаткова особлива поведінка.

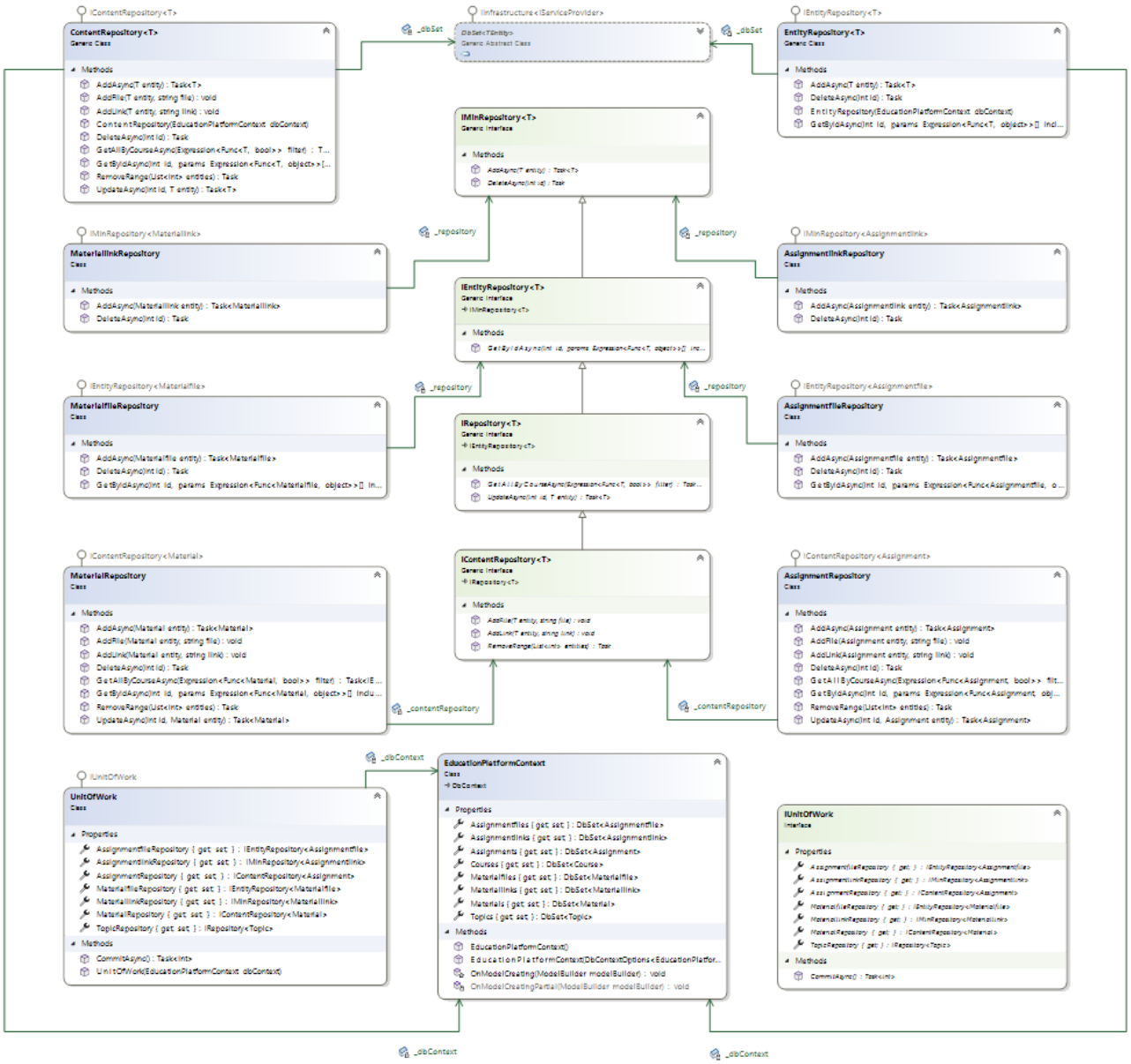

Рисунок 3.7 – Взаємодія класів рівня Infrastructure

В свою чергу, при використанні шаблону “Repository” з шаблоном “Unit of Work”, другий відповідає за керування репозиторіями, які забезпечують доступ до даних. Це дозволяє створити централізовану точку доступу до всіх репозиторіїв у додатку, спрощуючи управління даними та забезпечуючи узгодженість даних під час виконання операцій.

Завдяки шаблону “Unit of Work” не потрібно створювати екземпляр кожного окремого репозиторію на рівні Core. Замість цього, достатньо створити лише один екземпляр інтерфейсу IUnitOfWork, який включає всі необхідні репозиторії для взаємодії з даними.

Для закріплення розуміння коротко про інфраструктуру:

– Context – як і в попередньому модулі, містить контекст для роботи зі сховищем даних.

– Interfaces – директорія, що містить інтерфейси для різних компонентів рівня інфраструктури.

– Repositories – директорія, що містить реалізацію всіх репозиторіїв для кожної сутності, яка цього потребую. В свою чергу має піддиректорію GenericRepositories, де розташовані репозиторії загального типу, тобто такі, що можуть приймати об’єкти різних типів.

Тепер розглянемо рівень Core взаємодію класів якого наведено на рисунку 3.7. Варто зауважити, що не всі особливості були розглянуті, при описі модуля авторизації, тому згадаємо про це тут.

Одна з особливостей сервісів на цьому рівні – це обробка помилок. Оскільки генерація винятків доволі затратний варіант з точки зору продуктивності, то було прийнято рішення використати шаблон Result в поєднані з ExceptionHandler ASP. NET Core для виняткових ситуацій.

Шаблон “Result” є альтернативою обробці помилок на основі винятків. Він дозволяє уникнути генерації винятків і натомість повертає спеціальний об’єкт результату. Цей об’єкт вказує на успіх або невдачу операції, та може містити додаткову інформацію про помилку. Це дозволяє реалізовувати код, який є більш чистим та передбачуваним у виконанні. Розробники можуть

перевірити результат і точніше обробляти помилки. Він особливо корисний, для перехоплення очікуваних помилок [19].

Переваги шаблону “Result”:

– Явна обробка помилок. Розробники мають явно визначати, як обробляти успішний або невдалий результат операції.

– Чітка читабельність. Використання цього шаблону дозволяє уникнути глибоко вкладених блоків try-catch, що робить код більш зрозумілим та легким для читання та розуміння.

– Покращення продуктивності. Уникненя генерації винятків та явна обробка результатів може покращити продуктивність програми.

Для ще більш ефективного використання цього шаблону розроблено власний об’єкт помилки, що містить статус код та повідомлення. При виникненні передбачених виняткових ситуацій, об’єкт помилки обгортається у об’єкт відповіді API, і передається клієнту.

UseExceptionHandler – це метод, який використовується в ASP.NET Core для обробки помилок на рівні middleware. Цей метод дозволяє встановити обробник винятків, який буде викликатися, коли виникає помилка у додатку ASP.NET Core.

При використанні метода UseExceptionHandler, можна налаштувати обробник, який буде відповідати за логування помилок, відправку відповіді клієнту з відповідним HTTP статусом та інші дії, пов’язані з обробкою винятків. В моєму випадку він відповідає за обробку усіх непередбачуваних помилок, та повертає на клієнтський застосунок статус код 500.

Окрім цього варто ще згадати про шаблон “Option”. Цей шаблон використовує класи для надання строго типізованого доступу до груп пов’язаних параметрів [20].

В цьому випадку клас AwsOption, який містить основні параметри необхідні для використання сервіс AWS S3, отримує дані з AWS System manager [21] через інтерфейс IOption.

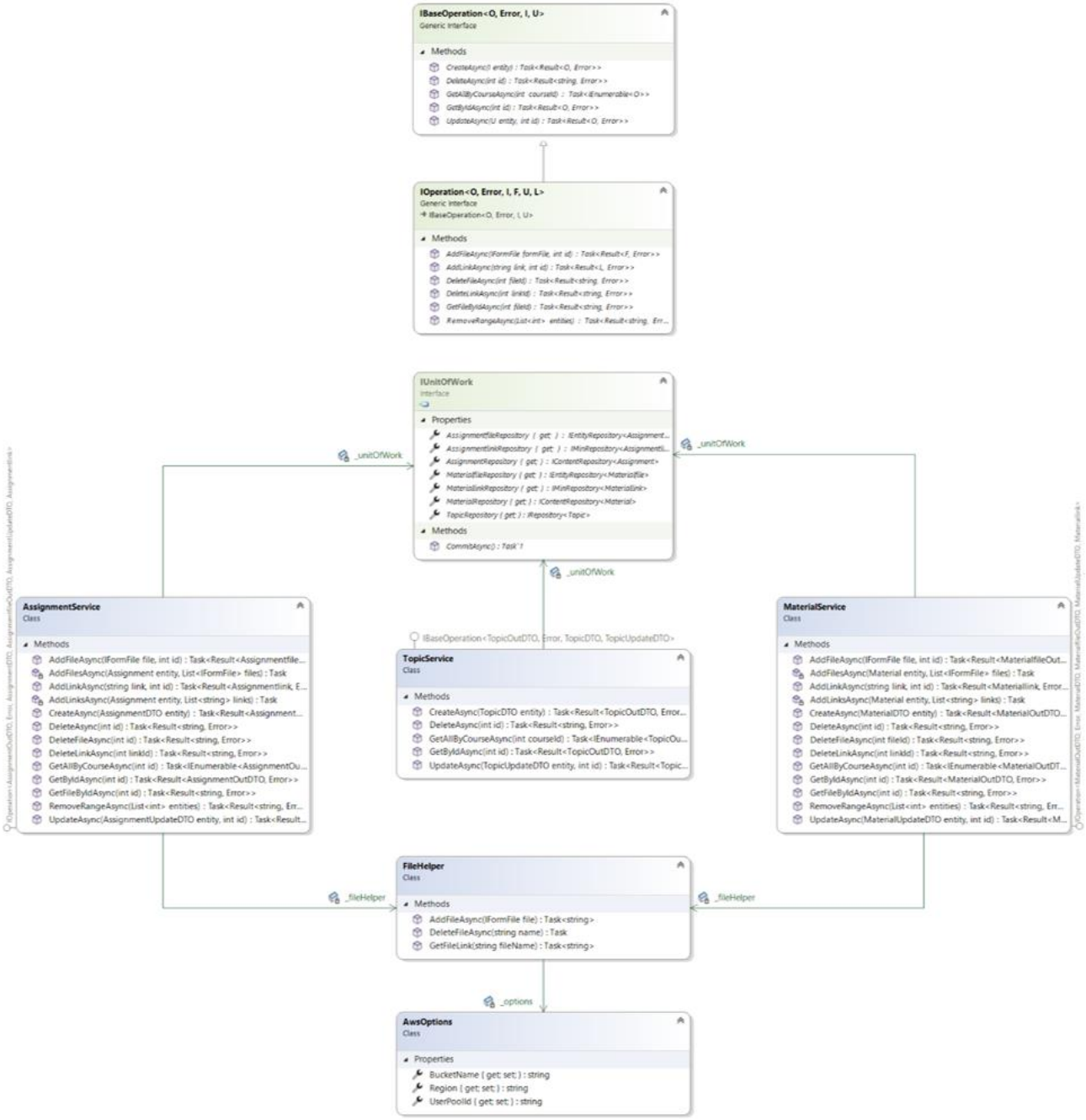

Рисунок 3.8 – Взаємодія класів рівня Core

Тепер розглянемо взаємодію класі рівня Web на рисунку 3.8. Як видно з діаграми класі, взаємодія елементів цього рівня з елементами рівня Core ідентична до розглянутої в модулі авторизації.

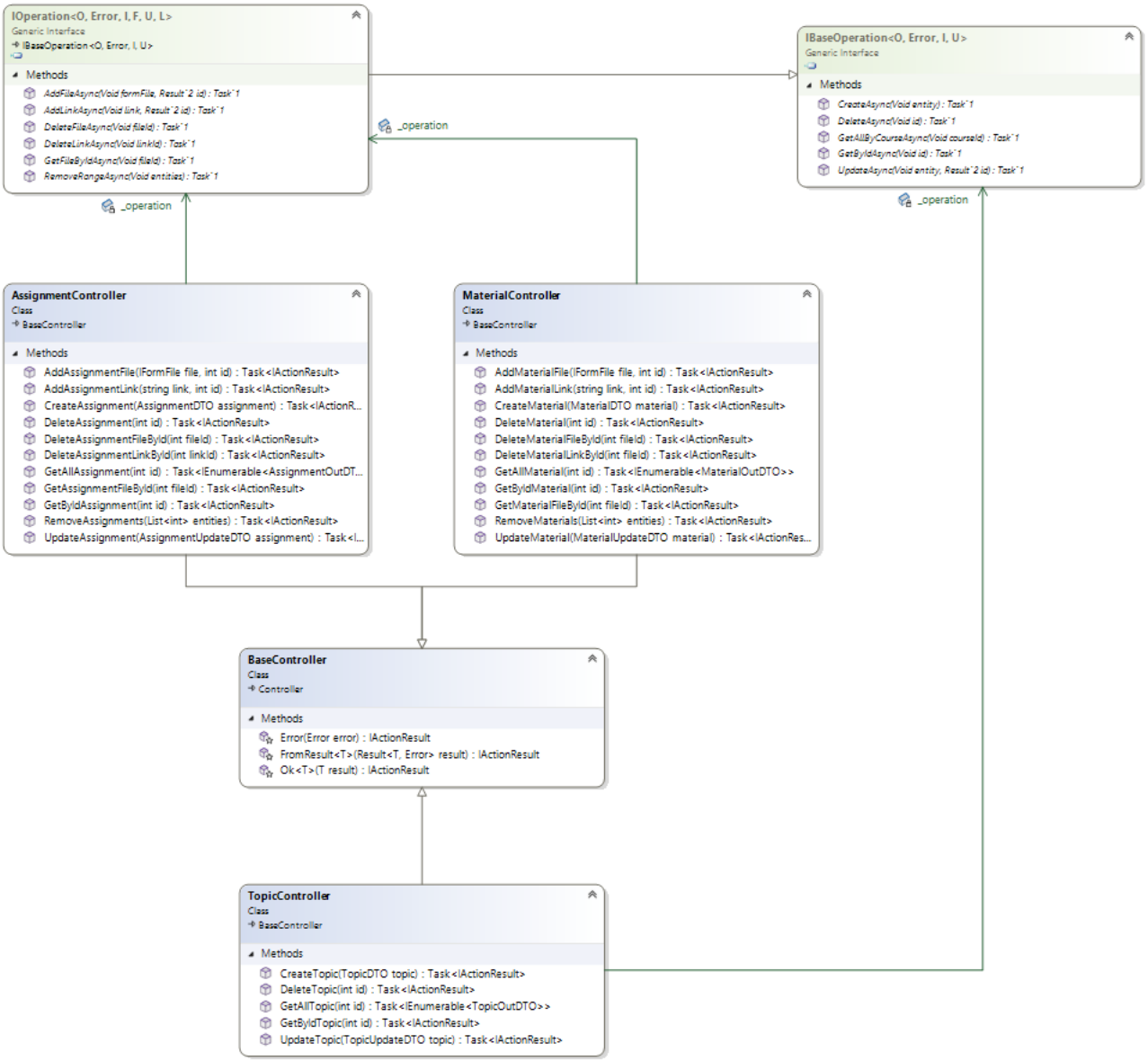

Рисунок 3.9 – Взаємодія класів рівня Web

На цьому розробку модуля адміністрування контенту, можна вважати завершеною.

3.4 Розробка програмного модуля чату в реальному часі

Для розробки чату в реальному часі застосовано усі структурні підходи описані вище, тобто рівні Domain, Infrastructure та Core відповідають за те, що й в попередніх модулях, а от рівень Web має свою особливість. Розглядати детально усі рівні не будемо, просто для контексту наведемо файлову структуру на рисунку 3.10 та діаграми класів на рисунках 3.11-3.14.

