

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СПІЛЬНОГО ВІДТВОРЕННЯ МЕДІАРЕСУРСІВ
SOFTWARE FOR SHARED PLAYBACK OF MEDIA RESOURCES

Виконав: студент 4 курсу, групи КНТ-127

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Камєнський Д.В.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Камєнського Дениса Володимировича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення спільного відтворення медіаресурсів
Software for Shared Playback of Media Resources
- керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) 17 травня 2021 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Льовкін В.М., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання “10” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент

_____ **Каменський Д. В.**
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ **Льовкін В. М.**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 128 с., 51 рис., 4 табл., 3 дод., 22 джерела.

CQRS, БАГАТОШАРОВА АРХІТЕКТУРА, КІМНАТА, КОНТЕНТ, МЕДІАРЕСУРС, ПРЕДМЕТНО-ОРІЄНТОВНЕ ПРОЄКТУВАННЯ

Об'єкт дослідження – процес відтворення медіаресурсів для групи людей у синхронному режимі.

Предмет дослідження – програмне забезпечення спільного відтворення медіаресурсів.

Мета роботи – розроблення програмного забезпечення спільного відтворення медіаресурсів для надання можливості одночасного споживання контенту спільнотою людей.

Матеріали, методи та технічні засоби: об'єктно-орієнтовне програмування, предметно-орієнтовне проєктування, CQRS, мова програмування C#, фреймворк для розробки вебзастосунків ASP.NET Core, СКБД Microsoft SQL Server, фреймворк доступу до даних Entity Framework Core.

Результати. Розроблено програмне забезпечення спільного відтворення медіаресурсів для споживання контенту спільнотою людей.

Висновки. Проаналізовано поняття та процеси предметної області. Досліджено архітектурні особливості реалізації розроблюваної системи. Спроектовано та реалізовано програмне забезпечення відповідно функціональних та архітектурних вимог. Протестостовано та проведено порівняння розробленої програми із існуючими рішеннями.

Галузь використання – сфера онлайн-розваг для спільного проведення часу спільнотою людей.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 128 pages, 51 figures, 4 tables, 3 appendixes, 22 sources.

CONTENT, CQRS, DOMAIN-DRIVEN DESIGN, ROOM, MEDIA RESOURCE, N-LAYER ARCHITECTURE

Object of study is the process of reproducing media resources for a group of people in synchronous mode.

Subject of study is software for shared playback of media resources.

The purpose of study is development of software for shared playback of media resources for a group of people.

Materials, methods and tools: object-oriented programming, domain-driven design, CQRS, C# programming language, web application development framework ASP.NET Core, Microsoft SQL Server, data access technology Entity Framework Core.

Results. Software for shared playback of media resources for content consumption by the community of people has been developed.

Conclusions. Analyzed concepts and processes of the domain. Investigated architectural features of realization of the developed system. The software is designed and implemented in accordance with the functional and architectural requirements. The developed program was tested and compared with existing implementations.

Field of use is online entertainment area for a group of people spending time together.

ЗМІСТ

Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Постановка проблеми	11
1.2 Існуючі рішення	12
1.2.1 Watch2gether	12
1.2.2 Twoseven	14
1.2.3 Kast.....	16
1.3 Вимоги до програмного забезпечення	18
1.4 Постановка завдання до роботи.....	19
1.5 Висновки за розділом 1	19
2 Проєктування програмного забезпечення	20
2.1 Функціональні вимоги до програми	20
2.2 Проєктування архітектури програмного забезпечення.....	23
2.2.1 Основа структури системи	23
2.2.2 Організація шару предметної області	25
2.2.3 Domain Driven Design	27
2.2.4 Організація шару прикладної логіки.....	28
2.3 Проєктування ER-моделі.....	29
2.4 Вибір мови програмування та платформи.....	31
2.4.1 Порівняння мов програмування	31
2.4.2 Огляд платформи для розробки.....	33
2.5 Вибір джерел даних та способу взаємодії з ними	33
2.6 Висновки за розділом 2	34
3 Основні рішення щодо реалізації компонентів системи.....	35
3.1 Алгоритм функціонування програмного забезпечення	35
3.1.1 Ідентифікація користувача.....	35
3.1.2 Керування кімнатами.....	36

3.1.3 Пошук та відтворення контенту	37
3.1.4 Звільнення кімнати	38
3.1.5 Можливі стани кімнати	38
3.2 Опис структури програмного забезпечення.....	39
3.2.1 Mediaverse.Domain	39
3.2.1.1 Контекст спільного відтворення контенту	40
3.2.1.2 Контекст пошуку контенту	45
3.2.1.3 Контекст аутентифікації.....	46
3.2.2 Mediaverse.Application	46
3.2.3 Mediaverse.Web.....	50
3.2.4 Mediaverse.Infrastructure	51
3.2.5 Mediaverse.Domain.Tests.....	52
3.3 Особливості реалізації	53
3.3.1 Аутентифікація.....	53
3.3.2 Синхронізація даних між глядачами.....	54
3.3.3 Узгодженість даних	56
3.3.5 Інтеграція з медіаресурсами.....	57
3.3.6 Кешування контенту	58
3.3.7 Оброблення помилок	59
3.3.8 DTO для репозиторіїв	60
3.4 Висновки за розділом 3	60
4 Експлуатація, тестування та експериментальне дослідження програми	62
4.1 Призначення програми	62
4.2 Умови виконання програми	62
4.2.1 Апаратні вимоги програми до серверу	62
4.2.2 Апаратні вимоги програми до клієнту.....	63
4.3 Робота з програмою	63
4.4 Тестування програми.....	69
4.5 Порівняння результатів	70
4.6 Висновки за розділом 4	72

Висновки	73
Перелік джерел посилання	75
Додаток А Технічне завдання	77
Додаток Б Текст програми	81
Додаток В Слайди презентації.....	119

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CQRS – Command Query Responsibility Segregation;

DTO – Data Transfer Object;

БД – база даних.

ВСТУП

Із технологічним прогресом все більше сфер нашого життя переноситься в онлайн-формат, все більше людей мають облікові записи у соціальних мережах та проводять вільний час у інтернеті. Також слід зазначити, що все більше популярності отримують потокові медіаресурси, у порівнянні до класичного телебачення та радіо, коли люди обирають спосіб проведення вільного часу.

Однак більшість подібних сервісів реалізують відтворення контенту лише на один пристрій, що є не найкращим варіантом для спільноти людей, що не можуть зібратися в одному місці. Особливо це стало актуальним протягом останнього року і карантинних обмежень.

Актуальність роботи полягає в дослідженні архітектурних особливостей при проєктуванні та розробці програмного забезпечення спільного відтворення медіаресурсів для надання можливості спільного споживання контенту спільнотою людей та у створенні відповідного застосунку.

В рамках кваліфікаційної випускної роботи проводиться аналіз предметної області та існуючих рішень поставленої задачі. Базуючись на результатах аналізу, формуються вимоги та, зважаючи на них, розробляється програмне забезпечення, що має задовольняти як функціональним вимогам, так і вимогам до архітектурних характеристик.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка проблеми

В рамках цієї роботи предметною областю є відтворення контенту зі сторонніх джерел. Її основними поняттями є:

- зміст (контент) – ідеї, що містяться в творі, виступі чи фільмі [1];
- попередній перегляд (прев'ю) – можливість перегляду або прослуховування невеликої частини контенту для оцінки, загального ознайомлення тощо;
- список відтворення (плейлист) – перелік контенту, що відтворюється;
- медіаресурс – джерело контенту.

Основною метою роботи є розроблення програмного забезпечення спільного відтворення медіаресурсів для надання можливості одночасного споживання контенту спільнотою людей. Враховуючи специфіку поставленої задачі, додаються такі поняття:

- кімната (перегляду) – абстракція для синхронного відтворення відео чи аудіо для певної групи людей при використанні застосунку (в рамках цієї роботи кімнати поділяються відкриті за закриті: відкриті дозволяють долучатися за запрошенням або через пошук, закриті доступні лише за запрошенням);
- власник (кімнати) – користувач, що створює кімнату і наділений відповідними правами у керуванні її налаштуваннями;
- глядач – користувач, що доєднався до існуючої кімнати для споживання контенту спільно з іншими глядачами;
- запрошення – унікальний ідентифікатор, за допомогою якого глядач може доєднатися до кімнати.

Основними задачами програмного забезпечення для спільного відтворення медіаресурсів є:

- синхронізація відтворення контенту у глядачів
- надання можливості керування відтворенням;
- надання можливості пошуку контенту;
- надання можливості керування кімнатами перегляду.

1.2 Існуючі рішення

Як правило, програмні рішення вказаної проблеми вирішують іншу специфічну проблему та надають спільне відтворення контенту лише частково, або пропонують продукт із помилками, яких можна було уникнути при впровадженні. Це є досить важливим, бо загальна ідея має великий потенціал, як для бізнесу так і для розробників.

1.2.1 Watch2gether

Watch2gether – це сервіс, який дозволяє переглядати та прослуховувати різні види контенту спільно із іншими користувачами з різних медіаресурсів [2].

Переваги:

- можливість використання основної функціональності без авторизації;
- наявний суттєвий вибір джерел вебресурсів, з яких можна відтворювати контент;
- пошук за ключовими словами або за посиланням;
- наявність власної бібліотеки відео, матеріали з якої автоматично відтворюються при створенні кімнати;
- керування плейлистами;
- наявна можливість комунікації за допомогою чату, камери та мікрофону.

Недоліки:

- керування оригінальним віджетом перехоплюється, але із затримкою, через що відтворення на різних пристроях може бути не синхронним;
- перевантажений користувацький інтерфейс;
- пошук за ключовими словами доступний не для усіх платформ;
- велика кількість реклами.

На рис. 1.1 продемонстрована головна сторінка сервісу Watch2gether. Тут надається можливість аутентифікації, створення нової кімнати перегляду, переходу до спільноти, вказано базову інформацію про вебсайт та його використання. Слід зазначити, що створення кімнати дозволяється без реєстрації, що є однією з важливіших конкурентних переваг застосунку.

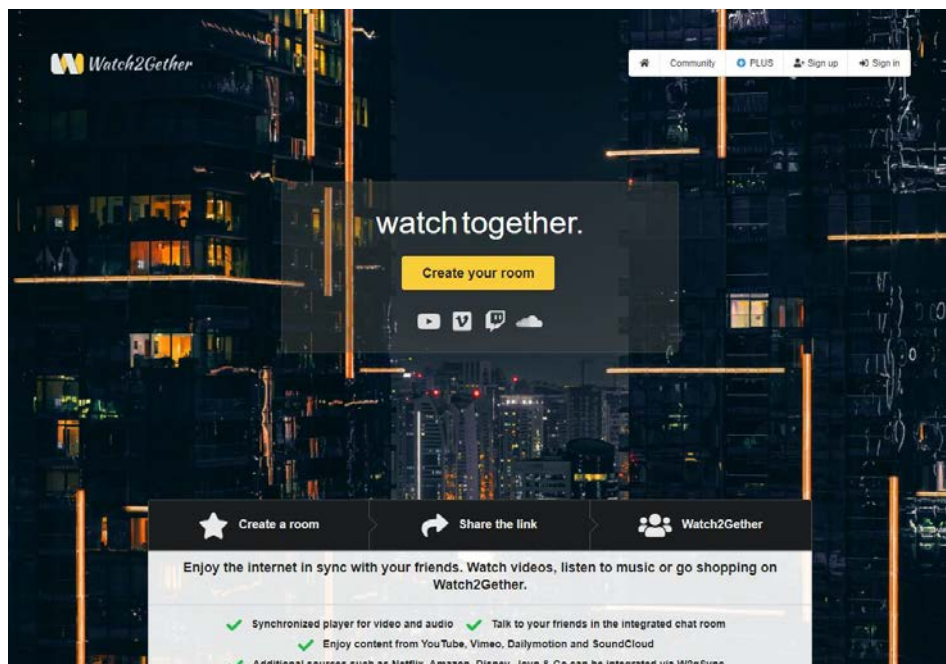


Рисунок 1.1 – Головна сторінка сервісу Watch2gether

На рис. 1.2 продемонстровано сторінку кімнати перегляду сервісу Watch2gether. Тут надаються можливості пошуку та відтворення контенту, керування плейлистами та налаштуваннями кімнати. Серед особливостей реалізації функціональності можна виділити автоматичне відтворення

контенту із власної відеотеки системи при створенні нової кімнати та можливість перегляду відео та аудіо без додавання до плейлиста кімнати [2].