Рисунок 3.10 – Файлова структура модуля чату в реальному часі

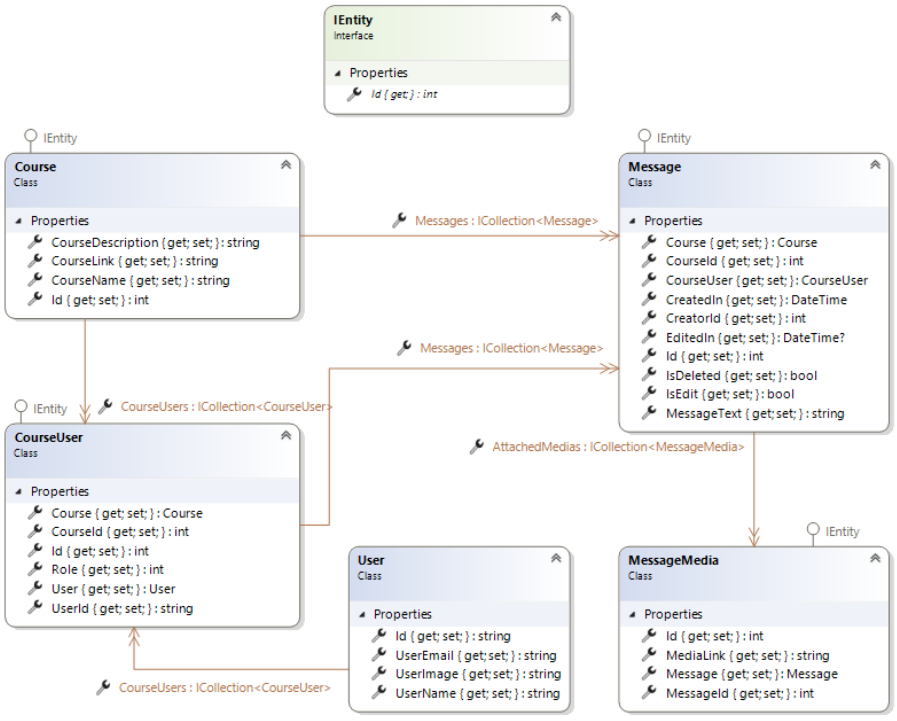

Рисунок 3.11 – Взаємодія класів рівня Domain

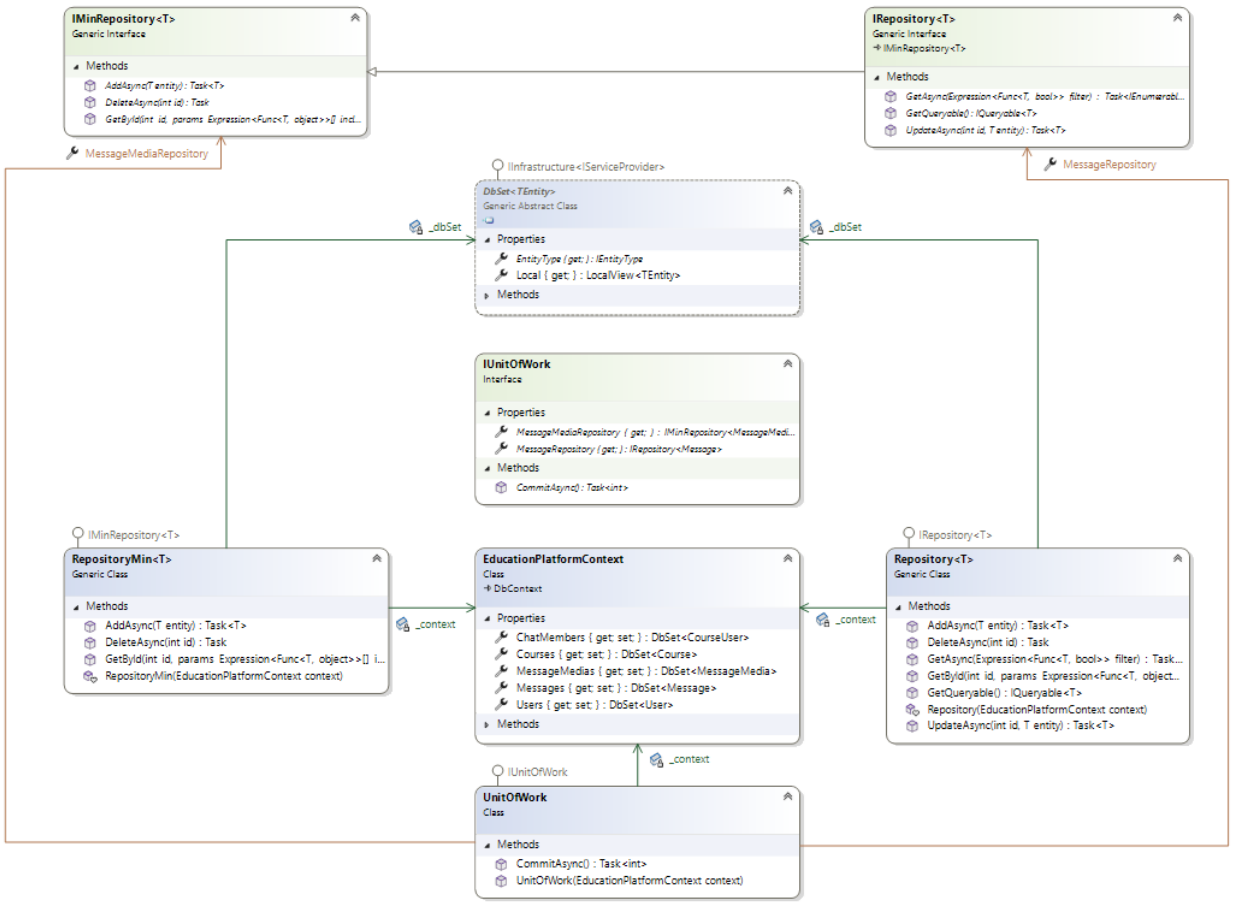

Рисунок 3.12 – Взаємодія класів рівня Infrastructure

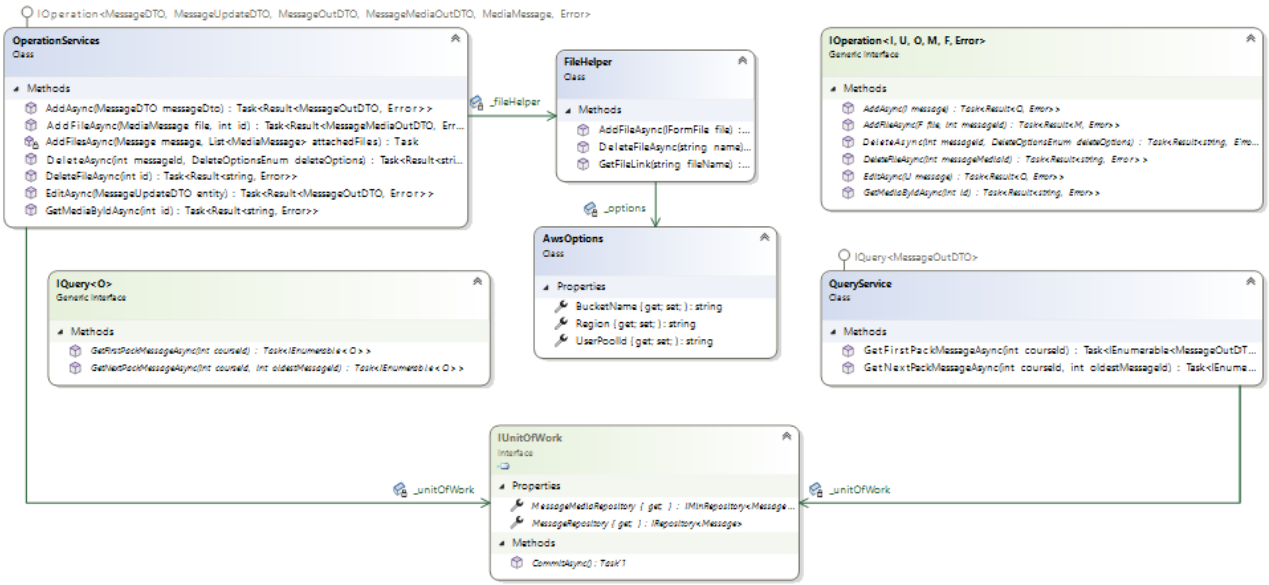

Рисунок 3.13 – Взаємодія класів рівня Core

На базі вже згаданого протоколу WebSocket заснована робота ASP.NET Core SignalR – бібліотеки з відкритим вихідним кодом, яка спрощує додавання

веб-функціональності в режимі реального часу до програм. Веб-функції в режимі реального часу дозволяють коду на стороні сервера миттєво передавати вміст клієнтським застосункам [22].

Ось деякі функції SignalR для ASP.NET Core [22]:

– Автоматично керує підключенням.

– Надсилає повідомлення всім підключеним клієнтам одночасно.

– Надсилає повідомлення певним клієнтам або групам клієнтів.

– Масштабується для обробки зростаючого трафіку.

Серед особливостей написання бізнес-логіки з використанням цієї бібліотеки є використання Hub для зв’язку між клієнтами та сервером, що є аналогом використання Controller при підході REST.

Hub – це високорівневий пайплайн, який дозволяє клієнту та серверу викликати методи один одного [22]. Він ініціює виконання коду на стороні клієнта, відправляючи повідомлення, що містять назву методу та його параметри на стороні клієнта. Об'єкти, які передаються як параметри методу, десеріалізуються за допомогою встановленого протоколу. Клієнт намагається знайти відповідність назви методу у своєму коді. Якщо збіг знайдено, клієнт викликає метод та передає йому десеріалізовані параметри [22].

Взаємодія Hub з бізнес-логікою застосунку наведена на рисунку 3.14.

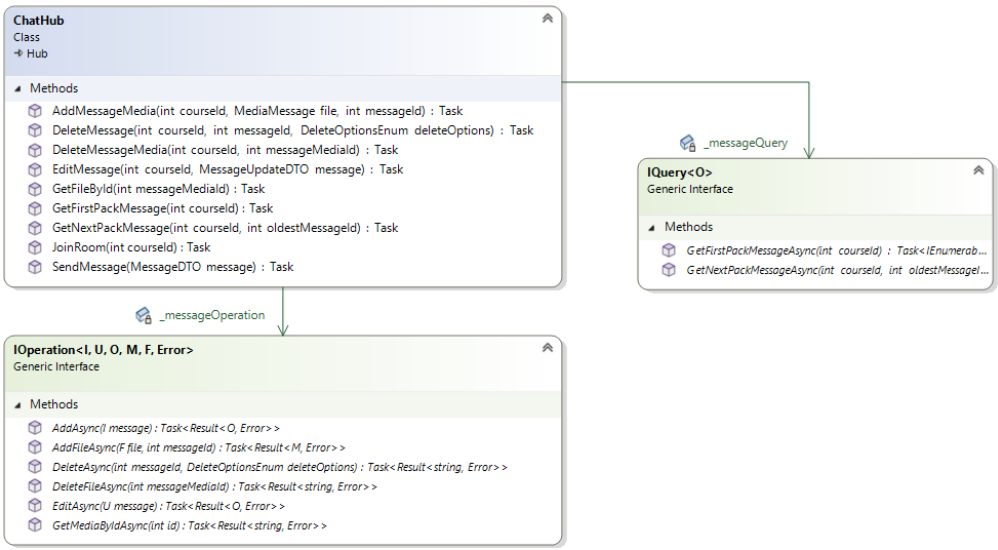

Рисунок 3.14 – Взаємодія класів рівня Web

3.5 Висновок до розділу 3

У цьому розділі аргументовано вибір мови програмування для реалізації проекту, та пояснено основні етапи розробки.

Отже обрано мову програмування C#, зважаючи на її основні переваги: статична типізація, повна підтримка об’єктно-орієнтованого програмування, наявність великої кількості інструментів для розробки застосунків такого типу.

Розглянуто розробку модулів авторизації, адміністрування контенту та чату в реальному часі. Наведено діаграми класів взаємодії компонентів кожного рівня для кожного модуля. Описано основні шаблони проектування, які було використано для реалізації програмного коду та пояснено яким чином вони використовувалися й для чого, продемонстровано особливості взаємодії між компонентами застосунку.

4 ТЕСТУВАННЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ МОДУЛІВ ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ

4.1 Методи тестування програмного забезпечення

Методи тестування програмного забезпечення включають різноманітні стратегії та підходи до перевірки якості програм. Наведемо кілька основних методів тестування:

– Модульне тестування. Тестування окремих модулів або компонентів програми. Це дозволяє перевіряти правильність роботи окремих частин програми без їх інтеграції з іншими.

– Інтеграційне тестування. Тестування взаємодії між різними модулями чи компонентами програми. Мета – переконатися, що вони правильно працюють після їх об’єднання.

– Системне тестування. Тестування програми як цілісної системи, щоб переконатися в її відповідності вимогам та очікуванням користувачів.

– Функціональне тестування. Тестування функціональності програми згідно зі специфікаціями. Це включає перевірку реакції програми на різні вхідні дані та умови.

– Нефункціональне тестування. Тестування характеристик програми, які не відносяться безпосередньо до її функціональності, наприклад, продуктивність, безпека, відмовостійкість тощо.

– Тестування відновлення. Повторне тестування програми після внесення змін чи покращень для впевненості, що вони не вплинули на роботу інших частин програми.

– Автоматизоване тестування. Використання програм для виконання тестів, що дозволяє швидше та ефективніше проводити тестування.

– Альфа- та бета-тестування. Тестування програми користувачами на ранніх та пізніх стадіях розробки відповідно. Альфа-тестування – внутрішнє

тестування, а бета-тестування – зовнішнє, з використанням допомоги реальних користувачів.

Для тестування розроблених модуль проведемо модульне, інтеграційне та функціональне тестування. Ці види тестування допоможуть впевнитися у коректності роботи розроблених модулів.

4.2 Модульне тестування розробленого програмного продукту

Основна мета модульних тестів – перевірити коректність роботи коду в ізоляції від інших частин програми, що дозволяє виявити та виправити помилки на ранніх етапах розробки.

Для написання модульних тестів можна використовувати різноманітні інструменти тестування, такі як бібліотеки nUnit, xUnit, MSTests та інші, які є для мови програмування .NET.

Для написання цього виду тестів використаємо бібліотеку xUnit – безкоштовний інструмент тестування з відкритим вихідним кодом для .NET [23].

Один з ключових аспектів модульного тестування – використання імітація. Вони дозволяють створювати умовні об’єкти, які імітують реальні об’єкти або середовище, щоб перевірити поведінку тестованого коду в різних умовах без необхідності використання реальних ресурсів або сервісів. Такий підхід дозволяє ефективно та повноцінно тестувати окремі частини програми, забезпечуючи високу якість та надійність розробленого програмного продукту.

В процесі написання тестів, через відсутність надскладної логіки, було прийнято рішення не розширювати кодову базу елементарними тестами, тому повний набір модульних тестів отримали сервіси модуля ідентифікації. Розглянемо оглядач тестів, який представлено на рисунку 4.1.

Тести IdentityServiceTests представлені у вигляді перехоплень викликів методів інтерфейсу IAmazonCognitoIdentityProvider, що належить AWS SDK, і залежно від ситуації повертає генерацію виняткової ситуації, або успішний результат.