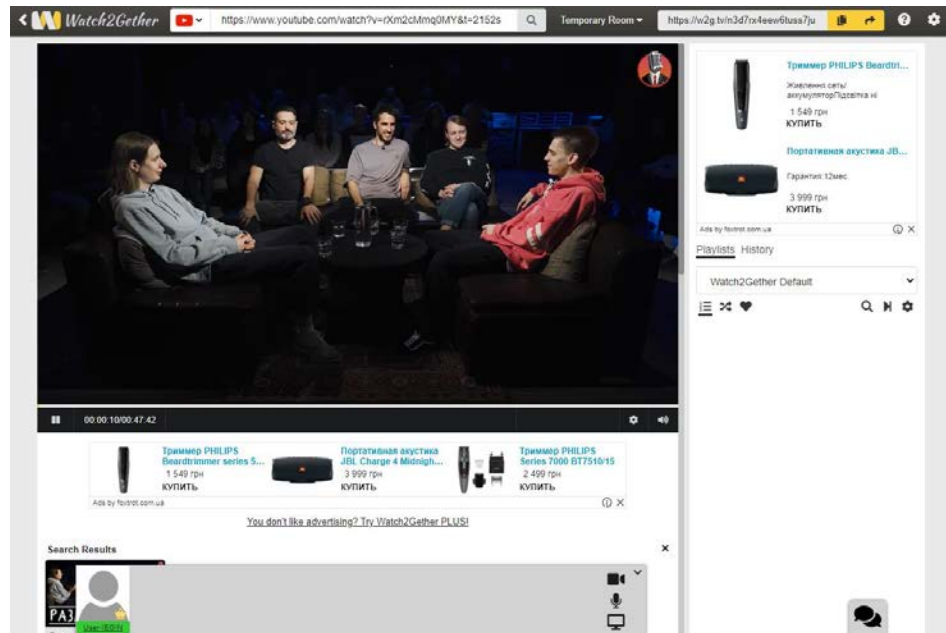


Рисунок 1.2 – «Кімната» для перегляду відео на Watch2gether

1.2.2 Twoseven

Twoseven – це вебсайт, призначений для спільного перегляду відео з декількох популярних платформ із можливістю відтворення приватних медіа-ресурсів [3].

Переваги:

- простота;
- наявна можливість комунікації за допомогою чату, камери та мікрофону;
- відтворення приватних відео;
- можливість поширювати зображення екрану;
- налаштування «кімнати»;
- список друзів.

Недоліки:

- пошук лише за посиланням;
- необроблена інфраструктурна помилка відтворення показується користувачам;
- обов’язкова реєстрація у системі.

На рис. 1.3 продемонстровано головну сторінку сервісу Twoseven. Після реєстрації або аутентифікації у системі тут надаються можливості створення кімнати, керування переліком «друзів», оновлення користувацьких даних та ознайомлення із інформацією про вебсайт. Також є можливість поширення інформації про застосунок на різних соціальних платформах.

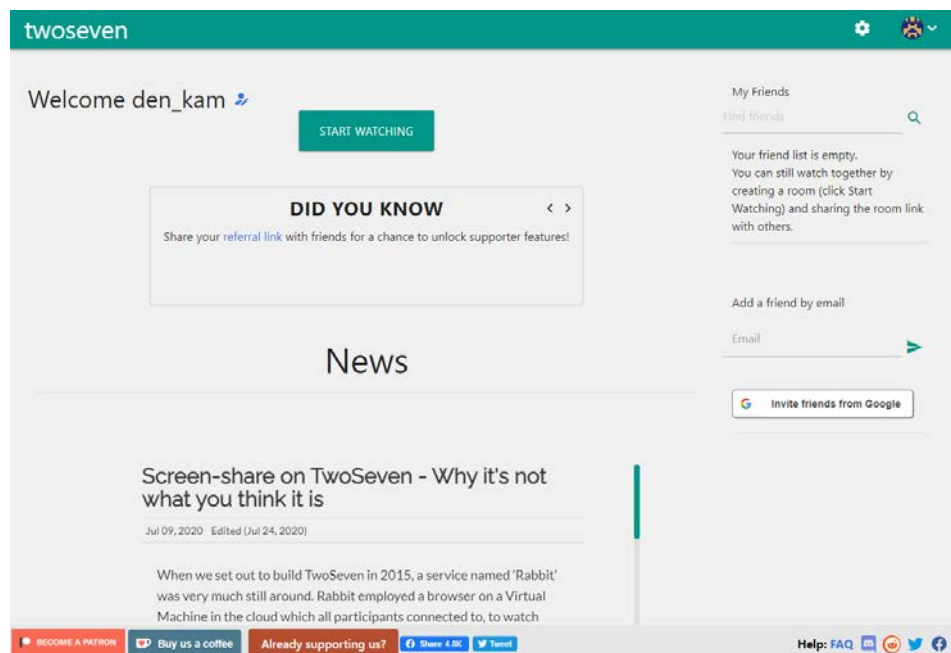


Рисунок 1.3 – Головна сторінка сервісу Twoseven

На рис. 1.4 продемонстровано сторінку кімнати перегляду сервісу Twoseven. Тут доступні пошук за посиланням та синхронізоване відтворення відео з різних платформ. Також тут наявний чат та можливість комунікації за допомогою мікрофону та вебкамери. До конкурентних переваг варто віднести можливість відтворення власних відео з пристрою [3].

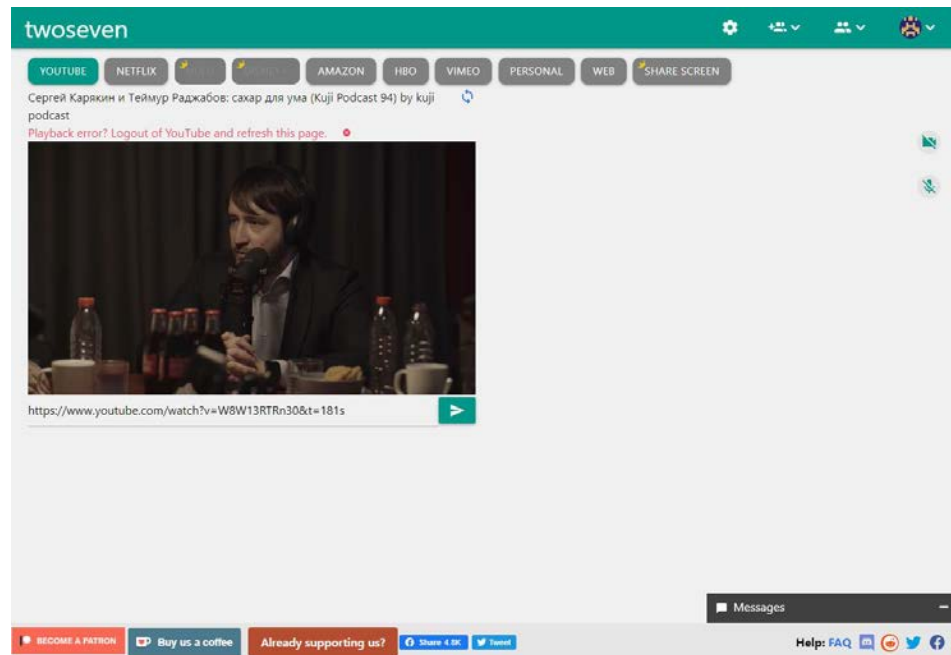


Рисунок 1.4 – «Кімната» для перегляду відео на Twoseven

1.2.3 Kast

Kast – додаток для спільного перегляду відео, кіно та серіалів з трьох основних медіа-ресурсів, серед яких є власна велика бібліотека різних кіно та шоу [4].

Переваги:

- можливість поширювати зображення екрану;
- наявна можливість комунікації за допомогою чату, камери та мікрофону;
- можливість налаштування кімнати;
- можливість створення і збереження декількох кімнат;
- список друзів;
- пошук за ключовими словами або посиланням;
- приємний користувацький інтерфейс;
- розподілення ролей і можливість модерації кімнат;
- два типи приватних кімнат.

Недоліки:

- невелика кількість джерел медіа-ресурсів;
- присутні необроблені програмні помилки, що призводять до незрозумілої поведінки застосунку;
- обов’язкова реєстрація у системі.

На рис. 1.5 продемонстровано головну сторінку застосунку Kast. Тут надається можливість керування кімнатами, до яких користувач має доступ. Також є управління переліком «друзів» та користувацькими даними, комунікація у чаті та перегляд повідомлень від системи.

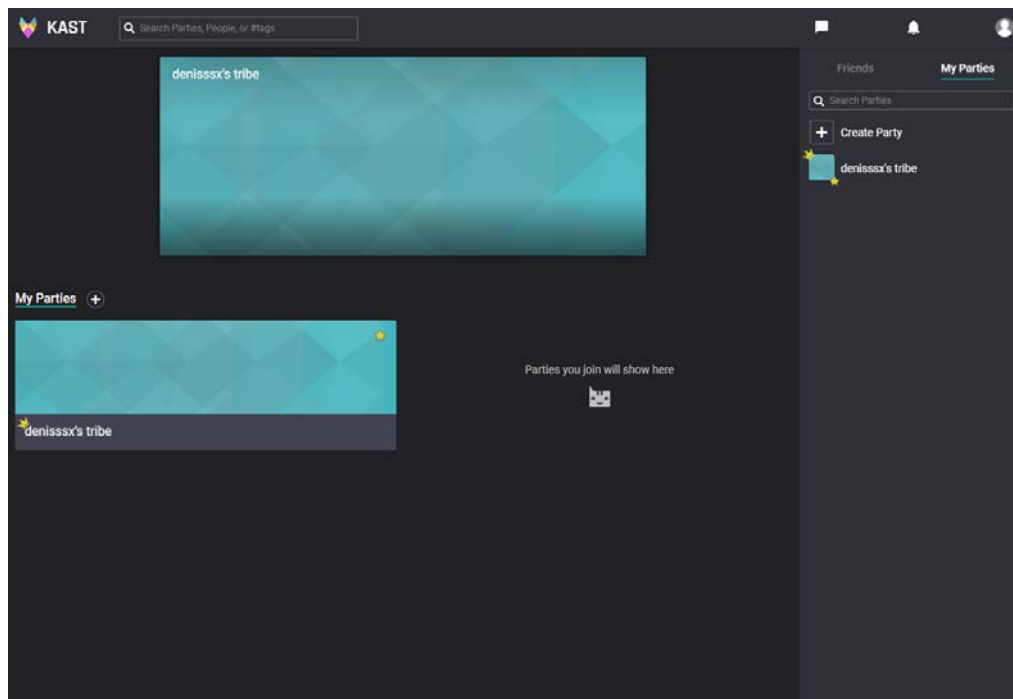


Рисунок 1.5 – Головна сторінка сервісу Kast

На рис. 1.6. продемонстровано сторінку кімнати сервісу Kast. Тут користувач отримує можливості для пошуку та відтворення відео з доступних платформ. Також доступні спільний чат, повідомлення від системи та оновлення користувацьких даних. Серед особливостей реалізації системи слід виділити можливість трансляції віку веббравзеру та наявність великої власної відеотеки [4].