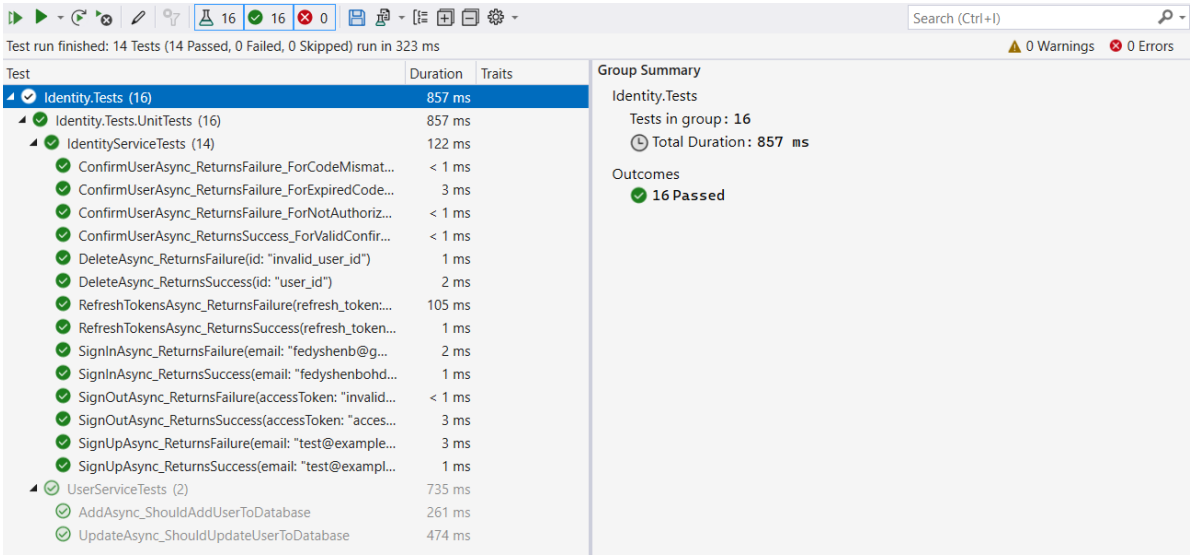

Рисунок 4.1 – Оглядач тестів в Visual Studio 2022

Для тестування методів UserService, використаємо інструмент Testcontainers – фреймворк із відкритим вихідним кодом для одноразових легких екземплярів баз даних, брокерів повідомлень, веб-браузерів або всього, що може працювати в контейнері Docker [24]. В Docker [25] розгортається контейнер з тестовим сховищем даних, що дозволяє імітувати роботу сервісу з ним. Після завершення тестів контейнер зупиняється.

Розглянемо додаткове виведення цих методів тестування рисунку 4.2.

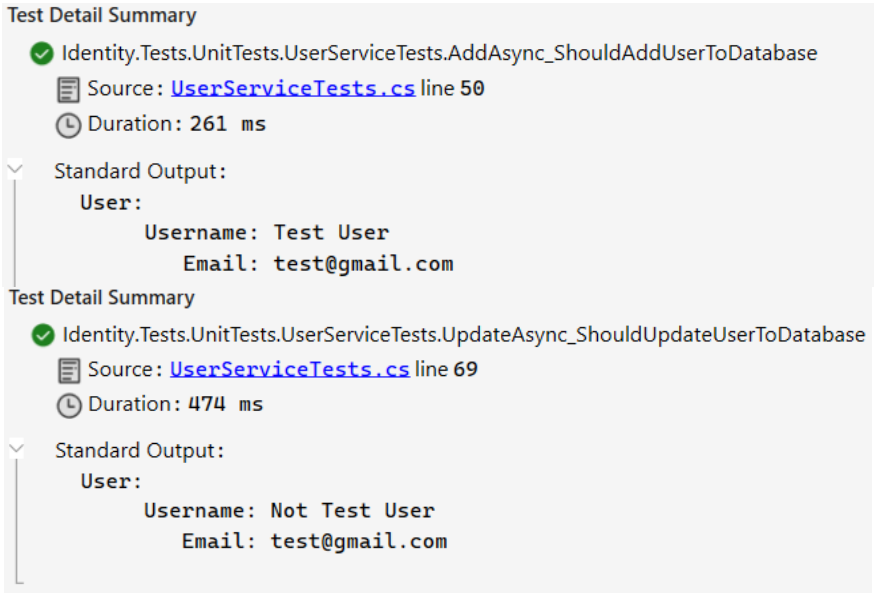

Рисунок 4.2 – Успішне виконання тестів з використанням Testcontainers

На цьому етапі розробку модульних тестів можна вважати завершеною, оскільки цей метод тестування не найдоцільніший в цьому випадку.

4.3 Інтеграційне тестування розробленого програмного продукту

Інтеграційне тестування призначене для перевірки взаємодії між компонентами програмного забезпечення для виявлення помилок на ранніх етапах розробки. На цьому етапі використання імітацій є недопустим. Отож необхідні сервіси потрібно буде розгортати локально та звертатися до них.

Для написання інтеграційних тестів окрім Testcontainers, потрібно ще використати інструмент LocalStack. Це емулятор хмарної служби, який працює в одному контейнері на персональному пристрої або в середовищі неперервної інтеграції. За допомогою LocalStack можна запускати сервіси від AWS повністю на локальній машині без підключення до віддаленого хмарного постачальника [26].

Варто зауважити, що для використання усіх сервісів потрібно купити ліцензію, але для виконання інтеграційного тестування цього застосунку використається тільки AWS S3, який є в безкоштовній версії.

Розглянемо як це працює на прикладі тестування двох методів CreateAssignment (рис. 4.3) та UpdateAssignment (рис. 4.4).

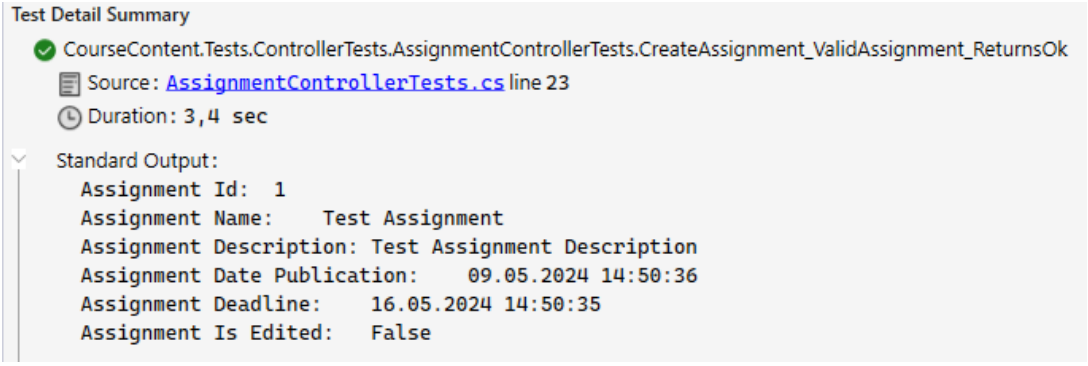

Рисунок 4.3 – Результат виконання тесту для методу CreateAssignment

В цьому випадку тести запускаються послідовно, хоча бібліотека дозволяє робити паралельне виконання різних тестів.

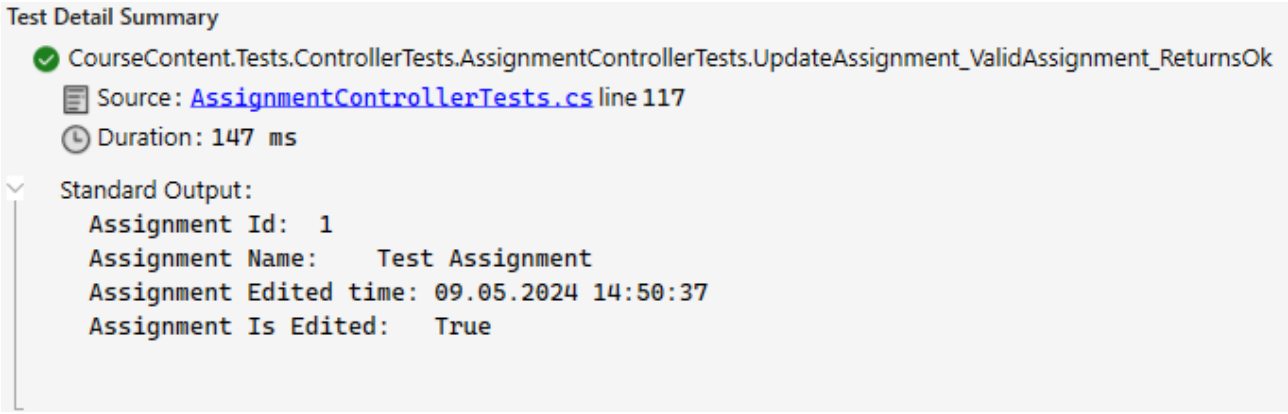

Рисунок 4.4 – Результат виконання тесту для методу UpdateAssignment

Після аналізу цих результатів, можна зробити висновок, що взаємодія сховища даних та бізнес-логіки коректна й все працює правильно.

Ще до особливостей реалізації інтеграційних тестів можна додати, що інфраструктура до тестування запускається в контейнерах разом з запуском тестів й очищується після їх закінчення, як видно з рисунку 4.5.

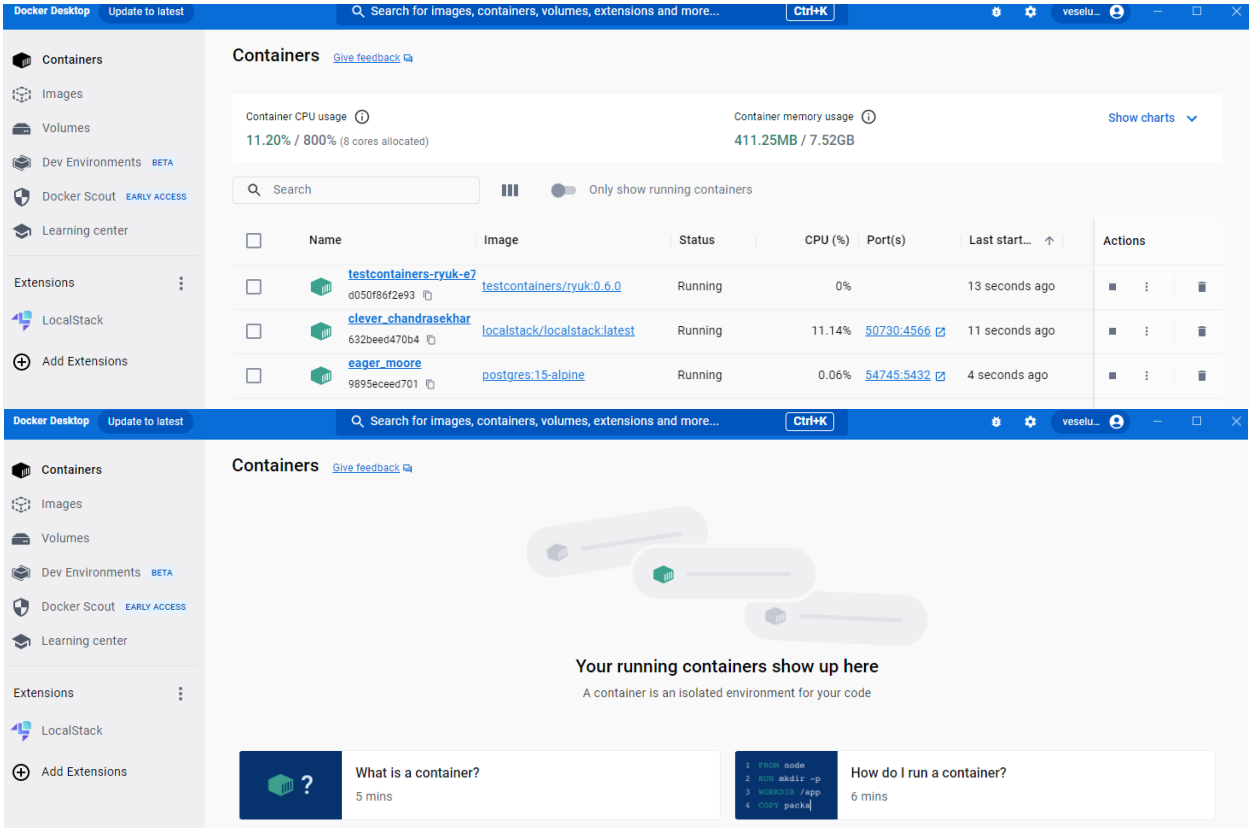

Рисунок 4.5 – Розгортання тестової інфраструктури в середовищі Docker

4.4 Функціональне тестування розробленого програмного продукту

Для перевірки коректності роботи модулів, протестуємо їх локально з використанням інструменту Postman.

Postman – це інструмент для тестування API (прикладний програмний інтерфейс). Він дозволяє розробникам створювати, відправляти та отримувати HTTP-запити та відповіді. Однак Postman не обмежується лише ручним тестуванням, але також надає можливість автоматизації тестування API [27].

Ось деякі види тестів, які можна автоматизувати за допомогою Postman:

– Функціональне тестування API. До нього належать перевірка правильності відповідей на HTTP-запити; тестування різних методів HTTP; перевірка обробки помилок та валідація статус кодів.

– Тестування авторизації та аутентифікації. Наприклад, перевірка роботи механізмів авторизації (наприклад, OAuth); перевірка прав доступу для різних користувачів.

– Тестування навантажень. До нього належать створення колекцій запитів для тестування навантаження API та можливість встановлення та вимірювання максимального обсягу запитів.

Отож для проведення функціонального тестування для модулів локально потрібно виконати такі дії:

– розгорнути модуль локально;

– кинути HTTP-запит до ентпоіту;

– проаналізувати результат.

Розглянемо запити до ендпоінтів модуля авторизації, їх результати представлені на рисунках 4.6-4.12.

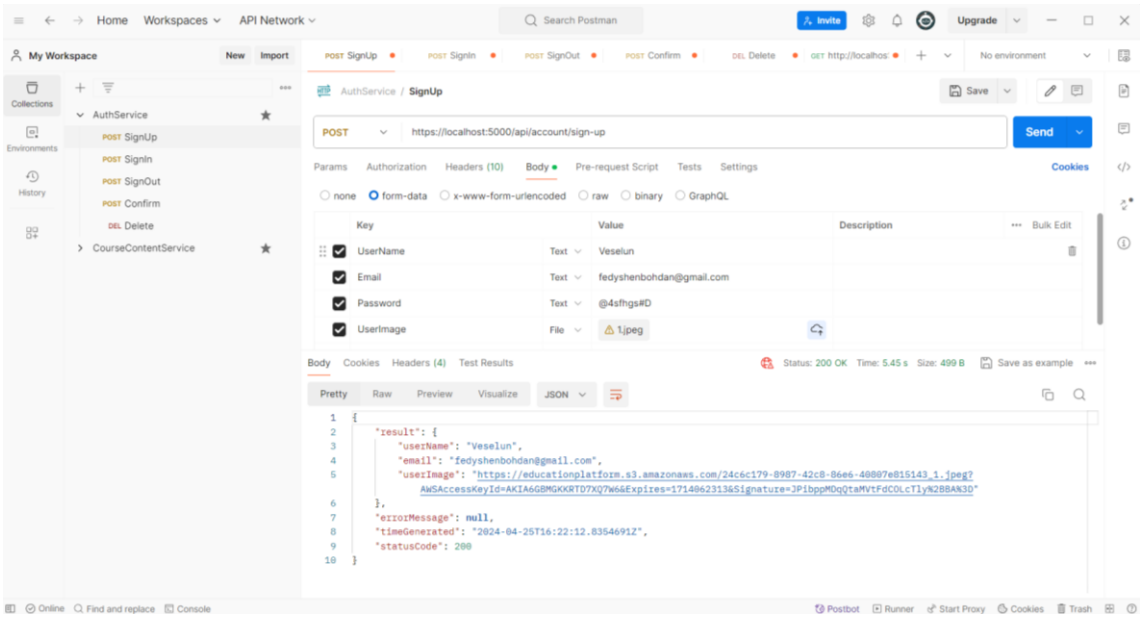

Рисунок 4.6 – Результат запиту до методу SignUp

Тут виконується POST-запит на сервер, для реєстрації користувача. Передається JSON-об’єкт(ім’я, електронна адреса, пароль, зображення), як тіло запиту. В результаті отримуємо повідомлення про успішну реєстрацію та дані користувача.

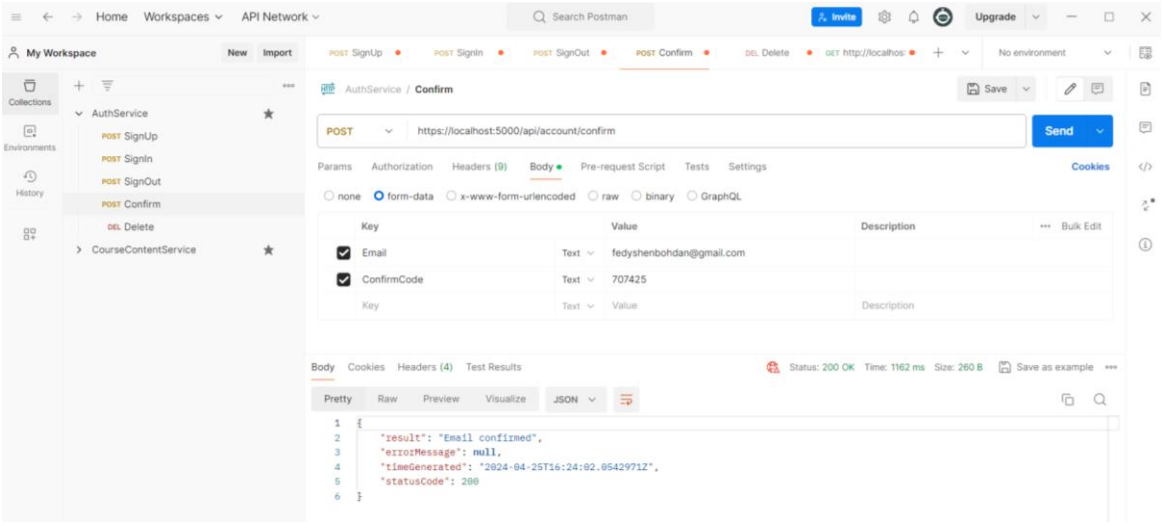

Рисунок 4.7 – Результат запиту до методу Confirm

Тут виконується POST-запит на сервер, для підтвердження пошти користувача. Передається JSON-об’єкт(електронна адреса та код

підтвердження), як тіло запиту. В результаті отримуємо повідомлення про успішне підтвердження електронної пошти.

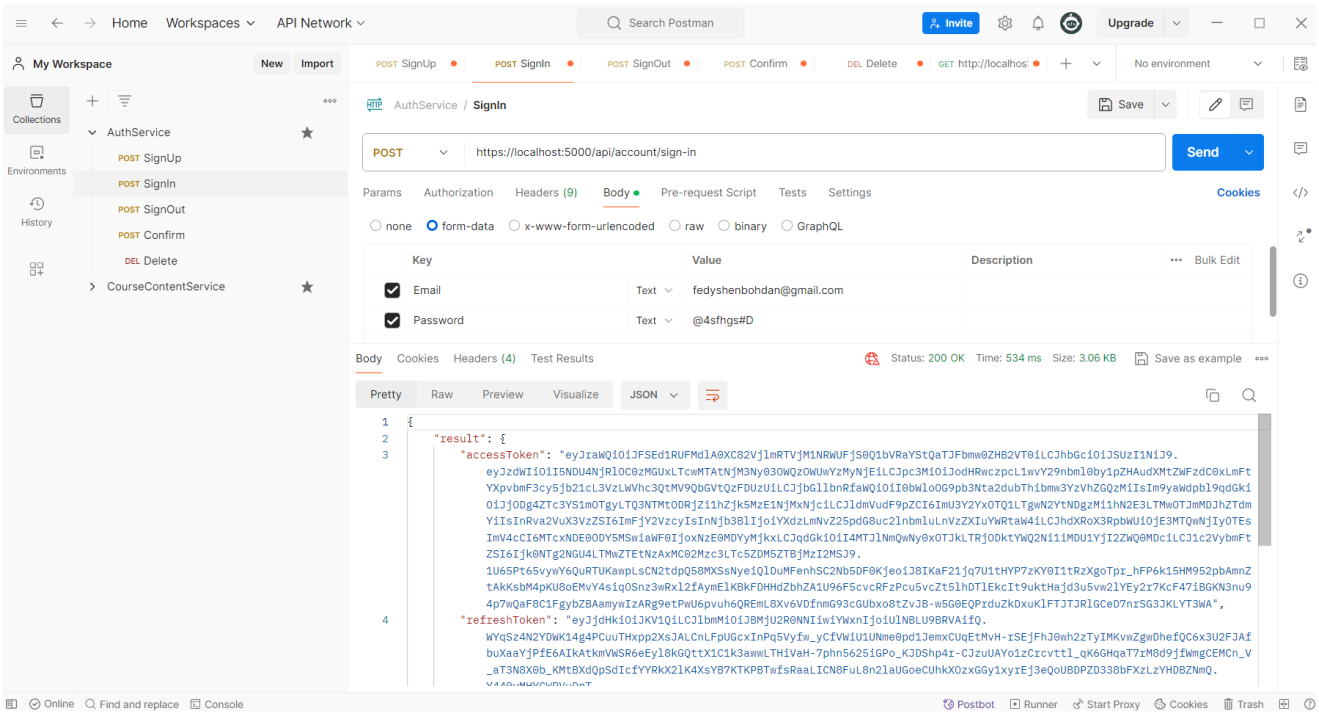

Рисунок 4.8 – Результат запиту до методу SignIn

Тут виконується POST-запит на сервер, для ідентифікації користувача та отримання токенів авторизації. Передається JSON-об’єкт(електронна адреса та пароль), як тіло запиту. В результаті отримуємо повідомлення про успішну авторизацію та токени користувача.

Всі інші методи тестуються подібним чином, тому розглянемо ще декілька прикладів різних контролерах. Варто зауважити, що проект містить 35 ендпоінтів, тому демонструвати тестування кожного немає сенсу. Загалом використано такі HTTP-методи як: GET, POST, PUT, DELETE.

Важливим нюансом тестування деяких методів, є обов’язкова наявність токену в заголовку запиту (рис. 4.9-4.12). Наприклад всі ендпоінти модуля адміністрування контенту позначені атрибутом Authorize [28].