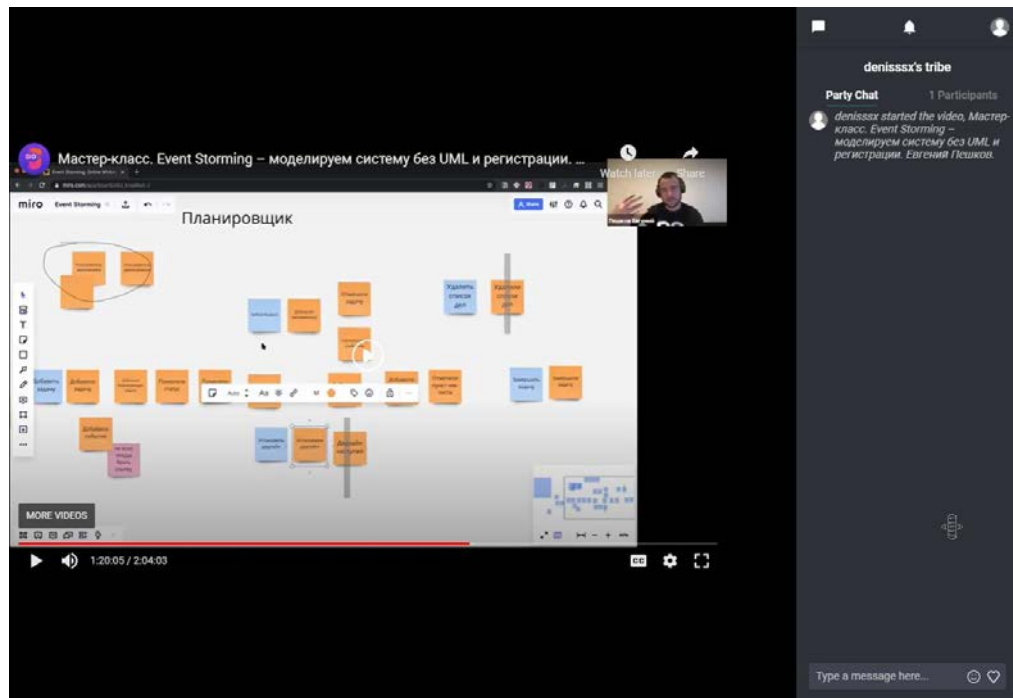


Рисунок 1.6 – «Кімната» для перегляду відео на Kast

1.3 Вимоги до програмного забезпечення

Проаналізувавши предметну область та існуючі на ринку рішення, можна сформулювати наступні вимоги до системи:

- синхронне відтворення контенту для глядачів;
- пошук контенту на доступних медіа-ресурсах за ключовими словами або посиланням;
- авторизація та можливість анонімного використання сервісу;
- можливість вибору і поповнення інформаційних джерел;
- керування списком контенту (плейлистами);
- гнучкість та здатність до розширення функціональності;
- якісна обробка програмних помилок із відображенням зручним для користувача.

1.4 Постановка завдання до роботи

Основною метою даної роботи є розроблення програмного забезпечення спільного відтворення медіаресурсів для надання можливості одночасного споживання контенту спільнотою людей.

Завдання до роботи:

- визначення архітектурних та технічних вимог до програмного забезпечення спільного відтворення медіареурсів;
- вибір архітектурних підходів та технічних засобів розроблюваної системи відповідно до вимог;
- проєктування програмного забезпечення;
- реалізація та впровадження розробленої системи;
- тестування та аналіз на відповідність поставленим вимогам.

1.5 Висновки за розділом 1

У результаті аналізу предметної області було виокремлено та досліджено основні поняття та проаналізовано існуючі програмні реалізації вирішуваної проблеми. В ході аналізу було виділено переваги та недоліки, із урахуванням яких було сформовано список вимог до розроблюваної системи.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Функціональні вимоги до програми

Проаналізувавши основні процеси предметної області було сформовано наступні функціональні вимоги до розроблюваного програмного забезпечення:

- на головній сторінці застосунку не ідентифікований користувач повинен мати можливість перейти до реєстрації, аутентифікації чи пропустити ці етапи та перейти до використання сервісу;

- якщо користувача ідентифіковано, система повинна перенаправляти його з головної сторінки до сторінки керування кімнатами перегляду;

- на сторінці керування кімнатами перегляду користувач повинен мати можливість створити нову кімнату перегляду, доєднатися до існуючої за запрошенням або через пошук, якщо кімната відкрита;

- користувач повинен мати можливість доєднатися до кімнати перегляду за посиланням: якщо користувач не зареєстрований, система повинна створити для нього тимчасовий обліковий запис;

- тільки відкриті кімнати мають бути відображенні на сторінці керування кімнатами перегляду у відповідній секції;

- користувачі без запрошення не повинні мати можливості доєднатися до закритих кімнат перегляду;

- якщо до кімнати перегляду доєдналася максимально дозволена кількість користувачів, система повинна відобразити відповідне повідомлення користувачу та перенаправити на сторінку керування кімнатами;

- при створенні нової кімнати користувач повинен мати можливість вказати базову інформацію, задати налаштування та обрати існуючий плейлист або створення нового;

- при переході користувача на сторінку нової кімнати система повинна показувати інформацію, про те, що контент для відтворення не обрано;

- при переході користувача на сторінку існуючої кімнати система повинна відображати останній відтворюваний контент;

- система має не допускати перехід за контентом у порожньому плейлисті;

- на сторінці кімнати користувач повинен мати можливість виконати пошук контенту за посиланням або за ключовими словами, а після отримання результатів користувач повинен мати можливість почати відтворення або додати бажаний контент до плейлиста;

- на сторінці кімнати користувач повинен мати можливість видалити контент з плейлиста;

- на сторінці відтворення контенту користувач повинен мати можливість змінити поточний плейлист та керувати іншими існуючими;

- відтворення контенту та пов'язані операції мають бути синхронними для власника та глядачів;

- при приєднанні до кімнати система повинна відобразити контент, що відтворюється у кімнаті перегляду відповідно до поточного стану та часу початку відтворення;

- при виході власника з кімнати система повинна обирати нового;

- при виході останнього глядача система повинна видаляти кімнату;

- користувач повинен мати можливість виходу із системи на будь-якому етапі, а після виходу його має бути перенаправлено на головну сторінку.

Проаналізувавши функціональні вимоги до програмного забезпечення та структурувавши їх, було сформульовано наступні прецеденти:

- створення та видалення кімнати перегляду;

- створення та видалення поточного плейлиста;

- приєднання до кімнати перегляду;
- вихід із кімнати перегляду;
- керування відтворенням контенту;
- керування контентом у плейлисті;
- пошук контенту на медіаресурсах.

Отриману діаграму прецедентів відповідно до розділення за ролями представлено на рис. 2.1.

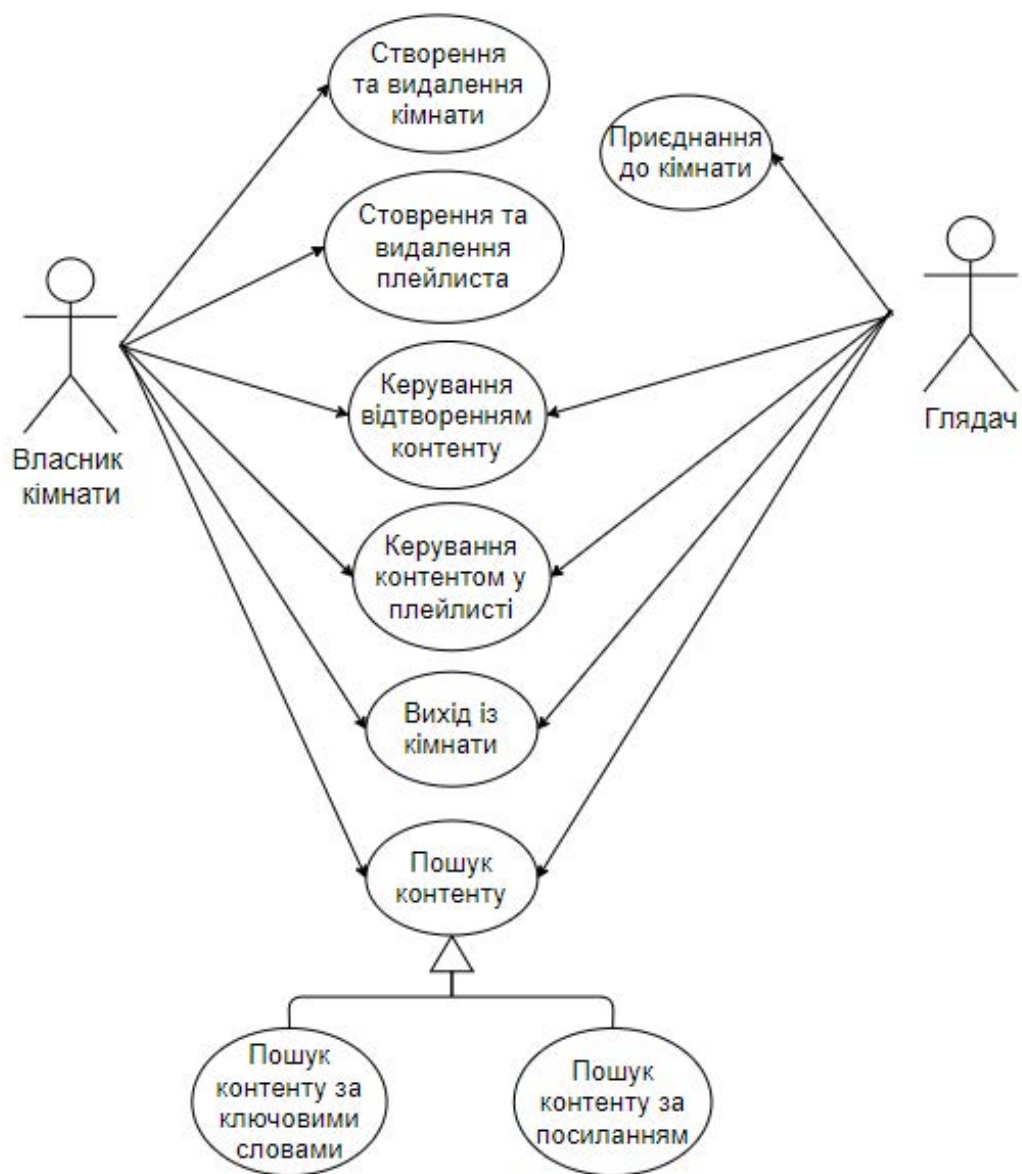


Рисунок 2.1 – Діаграма прецедентів

2.2 Проєктування архітектури програмного забезпечення

Проаналізувавши предметну область та вимоги до програмного забезпечення, було сформовано наступні необхідні характеристики архітектури:

- здатність до розширення програмного відображення об'єктів та правил предметної області;
- гнучка та проста частина інтеграції із медіаресурсами;
- універсальний механізм для взаємодії з власними джерелами даних;
- готовність до розширення функціональності системи та більшого навантаження;
- конкурентне використання спільних ресурсів;
- контроль узгодженості даних.

2.2.1 Основа структури системи

Зважаючи на вимоги до системи, які передбачають складність та потенційне розширення інфраструктури та бізнес-правил, впливає необхідність розділення відповідальності між частинами програми за видами логіки. Саме тому, в якості основи структури системи було обрано багат шарову архітектуру.

Багат шарова архітектура – це широко прийнята техніка, яка допомагає зменшити складність проєкту і поліпшити повторне використання коду [5]. Цей архітектурний підхід передбачає розділення системи за призначенням логіки на спеціальні шари. У класичному варіанті багат шарова архітектура складається з шарів представлення, бізнес-логіки та взаємодії з базою даних (БД). Але із зростанням складності проєктованих систем шар бізнес-логіки обтяжується заплутаними інваріантами, різновидами корисної логіки та використанням зовнішніх залежностей.

Виходячи з описаної проблеми та зважаючи на складність розроблюваної системи, було обрано більш гнучкий варіант багатошарової архітектури, який складається з наступних шарів:

- шар представлення, що забезпечує інтерфейс для користувача: використовує прикладний рівень для виконання сценаріїв застосування системи;

- шар прикладної логіки, що виступає посередником між рівнями представлення та домену, використовує бізнес-об’єкти для виконання конкретних практичних задач;

- шар предметної області – включає бізнес-об’єкти та пов’язані правила, це центральна частина програми;

- шар інфраструктури, що надає загальні технічні можливості, які підтримують вищі рівні, переважно за допомогою сторонніх залежностей [5].

На рис. 2.2 продемонстровано залежності між шарами та взаємодію із БД в рамках обох варіантів архітектури.

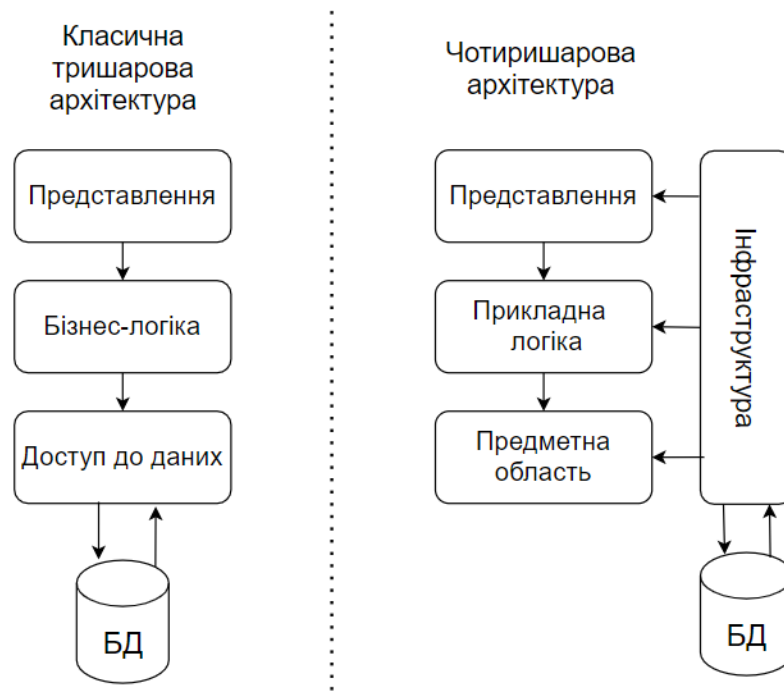


Рисунок 2.2 – Залежності між шарами та взаємодія із БД у тришаровій та чотиришаровій архітектурі

2.2.2 Організація шару предметної області

Як було зазначено у попередній частині, шар домену є головним у системі, бо саме тут мають бути сконцентровані усі значущі для бізнесу інваріанти та поняття. Тому проєктування цього рівня є найважливішим і має бути детально обмірковано із урахуванням дрібниць та можливих перешкод при впровадженні.