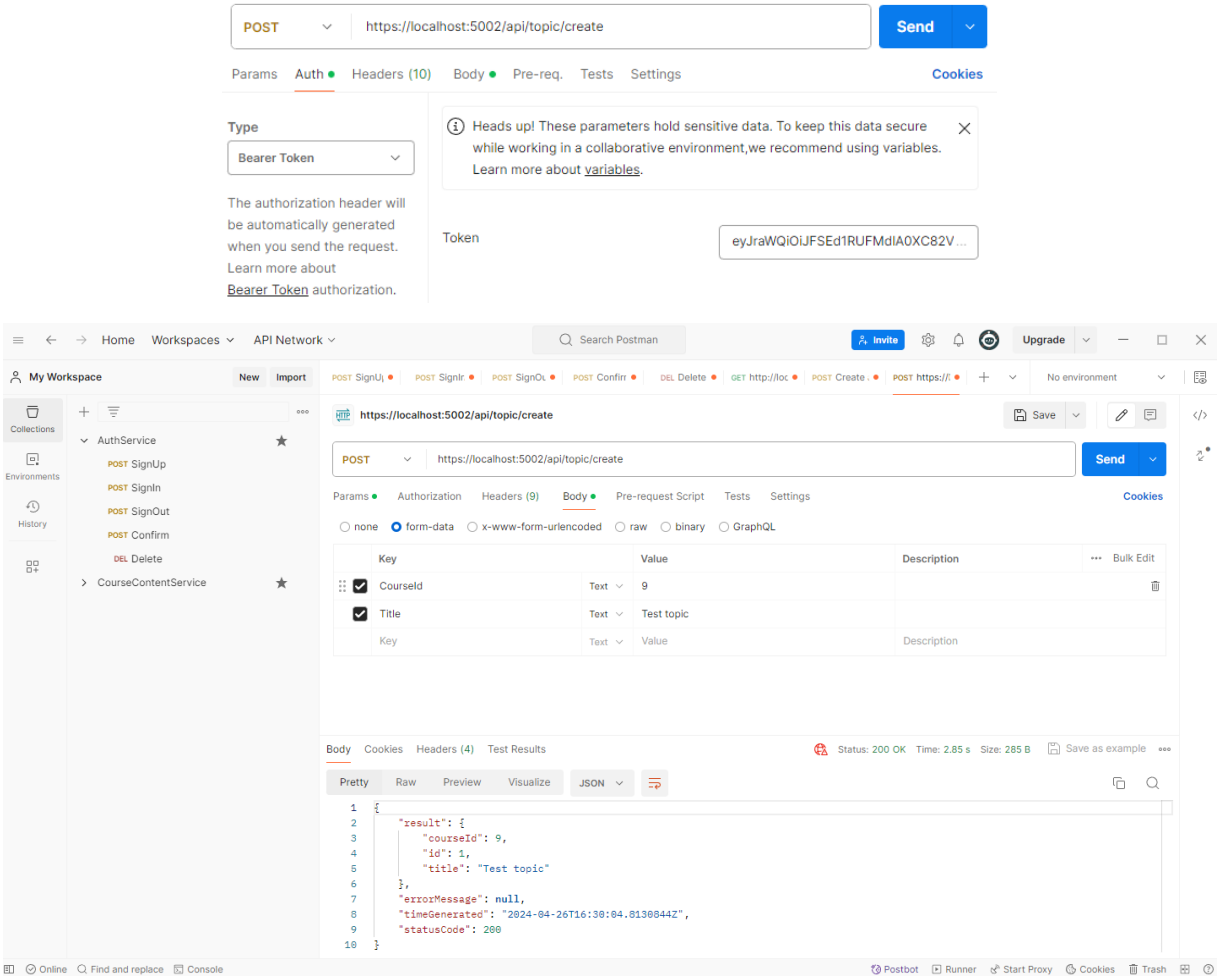

Рисунок 4.9 – Результат запиту до методу CreateTopic з використанням токену

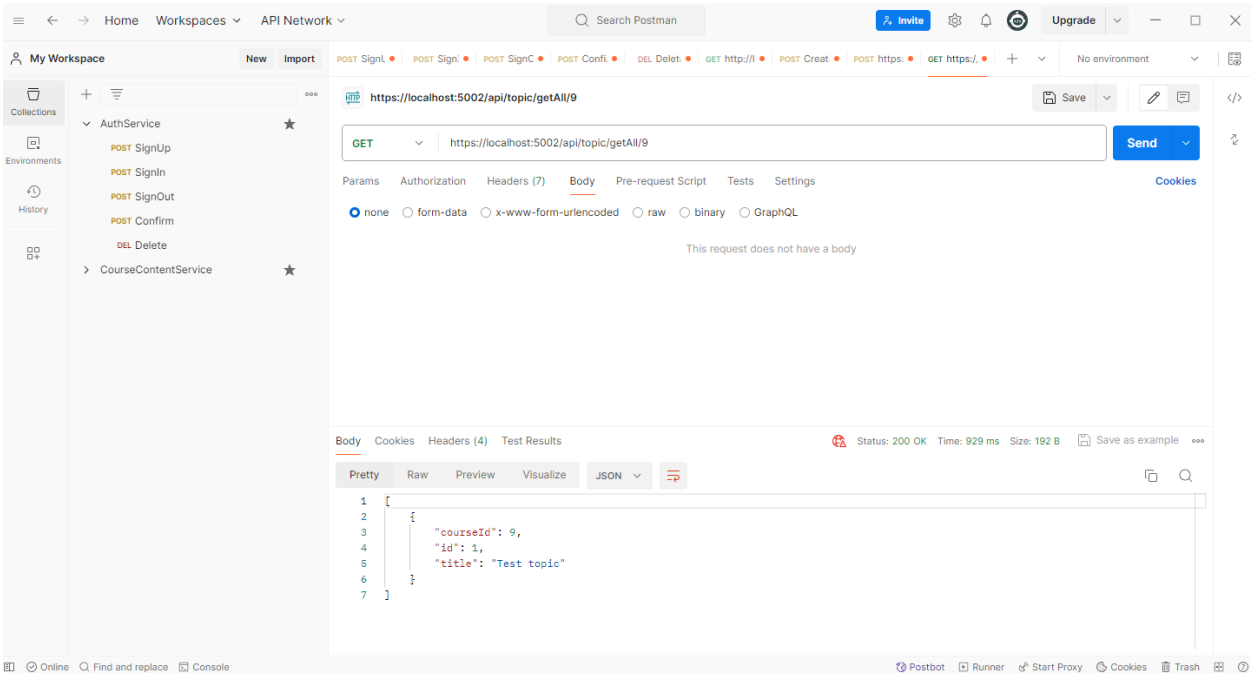

Рисунок 4.10 – Результат запиту до методу GetAllTopic

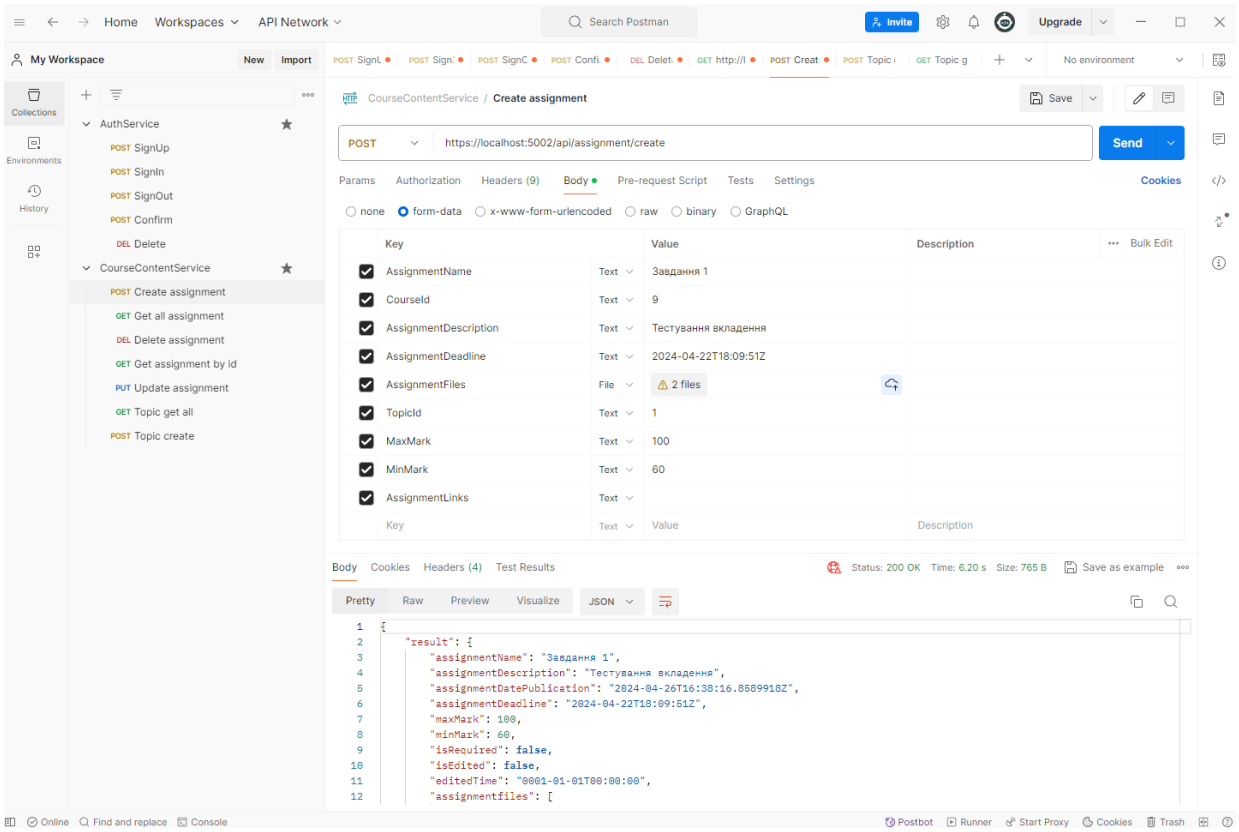

Рисунок 4.11 – Результат запиту до методу CreateAssignment

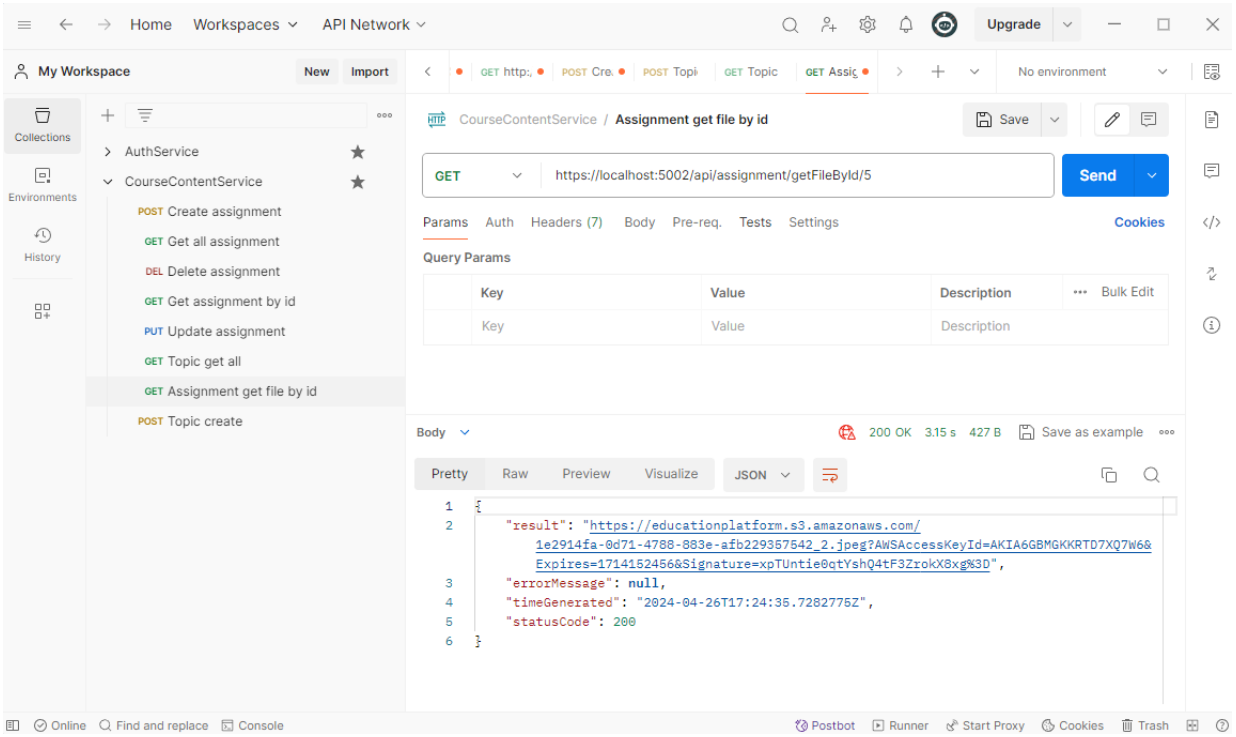

Рисунок 4.12 – Результат запиту до методу GetAssignmentFileById

На цьому етапі функціональне тестування можна вважати завершеним.

4.5 Висновок до розділу 4

У цьому розділі обрано декілька способів тестування розроблених програмних модулів.

Отже обрано модульне, інтеграційне та функціональне тестування для перевірки якості розробленого коду.

Проведено модульне тестування сервісів модуля авторизації, з використанням сучасних підходів

Проведено інтеграційне тестування певних розроблених модулів з використанням інструментів LocalStack та Testcontainers.

Виконано функціональне тестування ендпоінтів, проведена демонстрація основного підходу цього тестування з використанням інструменту Postman.

На цьому етапі цикл розробки програмного застосунку можна вважати завершеним.

ВИСНОВКИ

В ході виконання бакалаврської дипломної роботи виконано усі необхідні завдання:

– проаналізовано платформи-аналоги, визначено їх набір функцій та недоліки;

– визначено пріоритетні завдання та можливі покращення;

– обрано інструменти реалізації для бізнес-логіки платформи;

– реалізовано три модулі, для забезпечення потрібного набору функцій;

– протестовано роботу кожного модуля за допомогою декількох видів тестування

Для успішного виконання поставленого завдання було проаналізовано предметну область, визначено актуальність та доцільність розробки, розглянуто аналоги, для визначення недоліків існуючих платформ. На основі отриманих даних сформовано мету подальшого дослідження та доведено доцільність розробки.

Виконано проектування системи – визначено архітектуру, сформовано домен предметної області за принципами предметно-орієнтованого проектування, обрано варіанти взаємодії компонентів платформи, наведено алгоритм роботи одного з варіантів взаємодії, розроблено загальну структуру модулів бізнес-логіки.

Розглянуто інструменти для реалізації програмного коду. Визначено їх сферу застосування, розглянуто набір функцій. Було обрано мову C# для виконання завдання з огляду на її особливості та широкий набір інструментів для реалізації бізнес-логіки, а саме: програмний каркас ASP .NET, що призначений для розробки веб-застосунків, бібліотека Entity Framework Core, що призначена для взаємодії із сховищем даних та деякі інші інструменти.

Продемонстровано основні етапи реалізації програмного коду заданих модулів платформи. Описано особливості структури, взаємодії різних рівнів

кожного модуля, наведено діаграми класів, описано основні підходи та застосування різних шаблонів проектування й архітектурних шаблонів.

Протягом виконання дипломної роботи було розроблено три модулі освітньої платформи, які були протестовані. Для тестування розроблених модулів було проведено модульне, інтеграційне та функціональне тестування. Серед інструментів для проведення тестування було обрано бібліотеку xUnit для написання модульних та інтеграційних тестів, використано Docker для розгортання тестових контейнерів з використанням інструментів Testcontainers та LocalStack. Функціональне тестування проводилося за допомогою інструмента Postman.

В результаті виконання дипломної роботи розроблено та протестовано три модулі для реалізації мікросервісного веб-застосунку освітньої платформи. Розроблені модулі відповідають заданим вимогам, покращення комунікації на освітній платформі реалізовано за допомогою чату в реальному часі. Всі поставленні завдання були виконані в повному обсязі, а пояснювальна записка оформлена згідно методичних вказівок до виконання бакалаврської дипломної роботи [29].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Данило Соболєв, Богдан Федишен, Ігор Денисов. “Сучасні підходи при розробці архітектури програмного забезпечення” в Матеріалах конференції “LIII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)”, Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/(дата звернення: 10.05.2024).
2. Освітня платформа JetIQ. URL: https://jetiq.vntu.edu.ua/ (дата звернення: 09.05.2024).
3. Освітня платформа Classroom. URL: https://classroom.google.com/ (дата звернення: 10.05.2024).
4. Мікросервісна архітектура. URL: https://cloud.google.com/learn/what-is-microservices-architecture (дата звернення: 13.05.2024).
5. Clean Architecture .NET Core URL: https://positiwise.com/blog/clean-architecture-net-core (дата звернення: 13.05.2024).
6. REST API як спосіб спілкування компонент веб-додатків. URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 13.05.2024).
7. The WebSocket API and protocol explained. URL: https://ably.com/topic/websockets (дата звернення: 14.05.2024).
8. Domain first. Як DDD полегшує розробку у великих командах додатків. URL: https://www.gen.tech/post/sho-take-ddd (дата звернення: 15.05.2024).
9. C# language documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 16.05.2024).
10. JavaScript language documentation. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 16.05.2024).
11. Python language documentation. URL: https://www.python.org/ (дата звернення: 16.05.2024).
12. AWS. URL: https://aws.amazon.com/ (дата звернення: 17.05.2024).

13. Amazon Cognito Documentation. URL: https://docs.aws.amazon.com/cognito/ (дата звернення: 18.05.2024).
14. Create Data Transfer Objects (DTOs). URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 (дата звернення: 18.05.2024).
15. Mastering Dependency Injection in C#: Best Practices, Pitfalls, and Future Trends. URL: https://artemasemenov.medium.com/mastering-dependency-injection-in-c-best-practices (дата звернення: 18.05.2024).
16. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started (дата звернення: 20.05.2024).
17. Repository Pattern – A controversy explained. URL: https://steven-giesel.com/blogPost/ (дата звернення: 20.05.2024).
18. Принципи SOLID – що це? URL: https://www.gen.tech/post/principi-solid (дата звернення: 20.05.2024).
19. Result Pattern. URL: https://medium.com/@wgyxxbf/result-pattern-a01729f42f8c (дата звернення: 20.05.2024).
20. Options pattern in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options (дата звернення: 21.05.2024).
21. AWS Systems Manager. URL: https://aws.amazon.com/systems-manager/ (дата звернення: 22.05.2024).
22. Overview of ASP.NET Core SignalR. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction (дата звернення: 23.05.2024).
23. About xUnit.NET. URL: https://xunit.net/ (дата звернення: 24.05.2024).
24. Unit tests with real dependencies. URL: https://testcontainers.com/ (дата звернення: 27.05.2024)
25. Docker. URL: https://docs.docker.com/ (дата звернення: 27.05.2024)
26. LocalStack Documentation. URL: https://docs.localstack.cloud/overview/ (дата звернення: 27.05.2024)

27. What is Postman? URL: https://www.postman.com/product/what-is-postman/ (дата звернення: 28.05.2024)
28. Simple authorization in ASP.NET Core. URL: https://shorturl.at/fjtLW (дата звернення: 28.05.2024)
29. Методичні вказівки до виконання бакалаврської дипломної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп`ютерні науки». URL: https://iq.vntu.edu.ua/method/ (дата звернення: 03.06.2024)

ДОДАТКИ

ДОДАТОК А. ПРОТОКОЛ ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мікросервісного WEB-додатку освітньої платформи. Частина 1. Мікросервіси авторизації, адміністрування наповнення курсів та чату в реальному часі.

Тип роботи: бакалаврська дипломна робота

(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА

(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,95 % Схожість 2,05 %

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Федишен Б. В.

Керівник роботи Денисов І. К.

ДОДАТОК Б. ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```
using CourseContent.Domain.Entities;
namespace CourseContent.Core.DTO.Requests.UpdateDTO
{
   public class AssignmentUpdateDTO
   {
      public int Id { get; set; }
      public int CourseId { get; set; }
      public int? TopicId { get; set; }
      public required string AssignmentName { get; set; }
      public string? AssignmentDescription { get; set; }
      public int MaxMark { get; set; }
      public int MinMark { get; set; }
      public bool IsRequired { get; set; }
      public DateTime AssignmentDatePublication { get; set; } = DateTime.UtcNow;
      public DateTime AssignmentDeadline { get; set; }
      public static Assignment FromAssignmentUpdateDto(AssignmentUpdateDTO assignmentDto)
      {
         return new Assignment
         {
            Id = assignmentDto.Id,
            CourseId = assignmentDto.CourseId,
            TopicId = assignmentDto.TopicId,
            AssignmentName = assignmentDto.AssignmentName,
            AssignmentDescription = assignmentDto.AssignmentDescription,
            MaxMark = assignmentDto.MaxMark,
            MinMark = assignmentDto.MinMark,
            IsRequired = assignmentDto.IsRequired,
            AssignmentDatePublication = assignmentDto.AssignmentDatePublication,
            AssignmentDeadline = assignmentDto.AssignmentDeadline
         };
      }
   }
}

using CourseContent.Core.DTO.Requests.AssignmentDTO;
using CourseContent.Core.DTO.Requests.UpdateDTO;
using CourseContent.Core.DTO.Responses;
using CourseContent.Core.Interfaces;
using CourseContent.Core.Models.ErrorModels;
using CourseContent.Domain.Entities;
using CourseContent.Web.Controllers.Base;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
namespace CourseContent.Web.Controllers
{
   [Route("api/assignment")]
   [ApiController]
   public class AssignmentController(IOperation<AssignmentOutDTO, Error, AssignmentDTO,
```

```
    AssignmentfileOutDTO, AssignmentUpdateDTO, Assignmentlink> operation) :
BaseController
  {
    private readonly IOperation<AssignmentOutDTO, Error, AssignmentDTO,
      AssignmentfileOutDTO, AssignmentUpdateDTO, Assignmentlink> _operation = operation;
    [Authorize]
    [HttpPost("create")]
    public async Task<IActionResult> CreateAssignment([FromForm] AssignmentDTO
assignment)
    {
      var result = await _operation.CreateAsync(assignment);
      return FromResult(result);
    }
    [Authorize]
    [HttpPut("update")]
    public async Task<IActionResult> UpdateAssignment([FromForm] AssignmentUpdateDTO
assignment)
    {
      var result = await _operation.UpdateAsync(assignment, assignment.Id);
      return FromResult(result);
    }
    [Authorize]
    [HttpDelete("delete/{id}")]
    public async Task<IActionResult> DeleteAssignment(int id)
    {
      var result = await _operation.DeleteAsync(id);
      return FromResult(result);
    }
    [Authorize]
    [HttpGet("getById/{id}")]
    public async Task<IActionResult> GetByIdAssignment(int id)
    {
      var result = await _operation.GetByIdAsync(id);
      return FromResult(result);
    }
    [Authorize]
    [HttpGet("getAll/{id}")]
    public async Task<IEnumerable<AssignmentOutDTO>> GetAllAssignment(int id)
    {
      return await _operation.GetAllByCourseAsync(id);
    }
    [Authorize]
    [HttpDelete("removeList")]
    public async Task<IActionResult> RemoveAssignments([FromBody] List<int> entities)
    {
      var result = await _operation.RemoveRangeAsync(entities);
      return FromResult(result);
    }
    [Authorize]
    [HttpGet("getFileById/{fileId}")]
    public async Task<IActionResult> GetAssignmentFileById(int fileId)
    {
```

```
            var result = await _operation.GetFileByIdAsync(fileId);
            return FromResult(result);
        }
        [Authorize]
        [HttpDelete("deleteFileById/{fileId}")]
        public async Task<IActionResult> DeleteAssignmentFileById(int fileId)
        {
            var result = await _operation.DeleteFileAsync(fileId);
            return FromResult(result);
        }
        [Authorize]
        [HttpPost("addFile/{id}")]
        public async Task<IActionResult> AddAssignmentFile([FromForm] IFormFile file, int id)
        {
            var result = await _operation.AddFileAsync(file, id);
            return FromResult(result);
        }
        [Authorize]
        [HttpPost("addLink/{id}")]
        public async Task<IActionResult> AddAssignmentLink([FromBody] string link, int id)
        {
            var result = await _operation.AddLinkAsync(link, id);
            return FromResult(result);
        }
        [Authorize]
        [HttpDelete("deleteLinkById/{linkId}")]
        public async Task<IActionResult> DeleteAssignmentLinkById(int linkId)
        {
            var result = await _operation.DeleteLinkAsync(linkId);
            return FromResult(result);
        }
    }
}

using CourseContent.Core.DTO.Requests.AssignmentDTO;
using CourseContent.Core.DTO.Requests.UpdateDTO;
using CourseContent.Core.DTO.Responses;
using CourseContent.Core.Helpers;
using CourseContent.Core.Interfaces;
using CourseContent.Core.Models.ErrorModels;
using CourseContent.Domain.Entities;
using CourseContent.Infrastructure.Interfaces;
using CSharpFunctionalExtensions;
using Microsoft.AspNetCore.Http;
namespace CourseContent.Core.Services
{
    public class AssignmentService(IUnitOfWork unitOfWork, FileHelper fileHelper) :
        IOperation<AssignmentOutDTO, Error, AssignmentDTO, AssignmentfileOutDTO,
AssignmentUpdateDTO, Assignmentlink>
    {
        private readonly IUnitOfWork _unitOfWork = unitOfWork;
        private readonly FileHelper _fileHelper = fileHelper;
```

```
    public async Task<Result<AssignmentOutDTO, Error>> CreateAsync(AssignmentDTO
entity)
    {
        var assignment = AssignmentDTO.FromAssignmentDto(entity);
        await _unitOfWork.AssignmentRepository.AddAsync(assignment);
        await _unitOfWork.CommitAsync();
        if (entity.AssignmentFiles is not null)
        {
            await AddFilesAsync(assignment, entity.AssignmentFiles);
        }
        if (entity.AssignmentLinks is not null)
        {
            await AddLinksAsync(assignment, entity.AssignmentLinks);
        }
        return Result.Success<AssignmentOutDTO,
Error>(AssignmentOutDTO.FromAssignment(assignment));
    }
    private async Task AddFilesAsync(Assignment entity, List<IFormFile> files)
    {
        foreach (var file in files)
        {
            var fileLink = await _fileHelper.AddFileAsync(file);
            _unitOfWork.AssignmentRepository.AddFile(entity, fileLink);
        }
        await _unitOfWork.CommitAsync();
    }
    private async Task AddLinksAsync(Assignment entity, List<string> links)
    {
        foreach (var link in links)
        {
            _unitOfWork.AssignmentRepository.AddLink(entity, link);
        }
        await _unitOfWork.CommitAsync();
    }
    public async Task<Result<AssignmentOutDTO, Error>>
UpdateAsync(AssignmentUpdateDTO entity, int id)
    {
        try
        {
            var assignment = AssignmentUpdateDTO.FromAssignmentUpdateDto(entity);
            assignment.IsEdited = true;
            assignment.EditedTime = DateTime.UtcNow;
            await _unitOfWork.AssignmentRepository.UpdateAsync(id, assignment);
            await _unitOfWork.CommitAsync();
            var updatedAssignment = await _unitOfWork.AssignmentRepository.GetByIdAsync(id, a
=> a.Assignmentfiles, a => a.Assignmentlinks);
            return Result.Success<AssignmentOutDTO,
Error>(AssignmentOutDTO.FromAssignment(updatedAssignment!));
        }
        catch (KeyNotFoundException)
        {
            return Result.Failure<AssignmentOutDTO, Error>(Errors.General.NotFound());
```

```
    }
}
public async Task<Result<string, Error>> DeleteAsync(int id)
{
    try
    {
        await _unitOfWork.AssignmentRepository.DeleteAsync(id);
        await _unitOfWork.CommitAsync();
        return Result.Success<string, Error>("Deleted was successful");
    }
    catch (KeyNotFoundException)
    {
        return Result.Failure<string, Error>(Errors.General.NotFound());
    }
}
public async Task<Result<string, Error>> RemoveRangeAsync(List<int> entities)
{
    if (entities.Count == 0)
    {
        return Result.Failure<string, Error>(Errors.General.NotRecords());
    }
    await _unitOfWork.AssignmentRepository.RemoveRange(entities);
    await _unitOfWork.CommitAsync();
    return Result.Success<string, Error>("Deleted was successful");
}
public async Task<Result<string, Error>> DeleteFileAsync(int id)
{
    try
    {
        var assignmentFile = await _unitOfWork.AssignmentfileRepository.GetByIdAsync(id);
        if (assignmentFile is not null && assignmentFile.AssignmentFile is not null)
        {
            await _fileHelper.DeleteFileAsync(assignmentFile.AssignmentFile);
        }
        await _unitOfWork.AssignmentfileRepository.DeleteAsync(id);
        await _unitOfWork.CommitAsync();
        return Result.Success<string, Error>("Deleted was successful");
    }
    catch (KeyNotFoundException)
    {
        return Result.Failure<string, Error>(Errors.General.NotFound());
    }
}
public async Task<Result<string, Error>> DeleteLinkAsync(int linkId)
{
    try
    {
        await _unitOfWork.AssignmentlinkRepository.DeleteAsync(linkId);
        await _unitOfWork.CommitAsync();
        return Result.Success<string, Error>("Deleted was successful");
    }
    catch (KeyNotFoundException)
```

```
        {
            return Result.Failure<string, Error>(Errors.General.NotFound());
        }
    }
    public async Task<Result<AssignmentfileOutDTO, Error>> AddFileAsync(IFormFile file, int
id)
    {
        var fileLink = await _fileHelper.AddFileAsync(file);
        Assignmentfile assignmentFile = new ()
        {
            AssignmentId = id,
            AssignmentFile = fileLink
        };
        var addedFile = await _unitOfWork.AssignmentfileRepository.AddAsync(assignmentFile);
        await _unitOfWork.CommitAsync();
        return Result.Success<AssignmentfileOutDTO,
Error>(AssignmentfileOutDTO.FromAssignmentFile(addedFile));
    }
    public async Task<Result<Assignmentlink, Error>> AddLinkAsync(string link, int id)
    {
        Assignmentlink assignmentLink = new()
        {
            AssignmentId = id,
            AssignmentLink = link
        };
        var addedLink = await
_unitOfWork.AssignmentlinkRepository.AddAsync(assignmentLink);
        await _unitOfWork.CommitAsync();
        if (addedLink.AssignmentLink is not null)
        {
            return Result.Success<Assignmentlink, Error>(addedLink);
        }
        else
        {
            return Result.Failure<Assignmentlink, Error>(Errors.General.NotAdded());
        }
    }
    public async Task<Result<AssignmentOutDTO, Error>> GetByIdAsync(int id)
    {
        var entity = await _unitOfWork.AssignmentRepository.GetByIdAsync(id, a =>
a.Assignmentfiles, a => a.Assignmentlinks);
        if (entity is null)
        {
            return Result.Failure<AssignmentOutDTO, Error>(Errors.General.NotFound());
        }
        return Result.Success<AssignmentOutDTO,
Error>(AssignmentOutDTO.FromAssignment(entity));
    }
    public async Task<Result<string, Error>> GetFileByIdAsync(int id)
    {
        var assignmentFile = await _unitOfWork.AssignmentfileRepository.GetByIdAsync(id);
        if (assignmentFile is null || assignmentFile.AssignmentFile is null)
```

```
            {
                return Result.Failure<string, Error>(Errors.General.NotFound());
            }
            return Result.Success<string, Error>(await _fileHelper
                .GetFileLink(assignmentFile.AssignmentFile));
        }
        public async Task<IEnumerable<AssignmentOutDTO>> GetAllByCourseAsync(int id)
        {
            var assignments = await _unitOfWork.AssignmentRepository
                .GetAllByCourseAsync(m => m.CourseId == id);
            return assignments.Select(AssignmentOutDTO.FromAssignment).ToList();
        }
    }
}