Найбільш поширеними опціями організації шару предметної області є наступні архітектурні рішення:

- Transaction Script;
- Domain Model [6].

На цьому рівні стає дуже актуальною вимога контролю узгодженості даних, бо умови для досягнення консистентності диктуються саме правилами предметної області. Тому при розгляді зазначених типових рішень слід розглядати і зручність та надійність досягнення цієї архітектурної характеристики.

Типове рішення «сценарій транзакції» передбачає опис бізнес-логіки у вигляді низки транзакцій, кожній з яких у відповідність ставиться власний сценарій виконання [6]. Транзакції виконують деяку корисну логіку та виконують взаємодію із БД. Завдяки процедурному опису бізнес-логіки та тому, що він не вимагає труднощів організації структури системи, цей підхід є одним з найпростіших при проєктуванні шару предметної області [6]. Але через зосередження правил предметної області лише у транзакціях, де вони необхідні, такий підхід значно підвищує ризик дублювання коду і відповідно несе загрозу для узгодженості даних.

Типове рішення «модель предметної області» передбачає створення мережі взаємопов'язаних об'єктів, кожен з яких представляє якусь осмислену сутність та повністю інкапсулює пов'язані із нею правила предметної області [6]. Використання такого підходу вимагає зосередження поведінки та бізнес-

правил об'єктів у відповідних їм програмних відображеннях, завдяки чому об'єкти здатні контролювати узгодженість даних, якими володіють та більш якісно відображають предметну область, через високу концентрацію знань.

У табл. 2.1 виконується порівняння описаних підходів до організації шару предметної області.

Таблиця 2.1 – Порівняння підходів організації шару домену

Підхід Критерій	Transaction Script	Domain Model
Складність впровадження	Низька	Висока
Складність підтримки	Низька – на початку розробки системи, висока – із розвитком системи	Помірна
Складність впровадження нової функціональності	Низька – на початку розробки системи, висока – із розвитком системи	Помірна
Концентрація знань про предметну область	Низька	Висока
Узгодженість даних	Помірна – на початку розробки системи, низька – із розвитком системи	Висока
Структурованість	Помірна	Висока

Завдяки високому рівню узгодженості даних, структурованості та помірній складності підтримки системи із розвитком, для реалізації шару предметної області було обрано типове рішення «модель предметної області».

2.2.3 Domain Driven Design

Для розробки якісної моделі предметної області було обрано підхід «предметно-орієнтоване проектування» [7], яке першочергово акцентує увагу розробників при проектуванні саме на об'єктах та правилах домену, а не на використовуваних технологіях.

Головною метою використання цього підходу є боротьба зі складністю предметної області. Одним із важливіших інструментів для досягнення цього є «обмежені контексти» [7], які кластеризують пов'язані між собою поняття у певних лінгвістичних або логічних межах. Використання такого розділення гарантує однозначність понять домену та більшу зосередженість розробників на необхідних частинах програми при впровадженні нової функціональності.

Спираючись на розглянуті на етапі аналізу предметної області поняття та бізнес-процеси, було виділено наступні «обмежені контексти»:

- контекст аутентифікації – містить поняття та процеси для аутентифікації та має контролювати використання системи без проходження реєстрації завдяки автоматичному створенню тимчасового облікового запису для користувача;

- контекст спільного відтворення медіаресурсів – включає в себе поняття та пов'язані правила для кімнат, їх власників та глядачів, а також керування плейлистами та синхронним відтворенням контенту;

- контекст пошуку контенту – відповідає за пошук контенту на медіаресурсах за ключовими словами або за посиланням;

– контекст джерел контенту – позначає сторонні медіаресурси, з яких отримується та відтворюється контент.

2.2.4 Організація шару прикладної логіки

Найбільш популярними опціями для реалізації шару прикладної логіки є сервісний підхід та Command Query Responsibility Segregation (CQRS) [8]. В табл. 2.2 приведено порівняння цих підходів та описано базовий принцип.

Таблиця 2.2 – Порівняння підходів організації шару прикладної логіки

Підхід Критерій	Сервісний підхід	CQRS
	Сервісний підхід	CQRS
Базовий принцип	Прикладна логіка розміщується у спеціальних службових класах; методи описують сценарії використання	Прикладна логіка поділяється за типом на модифікацію і «читання» об'єктів; кожен сценарій розміщується у відповідному класі
Складність впровадження	Низька	Помірна
Розділення моделей запису та читання	У класичному варіанті – немає	Можливе
Продуктивність	Помірна	При впровадженні розділу моделей запису та читання – висока
Структурованість	Помірна	Висока

Не зважаючи на помірний рівень складності впровадження CQRS, його переваги мають більший сенс для розроблюваної системи у перспективі, враховуючи вимоги розширюваності та готовності до великих навантажень. Саме тому цей підхід було обрано для реалізації шару прикладної логіки програми – усі сценарії використання системи будуть програмно реалізовані у відповідних класах запитів чи команд.

Таким чином структура системи та взаємодія між шарами зміниться як продемонстровано на рис. 2.3.

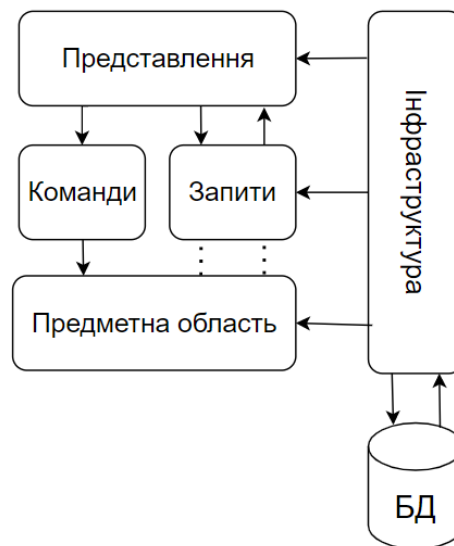


Рисунок 2.3 – Чотиришарова архітектура із використанням CQRS

2.3 Проєктування ER-моделі

Базуючись на поняттях домену, що були знайдені на етапі аналізу предметної області, було розроблено наступну модель «сутність-зв'язок»:

– User – одна із основних сутностей, яка відноситься до контексту аутентифікації і тому містить лише базову інформацію про користувача, необхідну для його ідентифікації у системі;

– Room – одна з основних сутностей, яка відповідно є відображенням поняття домену «кімната» із контексту спільного відтворення контенту,

містить базову інформацію про кімнату, власника та поточні налаштування;

– RoomViewer – сутність для відображення залежності між кімнатою перегляду та її глядачами;

– Playlist – одна з основних сутностей, яка відповідно є відображенням поняття домену «плейлист» із контексту спільного відтворення контенту, містить назву плейлиста та інформацію про його власника;

– PlaylistItem – сутність для відображення зв'язку контенту із плейлистом;

– CurrentRoomContent – сутність, що містить інформацію про поточний контент, що відтворюється у кімнаті: слід зазначити, що рішення стосовно залежностей між відтворюваним контентом і кімнатою більш гнучке, ніж зв'язок між відтворюваним контентом із поточним плейлистом, таким чином у кімнаті можна відтворювати контент, що не міститься у плейлисті, що значно покращує користувацький досвід.

На рис. 2.4 продемонстровано схему БД.

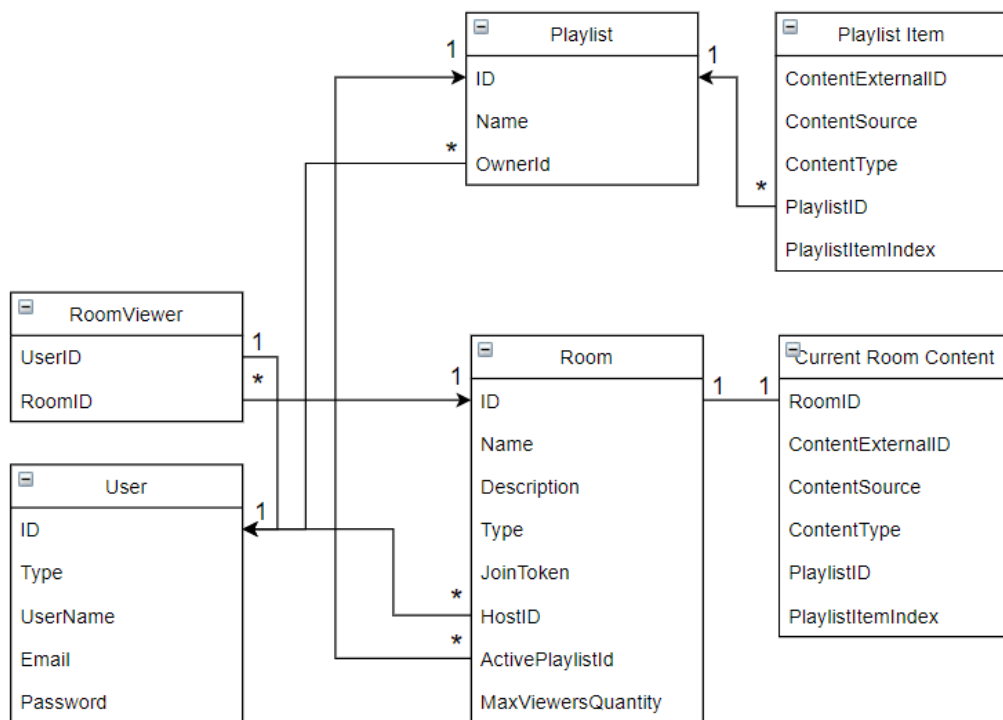


Рисунок 2.4 – Схема БД

2.4 Вибір мови програмування та платформи

Зважаючи на проведений аналіз вимог до архітектури та підходів, що будуть використані, для розробки проєкту необхідна об'єктно-орієнтовна мова програмування із платформою для розробки, що може взяти на себе більшу частину інфраструктури та надає каркас вебзастосунку з самого початку.

2.4.1 Порівняння мов програмування

До вказаних вимог до технічної архітектури найкраще підходять об'єктно-орієнтовні мови програмування C# та Java. Вони обидві постачаються разом із середою виконання, що бере на себе більшість інфраструктурних аспектів роботи, надаючи розробнику більше часу на реалізацією саме своїх ідей. Крім цього для обох мов програмування є власні колекції фреймворків, які значно спрощують та пришвидшують роботу.

C# – це сучасна, об'єктно-орієнтована та типобезпечна мова програмування. Вона дозволяє розробникам створювати багато типів безпечних та надійних додатків, що працюють в екосистемі .NET. Також C# забезпечує мовні конструкції для прямої підтримки концепцій компонентів, роблячи C# природною мовою, якою можна створювати та використовувати програмні компоненти [9].

Java – це загальноприйнята, заснована на класах, об'єктно-орієнтована мова програмування, призначена для зменшення залежностей реалізації. Це обчислювальна платформа для розробки додатків. Java швидка, безпечна та надійна [10].

У табл. 2.3 приведено результати порівняння зазначених мов програмування.

Таблиця 2.3 – Порівняння мов програмування

Мова програмування Критерій	C#	Java
Простота роботи із примітивними типами даних	++	+
Наявність Application Programming Interface (API) для роботи із колекціями	+	+
Підтримка показників на методи	+ (делегати)	+ - (за рахунок функціональних інтерфейсів)
Підтримка концепції властивостей	+	-
Поліморфізм	Вимагає використання відповідних ключових слів	За замовченням
Можливість наділяти перелічення поведінкою	+ - (лише за допомогою бібліотек)	+
Простота роботи із асинхронним кодом	++ (async/await)	+ (вимагає роботи з потоками)
Структуризація програми (розділення на субпроекти)	++	+

Таким чином, спираюсь на вказані відмінності мов програмування для розробки системи було обрано саме C# разом із платформою, в рамках якої вона постачається, завдяки більшій гнучкості та простоті використання.