using CourseContent.Domain.Entities;
using CourseContent.Infrastructure.Interfaces;
using System.Linq.Expressions;
namespace CourseContent.Infrastructure.Repositories
{
    public class AssignmentRepository(IContentRepository<Assignment> contentRepository) :
        IContentRepository<Assignment>{
        private readonly IContentRepository<Assignment> _contentRepository = contentRepository;
        public async Task<Assignment> AddAsync(Assignment entity){
            return await _contentRepository.AddAsync(entity);
        }
        public async Task<Assignment?> GetByIdAsync(int id, params
Expression<Func<Assignment, object>>[] includes){
            return await _contentRepository.GetByIdAsync(id, includes);
        }
        public async Task<Assignment?> UpdateAsync(int id, Assignment entity){
            return await _contentRepository.UpdateAsync(id, entity);
        }
        public async Task<IEnumerable<Assignment>>
GetAllByCourseAsync(Expression<Func<Assignment, bool>> filter)
        {
            return await _contentRepository.GetAllByCourseAsync(filter);
        }
        public async Task DeleteAsync(int id){
            await _contentRepository.DeleteAsync(id);
        }
        public async Task RemoveRange(List<int> entities){
            await _contentRepository.RemoveRange(entities);
        }
        public void AddFile(Assignment entity, string file){
            _contentRepository.AddFile(entity, file);
        }
        public void AddLink(Assignment entity, string link){
            _contentRepository.AddLink(entity, link);
        }
    }
}
```

ДОДАТОК В. ІНСТРУКЦІЯ КОРИСТУВАЧА

Для перегляду основних ендпоінтів кожного мікросервісу можна скористатися як оглядачем ендоінтів (рис. В.1) так і просто запустити конкретний модуль та отримати ці дані на сторінці swagger/index.html (рис. В.2).

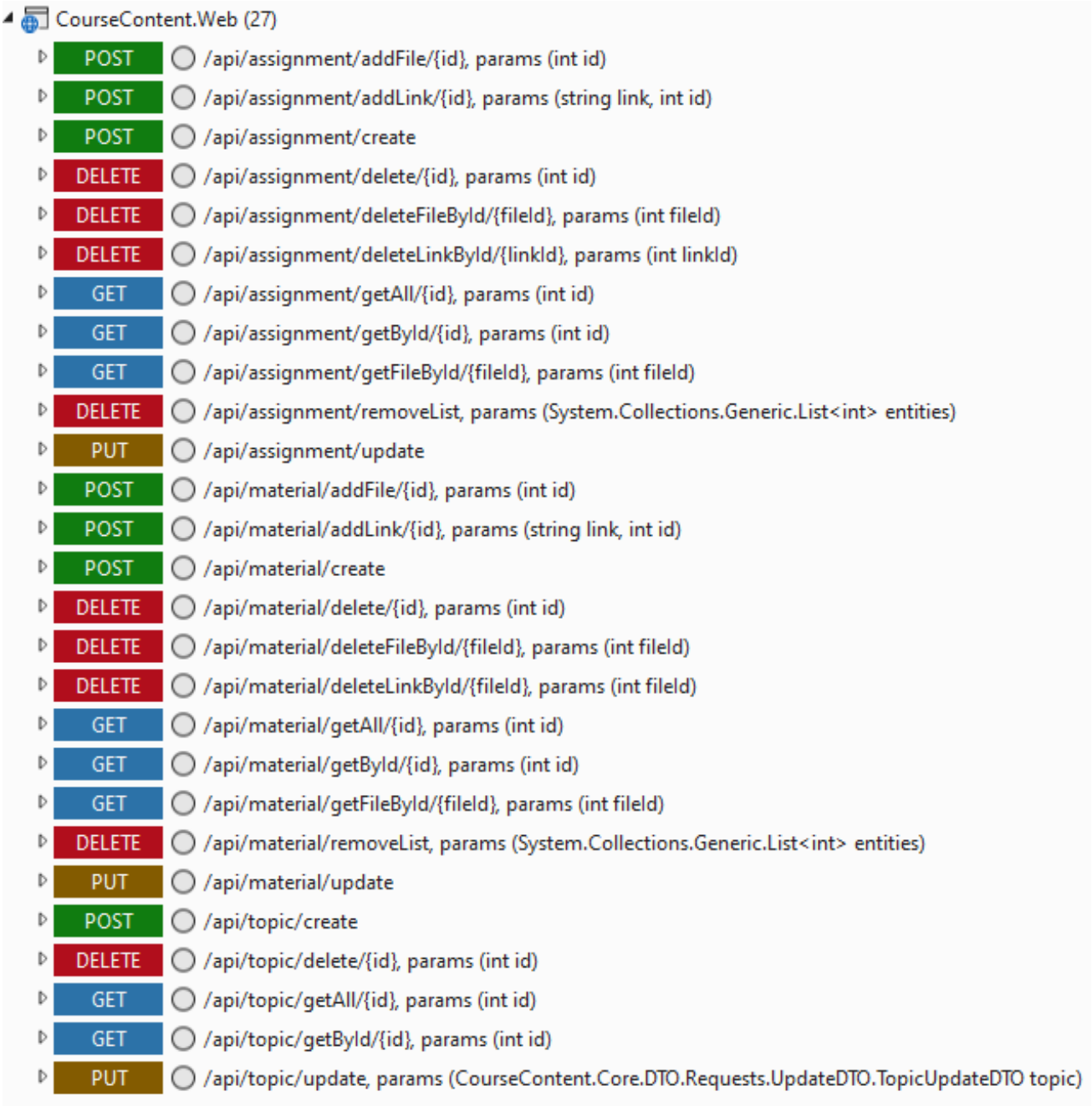

Рисунок В.1 – Оглядач ендпоінтів модуля адміністрування контенту

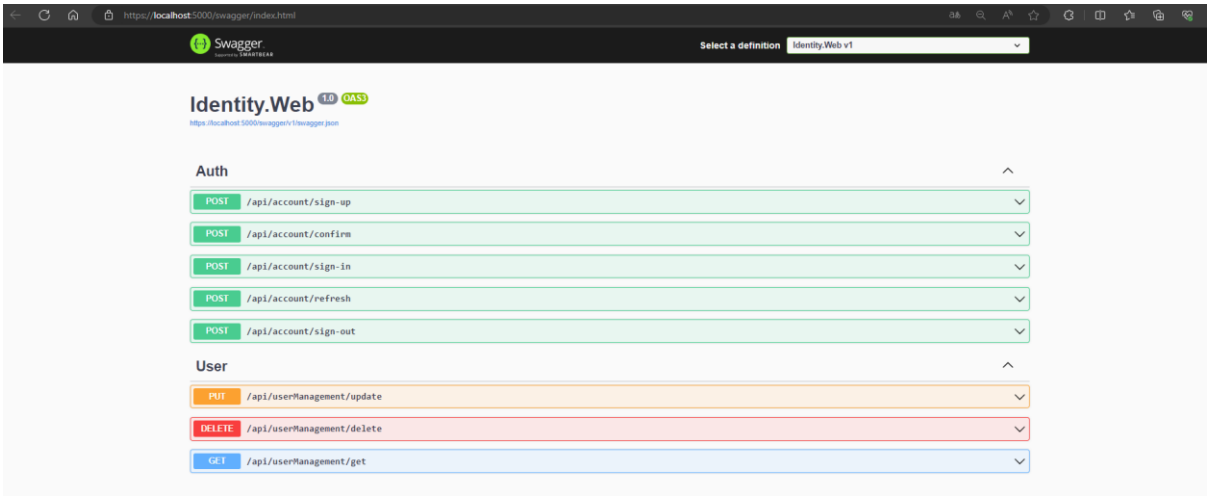

Рисунок В.2 – Вікно Swagger для модуля авторизації

Для того, щоб мати можливість зробити запити до API, необхідно зареєструватися та отримати токени доступу. Щоб отримати токен, після реєстрації потрібно звернутися до методу sign-in. Результат продемонстровано на рисунку В.3.

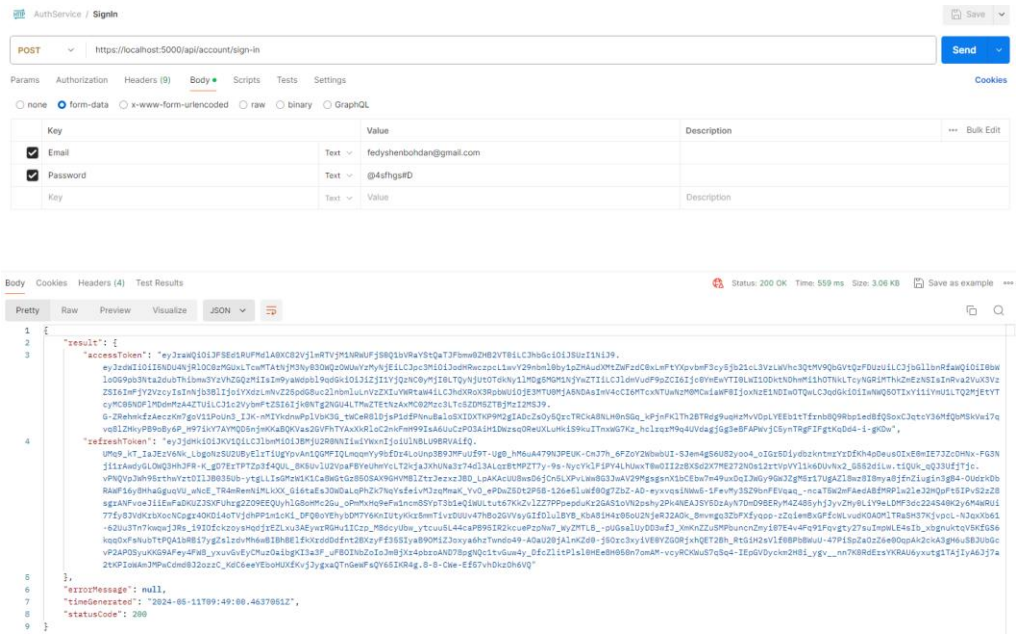

Рисунок В.3 – Отримані токени

Тепер для виконання запитів до модуля адміністрування контенту, необхідно взяти accessToken та додати його до заголовку. Для цього потрібно в середовищі Postman перейти на вкладку Authorization, вибрати Auth Type –

Bearer Token, та скопіювати вміст accessToken в поле Token. Результат цього етапу на рисунку В.4.

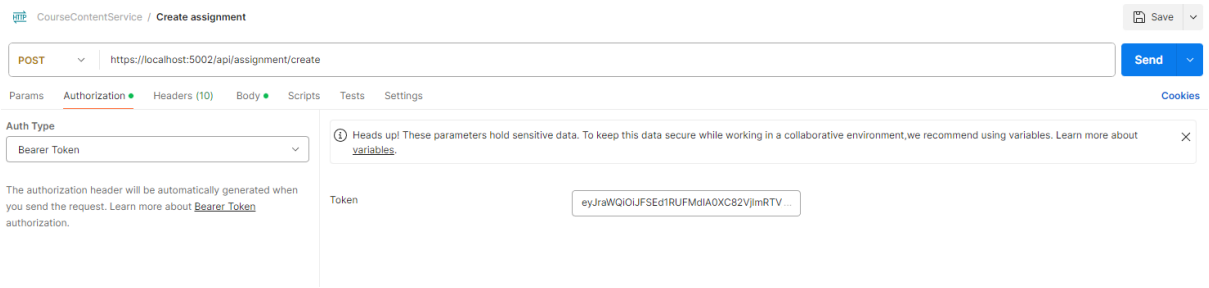

Рисунок В.4 – Додання accessToken до заголовку

Далі заповнюємо поля форми даними та надсилаємо на потрібну адресу. Заповнена форма та результат виконання наведені на рисунку В.5.

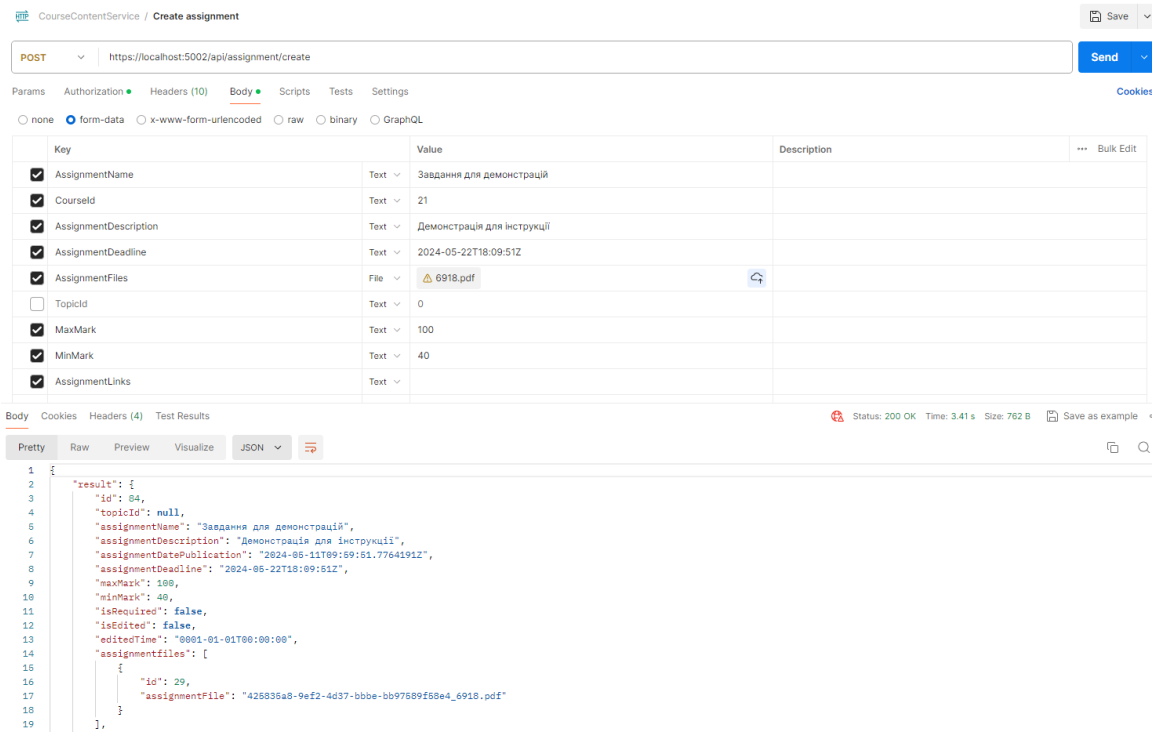

Рисунок В.5 – Результат виконання запиту із заголовком авторизації

Таким чином можна звернутися до кожного ендпоінту та отримати відповідні результат. Варто зауважити, що потрібно використовувати правильні HTTP-запити (POST, PUT, DELETE, GET), відповідно до оглядача ендпоінтів.

ДОДАТОК Г. ГРАФІЧНА ЧАСТИНА

РОЗРОБКА МІКРОСЕРВІСНОГО WEB-ДОДАТКУ ОСВІТНЬОЇ ПЛАТФОРМИ. ЧАСТИНА 1. МІКРОСЕРВІСИ АВТОРИЗАЦІЇ, АДМІНІСТРУВАННЯ НАПОВНЕННЯ КУРСІВ ТА ЧАТУ В РЕАЛЬНОМУ ЧАСІ

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Федишен Б. В
(прізвище та ініціали)
Керівник: асистент каф. КН
Денисов І. К.
(прізвище та ініціали)

«_____» _____ 2024 р.

Вінниця ВНТУ – 2024 рік

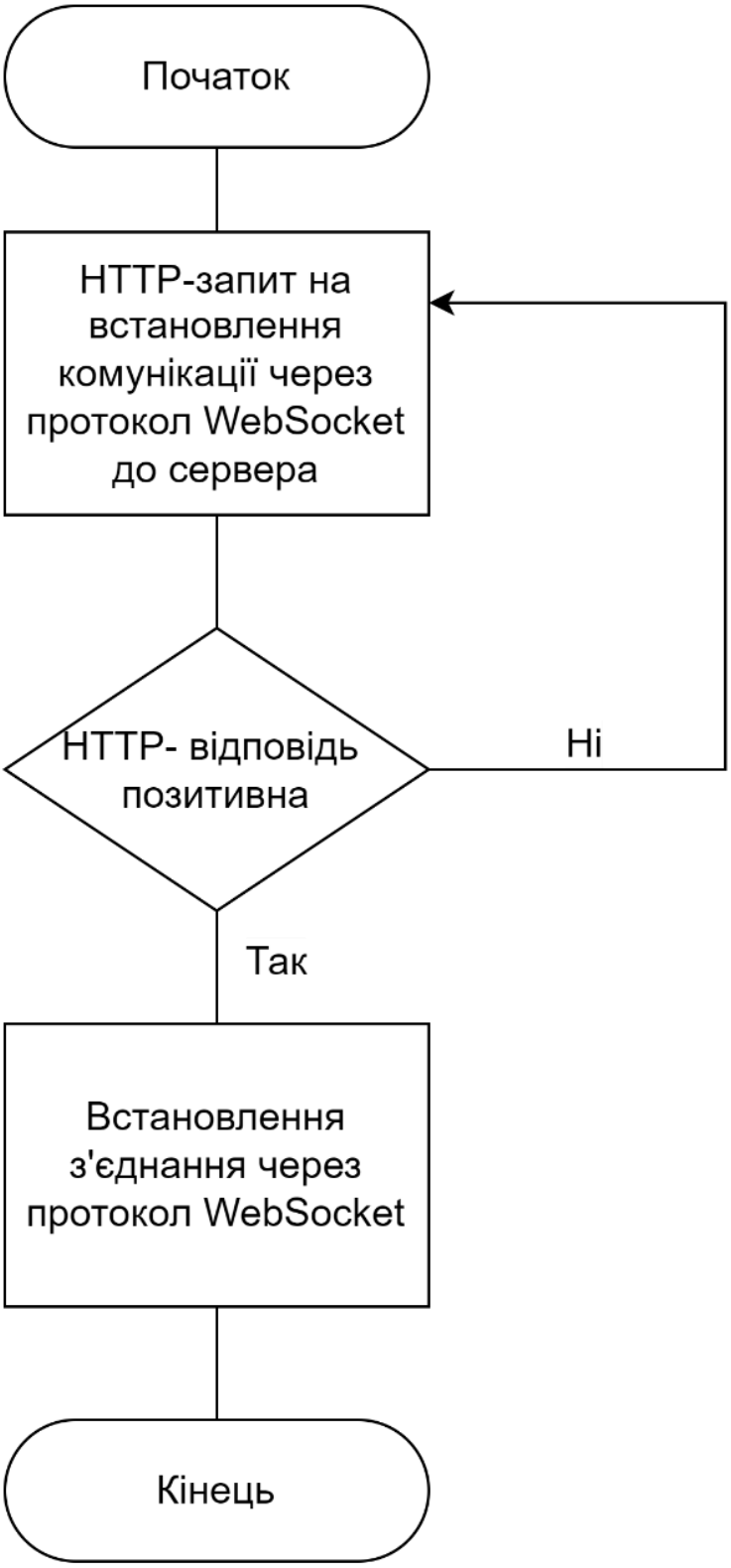

Рисунок Г.1 – Алгоритм роботи протоколу WebSocket

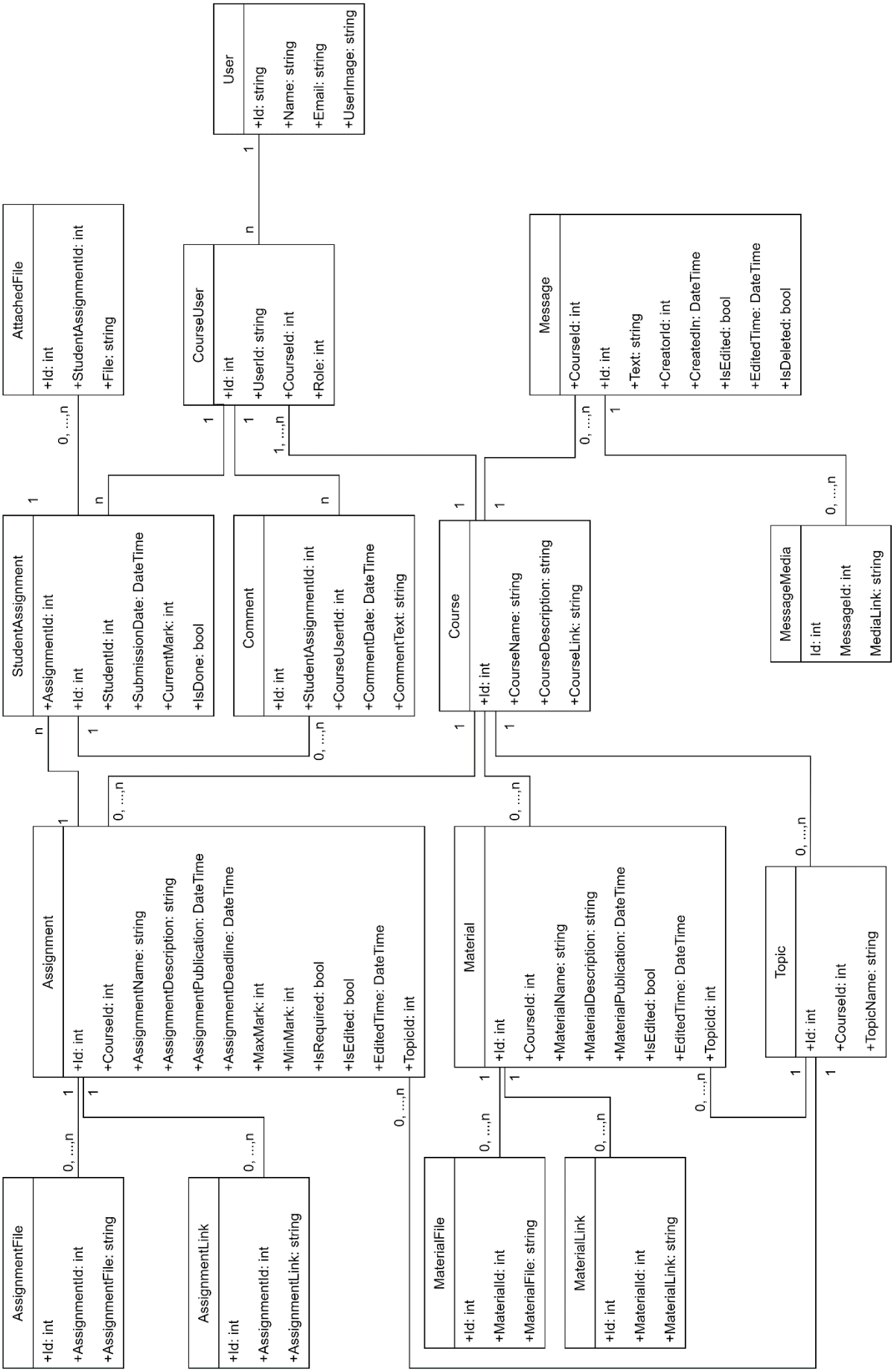

Рисунок Г.2 – Діаграма класів, що представляє домен

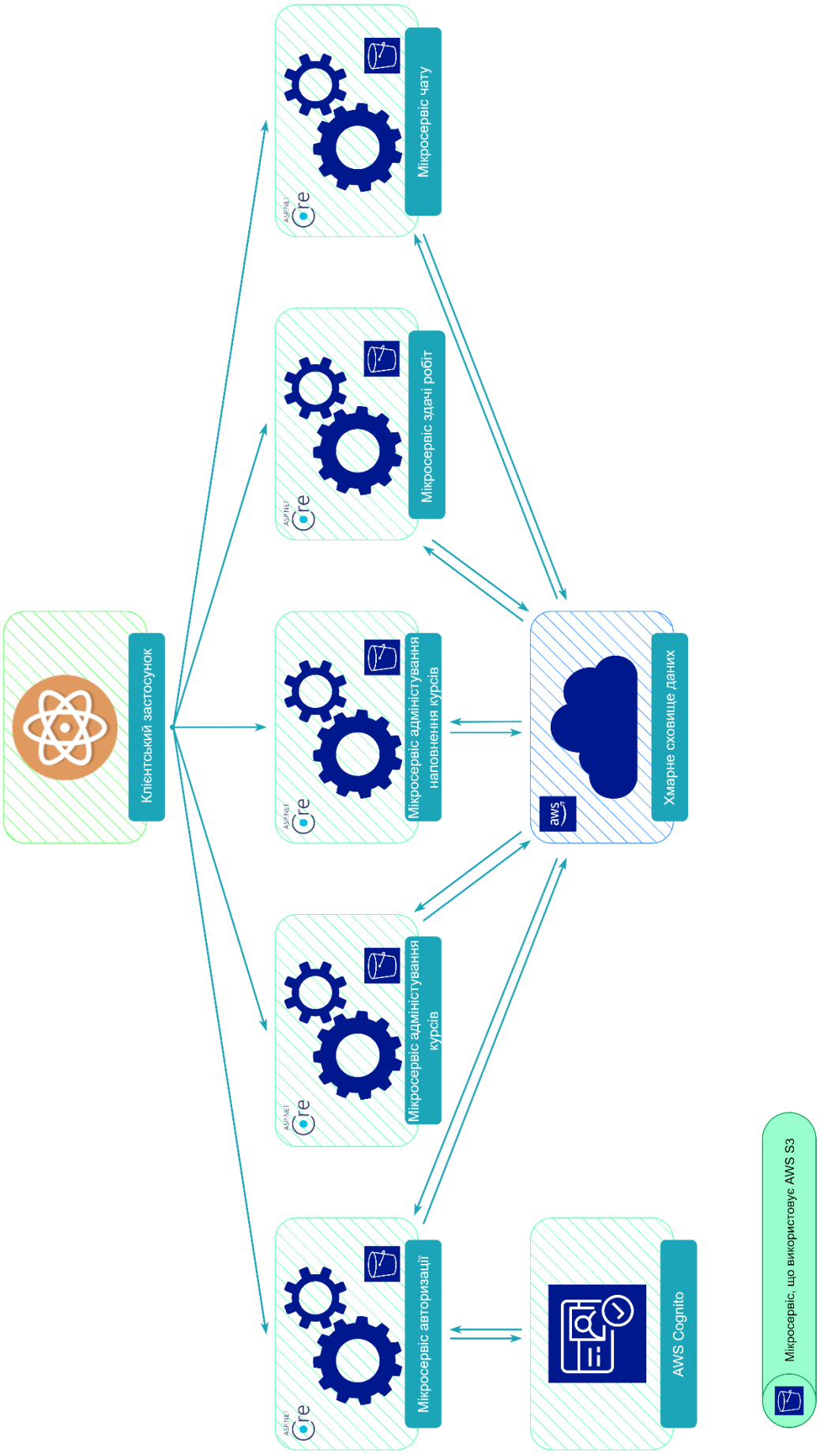

Рисунок Г.3 – Взаємодія усіх компонентів системи

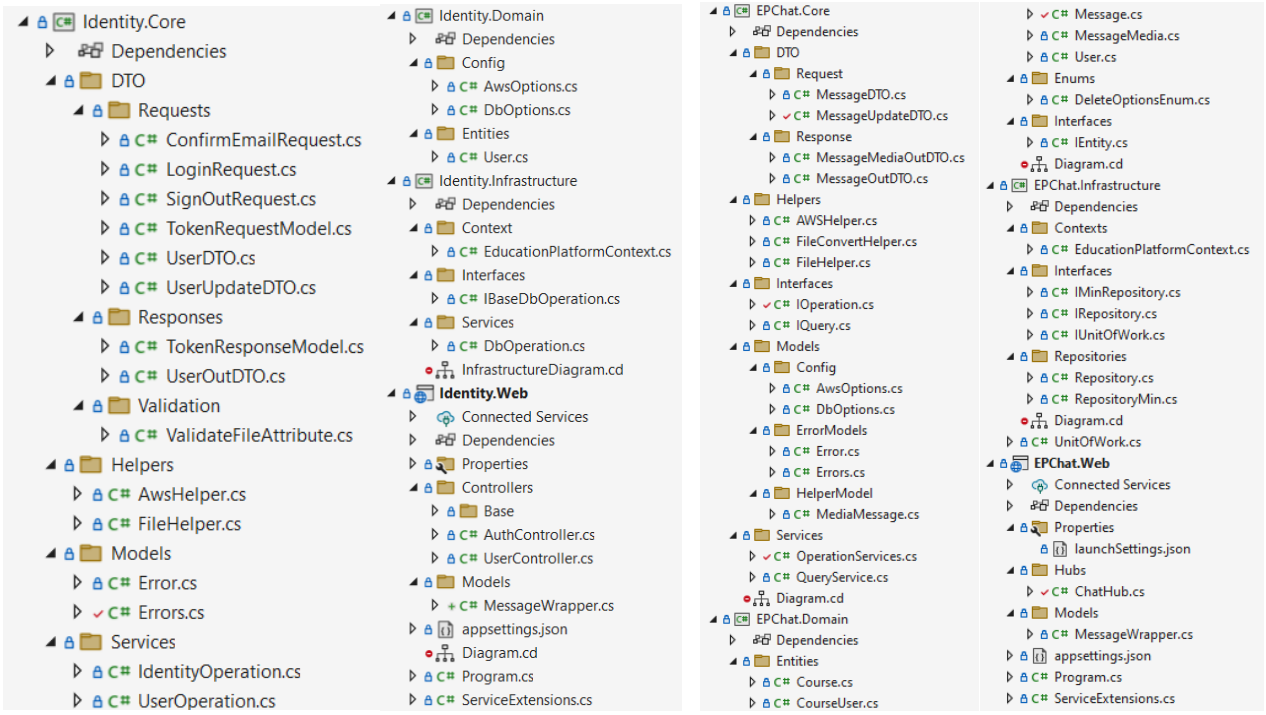

Рисунок Г.4 – Файлова структура модулів авторизації та чату в реальному часі

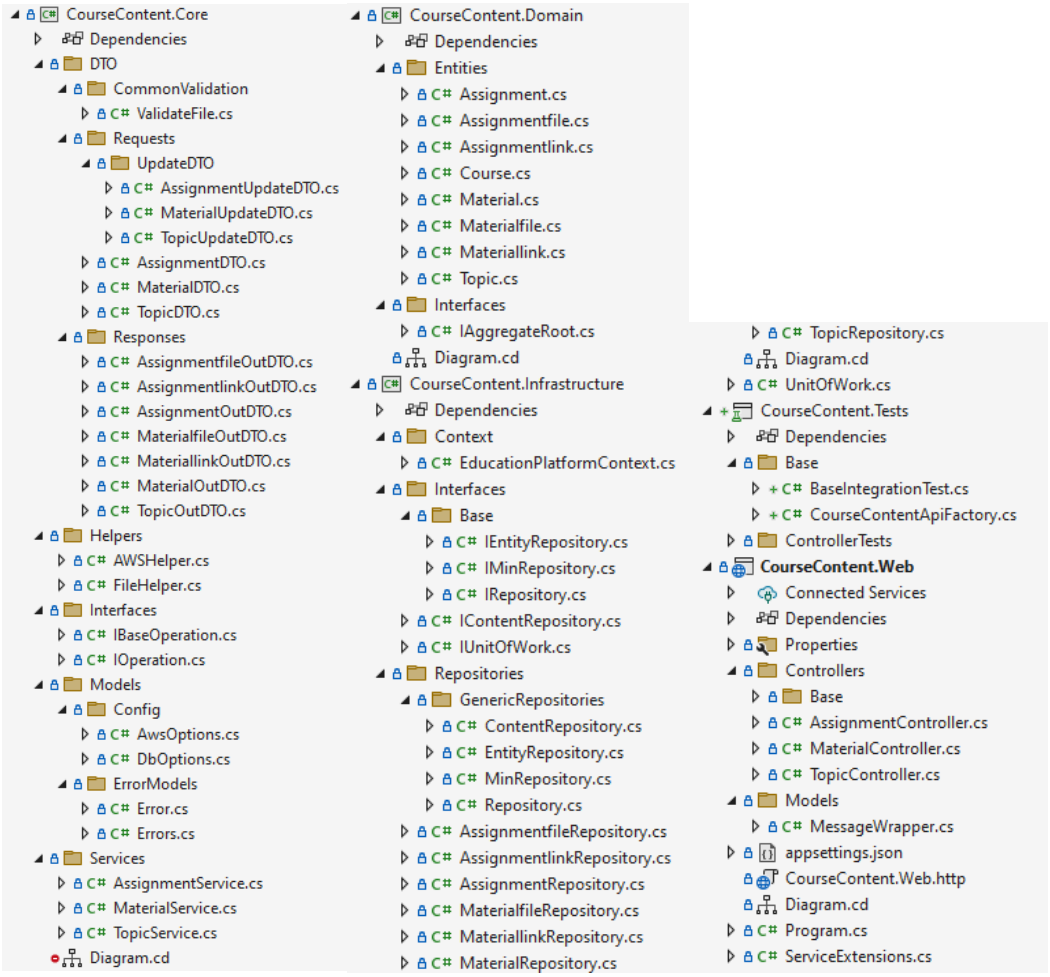

Рисунок Г.5 – Файлова структура модуля адміністрування контенту

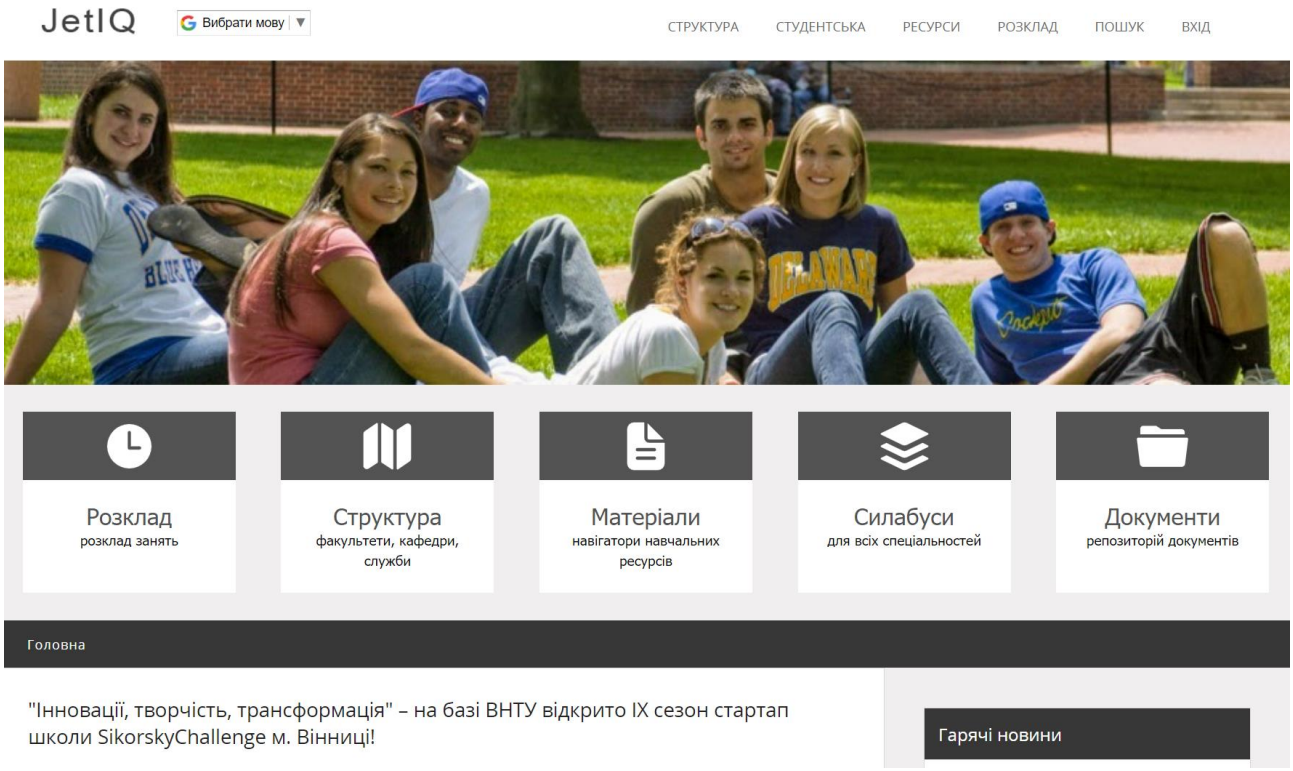

Рисунок Г.6 – Головна сторінка платформи JetIQ

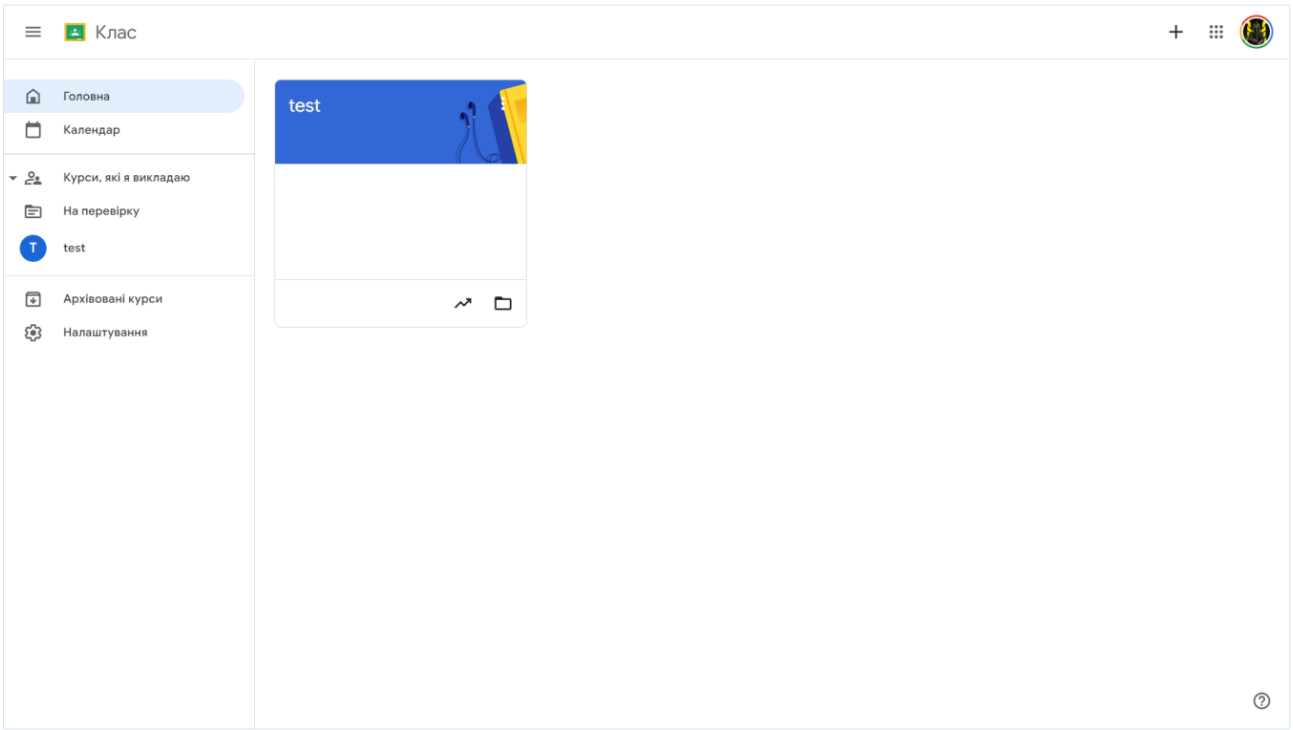

Рисунок Г.7 – Головна сторінка платформи Classroom