2.4.2 Огляд платформи для розробки

.NET Core – це програмна платформа загального призначення з відкритим кодом, що підтримується корпорацією Майкрософт, це багатоплатформений фреймворк, який працює на операційних системах Windows, macOS та Linux [11].

.NET Core є модульним, легким і продуктивним фреймворком, через те що він включає лише основні функції, необхідні для запуску базової програми, інші функції пропонуються як окремі пакети, які додаються у програму за необхідності через спеціальний менеджер [11].

Завдяки легкості, продуктивності, модульності та новизні .NET Core саме його було обрано в якості базової програмної платформи. Також слід зазначити, що в рамках вказаної платформи також представлений фреймворк для створення вебзастосунків ASP.NET Core. Він наділений усіма перевагами батьківської платформи, саме тому його було обрано для розробки вебзастосунку.

2.5 Вибір джерел даних та способу взаємодії з ними

Спираючись на екосистему обраної платформи розробки та необхідність реляційної БД, в якості власного джерела даних було обрано Microsoft SQL Server – систему керування базами даних, що забезпечує високу продуктивність, доступність та масштабованість. Основна використовувана мова запитів – Transact-SQL. Використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства [12].

Для програмної взаємодії із БД буде використано набір інструментів Entity Framework Core – це відкрита, легка, розширювана та міжплатформна версія технології доступу до даних, що надає можливість об’єктно-реляційного відображення – автоматизованого механізму доступу та зберігання даних у БД [13].

2.6 Висновки за розділом 2

В розділі було проаналізовано вимоги до проєктованої системи та архітектури. Виходячи з цього було проаналізовано можливі підходи до реалізації програми та досягнення визначених архітектурних характеристик. На основі обраного підходу та вимог було порівняно відповідні мови програмування та платформи для розробки, в рамках яких вони постачаються. Після вибору мови програмування було розглянуто саму платформу та основний фреймворк, на базі якого буде побудовано систему. Спираючись на платформу, було обрано БД та інструмент для взаємодії із нею.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм функціонування програмного забезпечення

3.1.1 Ідентифікація користувача

Взаємодія користувача із програмою починається на головній сторінці, де йому надається можливість реєстрації, аутентифікації або переходу до анонімного використання сервісу, що передбачає автоматичне створення тимчасового облікового запису.

При виборі реєстрації або аутентифікації система перенаправляє користувача на відповідні сторінки, де він може вказати необхідну інформацію. Після успішного виконання обраного сценарію система перенаправляє користувача на сторінку керування кімнатами перегляду.

Слід зазначити, що при приєднанні не ідентифікованого користувача до кімнати за посиланням також виконується сценарій анонімного використання сервісу зі створенням тимчасового облікового запису.

На рис. 3.1 зображено діаграму діяльності описаного сценарію.



Рисунок 3.1 – Діаграма діяльності ідентифікації користувача

3.1.2 Керування кімнатами

Як вже було зазначено, після успішної авторизації користувача система перенаправляє його на сторінку керування кімнатами. Тут надається можливість доєднатися до кімнати за спеціальним кодом, або обрати доступну відкриту кімнату через пошук у відповідній секції сторінки та доєднатися до неї. Якщо кімната була звільнена або максимальна кількість глядачів була досягнута, користувачу, що намагався доєднатися, буде показано відповідне повідомлення. Крім цього, усі користувачі на зазначеній сторінці можуть створити нову кімнату із вказанням необхідної інформації. Після успішного створення кімнати або приєднання до існуючої система перенаправить користувача на сторінку відповідної кімнати перегляду.

На рис. 3.2 зображено діаграму діяльності описаного сценарію.

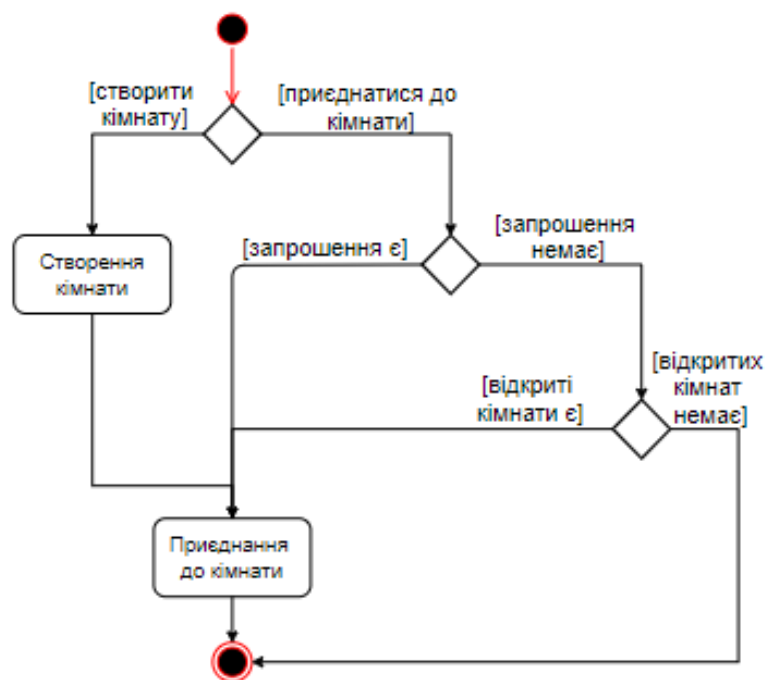


Рисунок 3.2 – Діаграма діяльності керування кімнатами

3.1.3 Пошук та відтворення контенту

Коли користувач потрапляє до кімнати перегляду, він отримує можливості пошуком необхідного контенту, керування змістом поточного плейлиста та відтворенням контенту.

Для пошуку користувачу надається спеціальна пошукова стрічка, в якій він може вказати посилання або ключові слова, за якими слід виконувати пошук. При пошуку за посиланням результатом буде запитуваний контент або повідомлення, що його не знайдено. При пошуку за ключовими словами буде відображено усі елементи контенту, що було знайдено на медіаресурсі. Після відображення результатів пошуку користувач може додати певний контент до поточного плейлиста або запустити відтворення одразу. Також, на цій сторінці користувач має можливість змінити поточний плейлист або видалити його.

На рис. 3.3 зображено діаграму діяльності описаного сценарію.

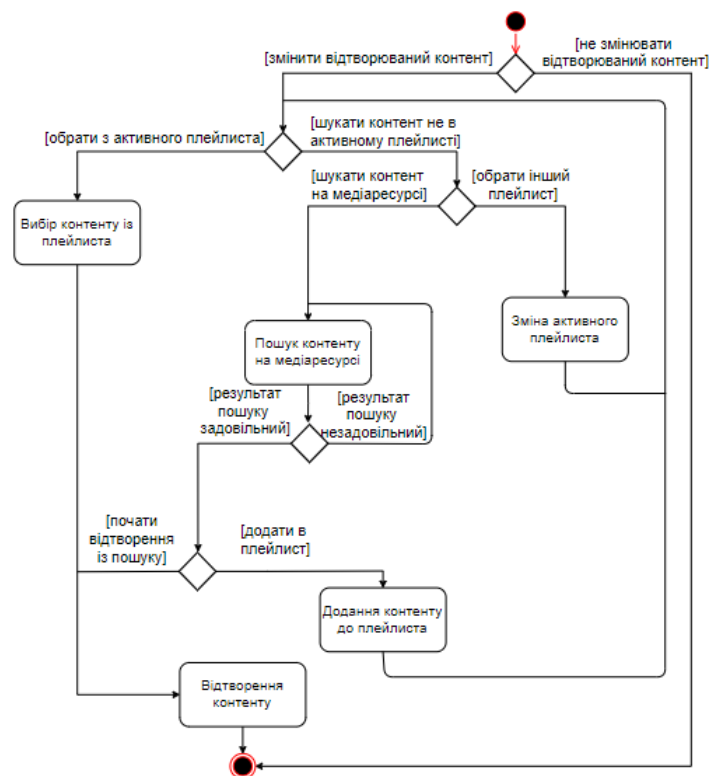


Рисунок 3.3 – Діаграма діяльності пошуку та відтворення контенту

3.1.4 Звільнення кімнати

Користувачі мають змогу вийти з кімнати, натиснувши на відповідну кнопку, перейшовши на сторінку керування кімнатами перегляду або при виході із облікового запису.

Слід зазначити, що за функціональними вимогами, при виході з кімнати останнього глядача, цикл життя кімнати завершується і вона видаляється. Після звільнення кімнати до неї неможливо доєднатись.

3.1.5 Можливі стани кімнати

Виходячи з описаного алгоритму функціонування програми можна створити діаграму станів кімнати перегляду (рис. 3.4), яка є однією з центральних сутностей предметної області та володіє найскладнішою системою станів, в яких може перебувати.

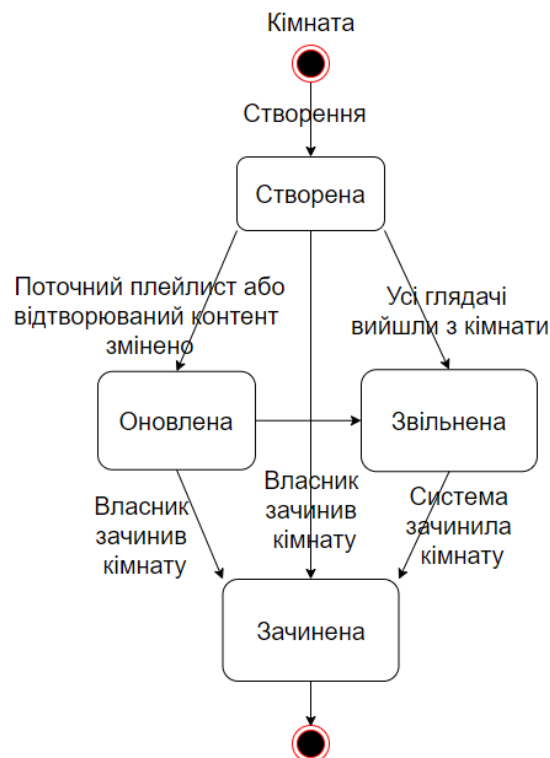


Рисунок 3.4 – Діаграма станів кімнати перегляду

3.2 Опис структури програмного забезпечення

Згідно з обраною основою архітектури на етапі проектування, проєкт поділено на субпроєкти у відповідності до обраних шарів.

3.2.1 Mediaverse.Domain

Цей модуль є програмним відображенням шару предметної області. Тут описані об'єкти домену, пов'язані правила та додані програмні інтерфейси репозиторіїв. В якості спільної функціональності тут додано клас супер-типу «Entity», який містить поле ідентифікатору сутностей, та власний клас виняткової ситуації «InformativeException», призначення якого буде описано у відповідній частині особливостей реалізації.

На рис. 3.5 продемонстровано зміст модулю Mediaverse.Domain.

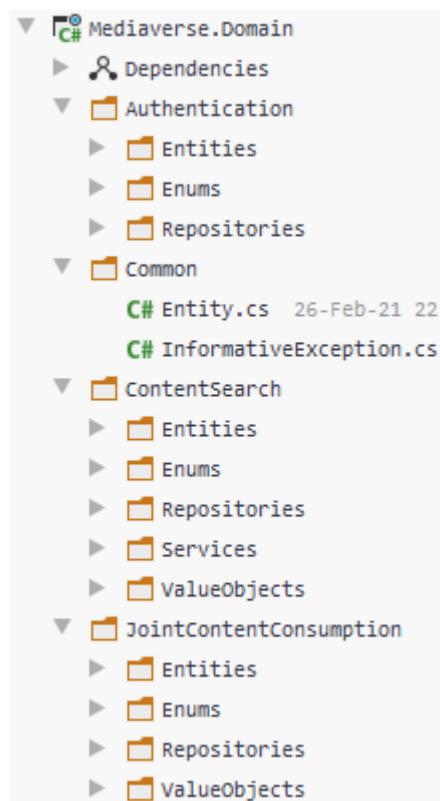


Рисунок 3.5 – Зміст модулю Mediaverse.Domain

3.2.1.1 Контекст спільного відтворення контенту

На рис. 3.6 зображено діаграму класів, які відображають поняття домену із контексту спільного відтворення контенту.

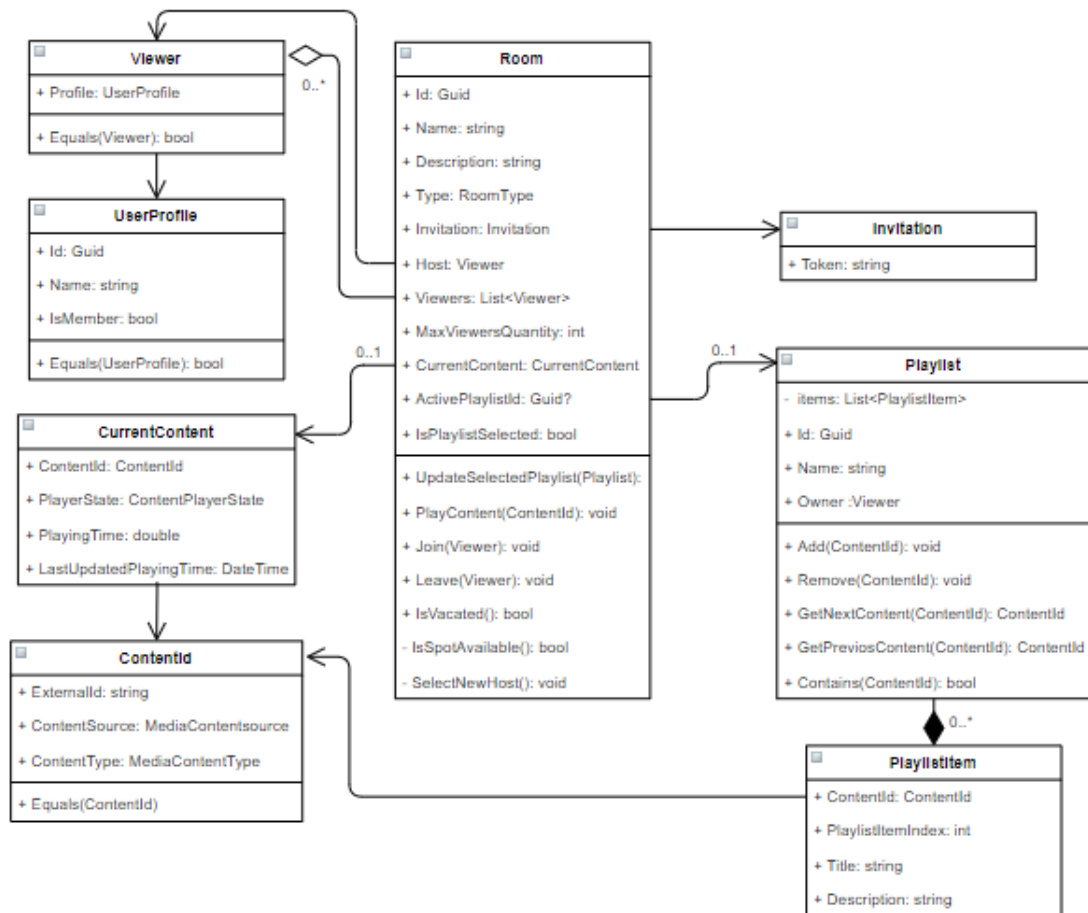


Рисунок 3.6 – Діаграма класів контексту спільного відтворення контенту

Так як контекст спільного відтворення контенту є відображенням основного субдомену предметної області, він містить значну кількість правил та логіки.

Центральною сутністю у зазначеному контексті є «кімната» та її програмне відображення «Room». Реалізаціями основних операцій цієї сутності є методи:

– «Join»;

- «Leave»;
- «UpdateSelectedPlaylist»;
- «PlayContent»;
- «SelectNewHost».

При приєднанні глядача спочатку перевіряється коректність його даних. Потім виконується перевірка на наявність місця для глядача та чи не було додано глядача раніше. Якщо глядач та поточний стан відповідає вимогам, його додають до інших глядачів кімнати.

Нижче наведено фрагмент коду, що реалізує метод «Join»:

```
public void Join(Viewer viewer)
{
    _ = viewer ?? throw new ArgumentNullException(nameof(viewer));

    if (!IsSpotAvailable)
    {
        throw new InformativeException("There is no spot for the viewer");
    }

    if (_viewers.Contains(viewer))
    {
        return;
    }

    _viewers.Add(viewer);
}
```

При виході користувача система має виконати базові перевірки: чи коректні дані надано та чи є переданий користувач глядачем цієї кімнати. Далі виконується перевірка: кімнату лишає звичайний глядач чи власник. Якщо це власник, то система має обрати нового за допомогою метода «SelectNewHost», інакше кімната просто видалить користувача із переліку глядачів.

Нижче наведено фрагмент коду, що реалізує метод «Leave»:

```
public void Leave(Viewer viewer)
{
    _ = viewer ?? throw new ArgumentNullException(nameof(viewer));
    if (!_viewers.Contains(viewer) && !viewer.Equals(Host))
    {

```

```

        throw new InformativeException("Viewer is not in the room");
    }

    bool isHostLeavingTheRoom = Host.Equals(viewer);

    if (isHostLeavingTheRoom)
    {
        SelectNewHost();
    }
    else
    {
        _viewers.Remove(viewer);
    }
}

```

Зміна поточного плейлиста є дещо простіше: тут перевіряється коректність даних та належність плейлиста поточному власнику кімнати.

Нижче наведено фрагмент коду, що реалізує метод «UpdateSelectedPlaylist»:

```

public void UpdateSelectedPlaylist(Playlist playlist)
{
    _ = playlist ?? throw new ArgumentNullException(nameof(playlist));
    if (!playlist.Owner.Equals(Host))
    {
        throw new InformativeException($"Playlist {playlist} does not belong
toT host {Host}");
    }

    ActivePlaylistId = playlist.Id;
}

```

Початок відтворення контенту обтяжується лише однією перевіркою на коректність наданого ідентифікатора контенту. Після чого встановлюється новий об'єкт-значення, що відповідає поточному відтворюваному у кімнаті контенту.

Нижче наведено фрагмент коду, що реалізує метод «PlayContent»:

```

public void PlayContent(ContentId contentId)
{
    _ = contentId ?? throw new ArgumentNullException(nameof(contentId));

    CurrentContent = new CurrentContent(
        contentId,
        ContentPlayerState.Paused,
        playingTime: 0,
        lastUpdatedPlayingTime: DateTime.Now);
}

```

```
}
```

Обрання нового власника кімнати виконується, коли поточний власник виходить з неї. В рамках цього методу випадковим чином обирається новий власник кімнати із її глядачів або якщо глядачів не лишилося, то у кімнати немає власника і відповідно до сценарію вона буде автоматично зачинена.

Нижче наведено фрагмент коду, що реалізує метод «SelectNewHost»:

```
private void SelectNewHost()
{
    var random = new Random();

    if (_viewers.Count > 0)
    {
        int newHostIndex = random.Next(0, _viewers.Count - 1);
        Host = _viewers[newHostIndex];

        _viewers.Remove(Host);
    }
    else
    {
        Host = null;
    }
}
```

Ще однією з головних сутностей контексту спільного відтворення контенту є «плейлист» та його програмне відображення «Playlist».

Реалізаціями основних операцій цієї сутності є методи:

- «Add» та «Remove»;
- «GetNextContent» та «GetPreviousContent».

Загалом вказані пари методів є схожими через «дзеркальність» операції, що вони реалізують, тому можна розглянути лише один із кожної пари.

Додання контенту виконується за допомогою його ідентифікатору. В рамках описуваної операції виконуються стандартні перевірки на коректність та наявність контенту у плейлісті. Якщо контент задовольняє вимоги, для

нього розраховується порядковий номер у плейлисті та ідентифікатор додається до переліку відтворення.

Нижче наведено фрагмент коду, що реалізує метод «Add»:

```
public void Add(ContentId contentId)
{
    _ = contentId ?? throw new ArgumentNullException(nameof(contentId));
    if (Contains(contentId))
    {
        throw new InformativeException("Content is added already");
    }

    int playlistItemIndex = (_items.LastOrDefault()?.PlaylistItemIndex ?? 0)
+ 1;
    _items.Add(new PlaylistItem(contentId, playlistItemIndex));
}
```

Отримання контенту дещо складніше, адже плейлист не зберігає інформацію про поточний відтворюваний контент і тому пошук наступного або попереднього елементу переліку відбувається за переданим ідентифікатором контенту. В рамках цієї операції виконується перевірка на наявність елементів у плейлисті. Якщо плейлист не є порожнім, виконується пошук наступного елементу після того, що відповідає переданому ідентифікатору. Якщо елемент наявний, він віддається у результаті виконання методу, інакше – генерується інформативне виключення про те що, початок плейлиста вже досягнуто.

Нижче наведено фрагмент коду, що реалізує метод «GetNextContent»:

```
public ContentId GetNextContent(ContentId contentId)
{
    if (!_items.Any())
    {
        throw new InformativeException("Playlist is empty");
    }

    PlaylistItem playlistItem = null;
    if (contentId != null)
    {
        playlistItem = _items.FirstOrDefault(i
        i.ContentId.Equals(contentId))
        ?? throw new InvalidOperationException(
            "Specified content does not belong to the
            playlist");
    }
}
```

```

var nextPlaylistItem = _items
    .FirstOrDefault(i =>
        i.PlaylistItemIndex > (playlistItem?.PlaylistItemIndex ?? 0));

if (nextPlaylistItem == null)
{
    throw new InformativeException("The end of the playlist is reached
already");
}

return nextPlaylistItem.ContentId;
}

```

Слід також зазначити, що використовуваний ідентифікатор контенту складається з ідентифікатора на зовнішньому медіаресурсі, джерела контенту та його типу. Така ідентифікація контенту є більш простою у порівнянні з введенням власного ідентифікатора і підтримки зв'язку між ними.

3.2.1.2 Контекст пошуку контенту

Контекст пошуку контенту містить поняття, необхідні для передачі даних з медіаресурсів до контексту спільного відтворення контенту. Тому класи, що є програмним відображенням зазначених понять не містять складної логіки і можуть бути описані лише даними, що зберігають.

На рис. 3.7 зображено діаграму класів, які відображають поняття домену із контексту пошуку контенту.

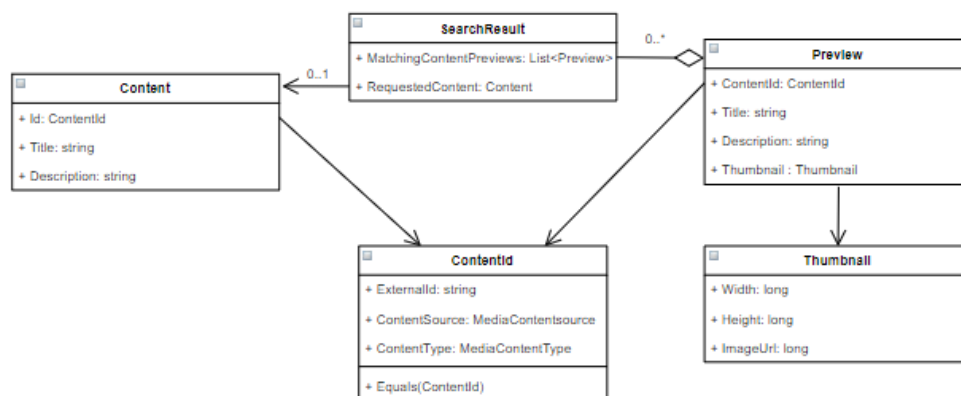


Рисунок 3.7 – Діаграма класів контексту пошуку контенту

3.2.1.3 Контекст аутентифікації

Контекст аутентифікації є досить простим, тому містить лише одну сутність – «Користувач». На рис. 3.8 зображено діаграму класів для відповідного контексту.

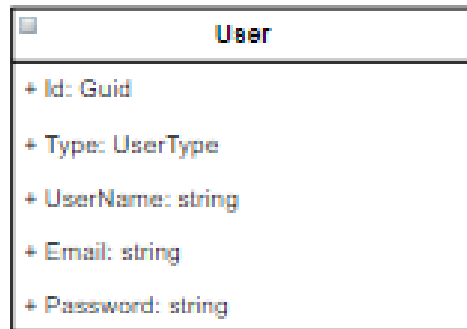


Рисунок 3.8 – Діаграма класів контексту аутентифікації

3.2.2 Mediaverse.Application

Цей модуль є програмним відображенням шару прикладної логіки. Тут реалізовано сценарії використання системи у вигляді команд та запитів. Також тут наявні класи Data Transfer Object (DTO) [14], які шар прикладної логіки повертає шару представлення. В рамках спільної функціональності тут додано інтерфейс сервісу «IdentifierProvider», реалізація якого має генерувати нове унікальне значення-ідентифікатор сутностей.

На рис. 3.9 продемонстровано зміст Mediaverse.Application.

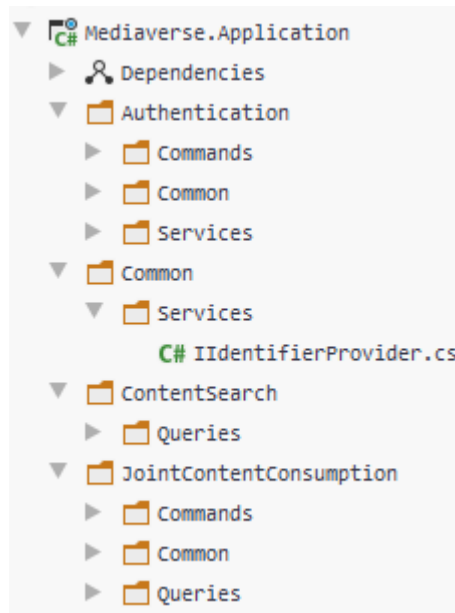


Рисунок 3.9 – Зміст Mediaverse.Application

Кожна команда і запит представляються двома власними класами. Один містить користувацьку інформацію для виконання обраного сценарію. Другий містить сам користувацький сценарій. На рис. 3.10 та 3.11 наведено приклад структури та команди відповідно.

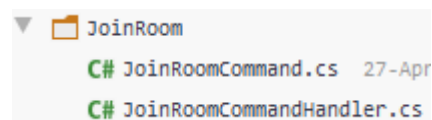


Рисунок 3.10 – Класи команди приєднання до кімнати

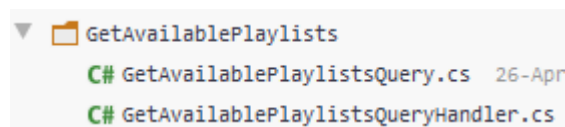


Рисунок 3.11 – Класи запиту наявних плейлистів

Загалом код обробників команд та запитів є однотипним – вони описують деякі сценарії предметної області, тобто виступають у ролі «оркестраторів» над програмними відображеннями сутностей предметної області та інфраструктурою. Структурно класи обробників команд та запитів

складаються із переліку залежностей, конструктору та методу, що виконує сценарій і повертає результат роботи.

Створення кімнати є одним із найважчих сценаріїв. Спочатку тут відкривається транзакція для контролю за узгодженістю даних. Потім отримується інформація про користувача-власника або якщо його не існує, генерується помилка. Якщо плейлист не було обрано, то в коді створюється новий і саме він буде використовуватися в кімнаті. Після цього виконується генерація ідентифікатора кімнати та на його основі створюється код запрошення. Далі створюється сама кімната і додається до БД. Якщо виконання успішне, транзакція завершується і у результаті метод повертає об'єкт класу «RoomDto».

Нижче наведено фрагмент коду, що реалізує метод «Handle» з класу обробника команди «CreateRoomCommandHandler»:

```
public async Task<RoomDto> Handle(CreateRoomCommand request,
CancellationTokentoken)
{
    try
    {
        using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        var host = await _viewerRepository.GetAsync(request.HostId,
cancellationTokentoken)
        ?? throw new ArgumentException($"Host
{request.HostId.ToString()} could not be found");

        if (request.PlaylistId == default)
        {
            Guid generatedPlaylistId = _identifierProvider.GenerateGuid();
            var playlist = new Playlist(generatedPlaylistId, "Temporary",
host) {IsTemporary = true};

            await _playlistRepository.AddAsync(playlist, cancellationTokentoken);
            request.PlaylistId = generatedPlaylistId;
        }

        Guid generatedRoomId = _identifierProvider.GenerateGuid();
        string invitationToken =
_identifierProvider.GenerateToken(generatedRoomId);

        var room = new Room(
            generatedRoomId,
            request.Name,
            host,
            request.Type,
```

```

        new Invitation(invitationToken),
        request.PlaylistId,
        request.Description);

    await _roomRepository.AddAsync(room, cancellationToken);

    transaction.Complete();

    return _mapper.Map<RoomDto>(room);
}
catch (InformativeException exception)
{
    _logger.LogError(exception, $"Could not create room with name
{request.Name} by user with Id " +
                        $"{request.HostId.ToString()}");
    throw;
}
catch (Exception exception)
{
    _logger.LogError(exception, $"Could not create room with name
{request.Name} by user with ID " +
                        $"{request.HostId.ToString()}");
    throw new InformativeException("Could not create room. Please
retry");
}
}

```

Виконання запитів простіше, бо не потребує виконання бізнес-логіки та не вимагає транзакції через відсутність модифікації даних.

Приклад методу «Handle» обробника запиту на отримання даних кімнати наведено у фрагменті нижче:

```

public async Task<RoomDto> Handle(GetRoomQuery request, CancellationToken
cancellationToken)
{
    try
    {
        var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
        ?? throw new InformativeException("Room could not be found");

        return _mapper.Map<RoomDto>(room);
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not get room {request.RoomId}");
        throw new InformativeException("Could not get room. Please retry");
    }
}

```

3.2.3 Mediaverse.Web

Цей модуль є програмним відображенням шару представлення. Тут реалізовано усі необхідні компоненти для роботи вебзастосунку: класи-контролери, які відповідають розділу предметної області за обмеженими контекстами; спеціальні файли для відображення інформації та класи, що необхідні для роботи програми.

Слід зазначити, що клас «Startup» є «точкою розширення» можливостей вебзастосунку. Саме тут виконується конфігурація ІоС контейнерів [15], налаштування конвеєру обробників запитів та т.п.

На рис. 3.12 продемонстровано зміст модулю Mediaverse.Web.

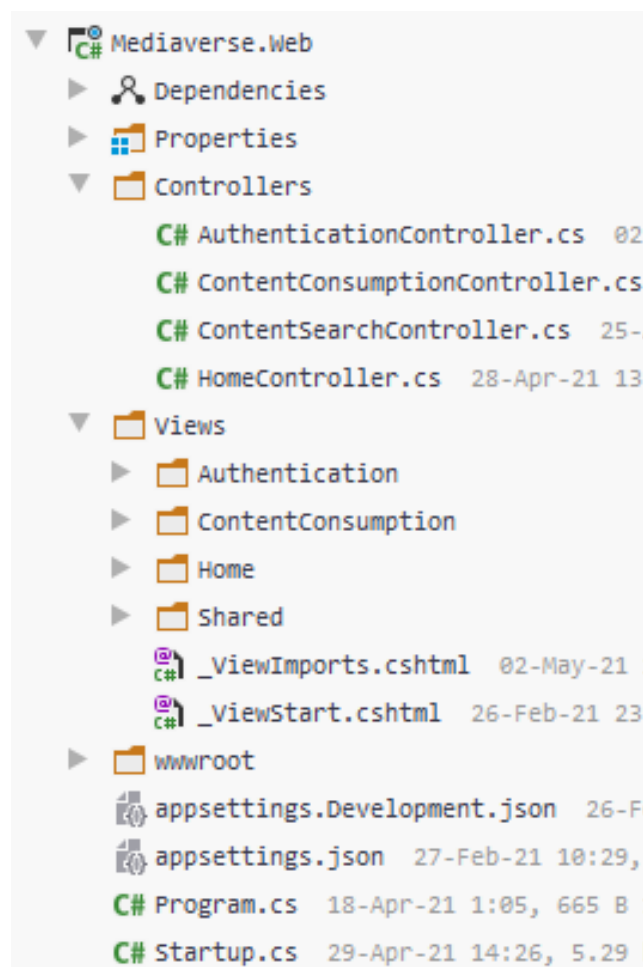


Рисунок 3.12 – Зміст модулю Mediaverse.Web

Основними класами тут є контролери, що отримують запити від клієнтської частини застосунку та передають на опрацювання до відповідних класів запитів та команд.

Наведений нижче фрагмент коду ілюструє структуру більшості методів в класах контролерах. Тут створюється або отримується команда або запит та передається до класу медіатора, який запустить відповідний обробник. Після виконання сценарію будуть отримано ідентифікатори глядачів кімнати та на їх клієнтські частини буде відправлено відповідну подію:

```
[HttpPost]
public async Task<ActionResult> DeletePlaylist(Guid roomId, CancellationToken
cancellation_token)
{
    try
    {
        var command = new DeletePlaylistCommand {MemberId =
User.GetCurrentUserId(), RoomId = roomId};
        var affectedViewerIds = await _mediator.Send(command,
cancellation_token);

        await _service.SendEventAsync(
            "playlist_updated",
            cancellation_token,
            client
            affectedViewerIds.Contains(client.User.GetCurrentUserId()));
        return Ok();
    }
    catch (Exception exception)
    {
        return BadRequest(new {message = exception.Message});
    }
}
```

3.2.4 Mediaverse.Infrastructure

Цей модуль є програмним відображенням шару використовуваної інфраструктури. Тут реалізовано оголошені на рівні предметної області репозиторії, сервіси, зазначено конфігурації перетворень. Також, через те, що саме на цьому рівні відбувається комунікація із БД, тут містяться міграції, клас конфігурації моделі даних та класи DTO для репозиторіїв.

На рис. 3.13 продемонстровано зміст модулю Mediaverse.Infrastructure.

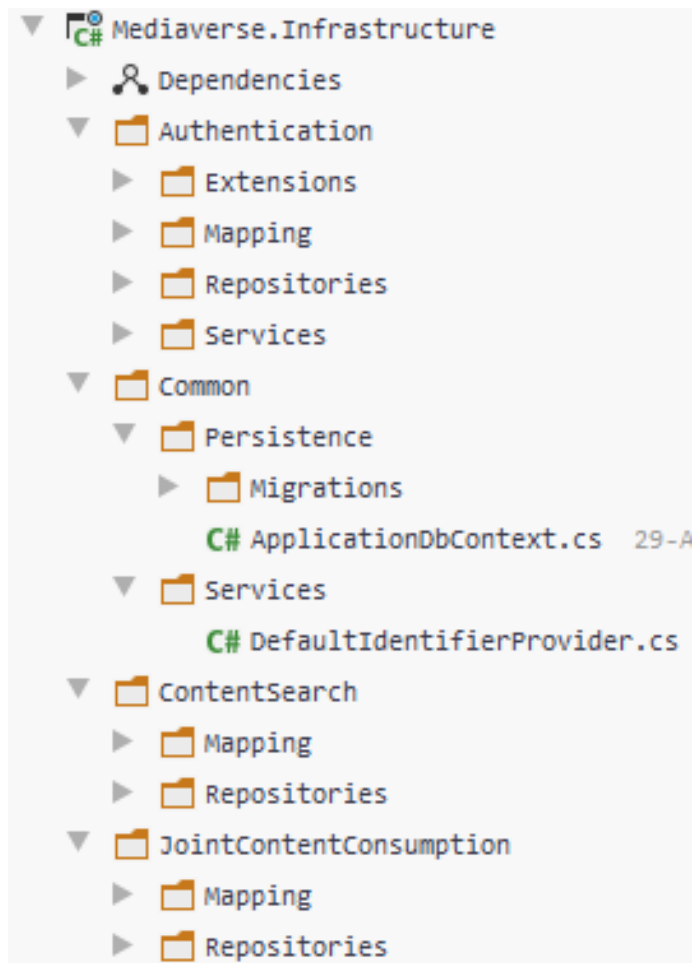


Рисунок 3.13 – Зміст модулю Mediaverse.Infrastructure

3.2.5 Mediaverse.Domain.Tests

Цей модуль є допоміжним та містить класи автоматичних тестів для класів із Mediaverse.Domain.

На рис. 3.14 продемонстровано зміст модулю Mediaverse.Infrastructure.



Рисунок 3.14 – Зміст модулю Mediaverse.Domain.Tests

3.3 Особливості реалізації

3.3.1 Аутентифікація

Аутентифікація є досить складним процесом, що має серйозно контролюватися при розробці програмного забезпечення і вимагає значних зусиль при впровадженні та тестуванні. Але дуже часто цей процес є стандартним і може вимагати лише невеликої конфігурації або розширення під специфічні потреби розроблюваної програми. Тому деякі компанії разом із програмною платформою постачають і розширювані механізми аутентифікації. Завдяки цьому було прийнято рішення скористатися механізмами ASP.NET Core Identity [16], створеної розробниками використовуваної програмної платформи, що надало більше часу на розробку програмних компонентів, які є конкурентними перевагами продукту.

Наведений нижче фрагмент коду ілюструє використання описаного механізму аутентифікації. У прикладі якщо результат аутентифікації є неуспішним, генерується інформативна помилка для користувача:

```
var result = await _signInManager.PasswordSignInAsync(
    login,
    request.Password,
    isPersistent: false,
    lockoutOnFailure: false);

if (!result.Succeeded)
{
```

```
        throw new InformativeException("Email or password is incorrect");
    }
```

3.3.2 Синхронізація даних між глядачами

Для синхронізації відтворення контенту та змін у кімнаті перегляду між користувачами з'являється необхідність у спеціальному механізмі, який при виникненні деякої події, що потребує синхронізації, повинен був сповістити клієнтські частини інших глядачів, щоб ті у свою чергу відповідно до події відреагували та оновили свій стан.

Для досягнення поставленої мети було обрано механізм Server-Sent Events (SSE) [17] – односпрямованим протоколом для стрімінгу подій [17]. Протокол передає повідомлення через активне HTTP-з'єднання [17]. Він зручний тим, що не вимагає знань, необхідний для роботи із альтернативним варіантом – сокетом. Крім того, сокет також вимагає TCP-з'єднання та пов'язаної додаткової інфраструктури, що у випадку розроблюваної системи просто було б накладними витратами ресурсів.

Сповіщення клієнтських частин було інтегровано в опрацювання користувацьких сценаріїв таким чином, що при успішному їх виконанні отримувався перелік користувачів, що мають бути сповіщені про зміни та завдяки реалізації SSE події відправлялися до необхідних користувачів.

Слід зазначити, що не дивлячись на помірну простоту SSE у порівнянні з роботою із сокетом, обраний механізм також вимагає певної додаткової інфраструктури. Проте, завдяки наявності потужних програмних реалізацій SSE, зазначену інфраструктурну функціональність було інтегровано із готового розповсюдженого рішення Lib.AspNetCore.ServerSentEvents [18].

Наведений нижче фрагмент демонструє використання механізму SSE. Спочатку у результаті виконання команди отримується перелік ідентифікаторів глядачів кімнати, яких має бути повідомлено про відповідну подію:

```

var affectedViewerIds = await _mediator.Send(command);
await _service.SendEventAsync(
    command.State,
    cancellation.Token,
    client =>
affectedViewerIds.Contains(client.User.GetCurrentUserId()));

```

На клієнтській частині відтворення та реагування на події відбувається за допомогою керування спеціальним віджетом медіаресурсу, що надає контент. Для цього завантажуються спеціальна бібліотека та створюється новий відеопрогравач:

```

<script type="text/javascript">
    var player;
    function createPlayer() {
        window.YT.ready(function() {
            player = new YT.Player('iframeContainer', {
                width: '100%',
                height: '100%',
                videoId: '@Model.Id.ExternalId',
                events: {
                    'onStateChange': onPlayerStateChange,
                    'onReady': onPlayerReady
                }
            });
        });
    }

    $(document).ready(function() {
        $.getScript("https://www.youtube.com/iframe_api", function() {
            createPlayer();
        });
    });

    function onPlayerStateChange(event) {
        if (event.data === YT.PlayerState.PLAYING) {
            onContentPlaybackStarted(player.getCurrentTime());
        }
    }

```

```

        else if (event.data === YT.PlayerState.PAUSED) {
            onContentPlaybackPaused(player.getCurrentTime());
        }
    }

    function onPlayerReady(event) {
        player.seekTo(@Model.Player.PlayingTime, true)
        if ('@Model.Player.State' === "Playing") {
            startPlayback();
        }
    }

    function startPlayback() {
        if (player.state !== YT.PlayerState.PLAYING){
            player.playVideo();
        }
    }

    function pausePlayback() {
        if (player.state !== YT.PlayerState.PAUSED){
            player.pauseVideo();
        }
    }
</script>

```

3.3.3 Узгодженість даних

Як було зазначено на етапі проєктування програмного забезпечення, однією з найважливіших архітектурних характеристик для розроблюваної системи є узгодженість даних. В класичному варіанті обраного архітектурного підходу для досягнення цієї характеристики використовуються поняття транзакційної узгодженості [19] – для контролю даних одного агрегату та одного користувацького сценарію, а при необхідності оновлення декількох агрегатів або автоматичному виконанні декількох користувацьких сценаріїв використовується кінцева узгодженість [19].

При розробці застосунку було вирішено використовувати лише транзакційну узгодженість через те, що кінцева узгодженість вимагає подієвої моделі і не гарантує виконання необхідних сценаріїв одразу, а лише із плином часу. Крім того подієва модель для остаточної консистентності вимагає механізму для збереження подій, який ускладнює розроблювану систему та значно сповільнює розробку. Також розподілені транзакції вимагають механізми компенсацій для випадків, коли певні сценарії не змогли виконатися і необхідно відновити узгодженість даних для уже успішно опрацьованих частин.

В розроблюваній системі узгодженість контролюється за допомогою механізмів платформи доступних через простір імен `System.Transactions` [20]. Тобто виконання кожного користувацького сценарію було обернено у `TransactionScope` [21]. Якщо виконання якоїсь частини сценарію не є успішним, система відкатує транзакцію та відображає помилку виконання операції.

3.3.5 Інтеграція з медіаресурсами

Для отримання даних з медіаресурсів було необхідно створити механізм інтеграції із наданими API. При цьому наявна вимога до розширення кількості медіаресурсів у майбутньому. Тому було вирішено запровадити абстракцію у вигляді класу-фасаду `ContentRepository`, що надавав контент у вигляді необхідному для розроблюваної системи, однак у свою чергу використовував класи-репозиторії, що реалізовували взаємодію із необхідними медіаресурсами.

Зазначене рішення надало гнучкість при інтеграції додаткових медіаресурсів.

Наведений нижче фрагмент коду демонструє методи класу-репозиторію для взаємодією із API YouTube.

Цей метод виконує пошуку за ключовими словами і повертає результат у вигляді переліку прев'ю:

```
public async Task<SearchResult> SearchForVideosByKeywords(string keywords,
CancellationTokен cancellationTokен)
{
    var searchRequest = _youTubeService.Search.List(SearchParts);
    searchRequest.Q = keywords;
    searchRequest.Type = SearchType;
    searchRequest.MaxResults = MaxSearchResultQuantity;
    searchRequest.VideoEmbeddable =
SearchResource.ListRequest.VideoEmbeddableEnum.True__;

    var searchResult = await searchRequest.ExecuteAsync(cancellationTokен);
    return _mapper.Map<SearchResult>(searchResult);
}
```

Цей метод реалізує пошук конкретного контенту і повертає специфічні відомості про нього:

```
public async Task<SearchResult> SearchForVideoById(string videoId,
CancellationTokен cancellationTokен)
{
    var searchRequest = _youTubeService.Videos.List(RetrieveByIdParts);
    searchRequest.Id = videoId;

    var searchResult = await searchRequest.ExecuteAsync(cancellationTokен);
    return _mapper.Map<SearchResult>(searchResult);
}
```

3.3.6 Кешування контенту

Через велике навантаження на медіаресурси, з яких отримується контент, деякі з них надають доступ до своїх API за квотами, тобто обмежують кількість запитів, які можна відправляти до їх програмного забезпечення. Для покращення використання квот було вирішено запровадити механізм кешування контенту.

Механізм кешування було інтегровано в пошук контенту на медіаресурсах за посиланням. Таким чином, при отриманні необхідних даних, система зберігала їх до власного джерела даних. Якщо користувачі

виконують пошук контенту, що було закешовано, то система не робить запит до API медіаресурсів, а отримує інформацію з власного джерела.

Нижче наведено фрагмент, що демонструє модифікований пошук контенту з отриманням закешованого контенту:

```
var cachedContent = await
_applicationDbContext.CachedContent.FirstOrDefaultAsync(cc => cc.ExternalId
== queryString && cc.ContentSource == source, cancellationToken);

if (cachedContent != null)
{
    searchResult = new SearchResult {RequestedContent =
_mapper.Map<Content>(cachedContent)};
}
else
{
    searchResult = await
_youtubeRepository.SearchForVideoById(queryString, cancellationToken);

    if (searchResult.RequestedContent != null)
    {
        await CacheContent(searchResult.RequestedContent,
cancellationToken);
    }
}
```

3.3.7 Оброблення помилок

Серед вимог до розроблюваного програмного забезпечення було також зазначено якісну обробку помилок із інформативними повідомленнями для користувача.

Для задоволення цієї вимоги до спільної функціональності шару предметної області було додано власний клас виняткової ситуації `InformativeException` для інформативних помилок, які можна надати користувачу. Відстеження та фіксація помилок, що виникли було реалізовано на рівні користувацьких сценаріїв. «Власні помилки» лише фіксуються і передаються на вищий рівень для опрацювання і відображення користувачу. Невідомі виняткові ситуації фіксуються та на вищий рівень передається власний об'єкт `InformativeException`, який повідомляє користувача, що при опрацюванні його запиту щось пішло не так.

На вищому рівні – в шарі представлення – текст отриманих помилок передається на клієнтську частину у вигляді помилки «BadRequest», яка там опрацьовується та відображається користувачу.

Такий підхід є досить простим і надійним для впровадження.

3.3.8 DTO для репозиторіїв

Хоча об'єктна та реляційні моделі досить схожі, тим не менш, вони все ж відрізняються типом зв'язків, розміщенням поведінки тощо. Крім того, при проєктуванні та програмній реалізації «багатих» моделей предметної області перетворення між зазначеними моделями стає досить складним. Також суттєву роль грають обмеження та вимоги, що висуває технологія доступу до даних.

Один із найпростіших варіантів вирішення цієї проблеми – введення шару класів DTO, сенс яких – просто містити дані, що зберігаються в БД. Після отримання даних з БД, на основі об'єктів цих DTO можна будувати об'єкти класів домену.

3.4 Висновки за розділом 3

На етапі розробки програми було успішно реалізовано систему відповідно до обраних архітектурних рішень та із врахуванням функціональних вимог. Зважаючи на специфіку та розмір розроблюваної системи, виникла необхідність певної адаптації загальних підходів, але це не змінило глобального напрямку впровадження. Розроблена архітектура показала серйозну стійкість до змін та розширення. Крім цього вона повністю задовольняє вимоги узгодженості даних та повного відображення концепцій предметної області.

У відповідному розділі було описано загальний алгоритм взаємодії з програмою, описано структуру та особливості реалізації розробленого програмного забезпечення.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення програми

Розроблене програмне забезпечення першочергово призначено для надання можливості одночасного споживання контенту спільнотою людей.

Функції, що реалізує система:

- реєстрація та аутентифікація;
- створення та видалення кімнат;
- створення та видалення плейлистів;
- додання та видалення контенту з плейлиста;
- приєднання та вихід із кімнат;
- синхронне відтворення контенту для спільноти людей;
- пошук контенту на медіаресурсах;
- відтворення контенту з плейлиста або з пошуку;
- анонімне використання сервісу.

4.2 Умови виконання програми

4.2.1 Апаратні вимоги програми до серверу

Для розгортання та виконання розробленого програмного забезпечення є необхідним персональний комп'ютер або сервер із наступними характеристиками:

- процесор з архітектурою x86 або x64;
- від 2-х Гб оперативної пам'яті;
- від 2-х Гб вільного місця для збереження файлів програми;
- наявність доступу до мережі Інтернет;
- встановлено одну з операційних систем: Windows 7 SP1+, Windows Server 2008 R2+, Mac OS X 10.11, Ubuntu 14.04+ або Red Hat Enterprise Linux

7, встановлено SDK та виконуванні файли .NET Core 3.1.

4.2.2 Апаратні вимоги програми до клієнту

Для використання можливостей вебзастосунку є необхідним пристрій із наступними характеристиками:

- від 2-х Гб оперативної пам'яті;
- наявність сучасного веббравзера із підтримкою HTML 5 та функції postMessage;
- наявність доступу до мережі Інтернет.

4.3 Робота з програмою

Взаємодія користувача із застосунком починається із головної сторінки, де йому запропоновано реєстрацію, аутентифікацію або використання сервісу без облікового запису.

Головну сторінку вебсайту продемонстровано на рис. 4.1.

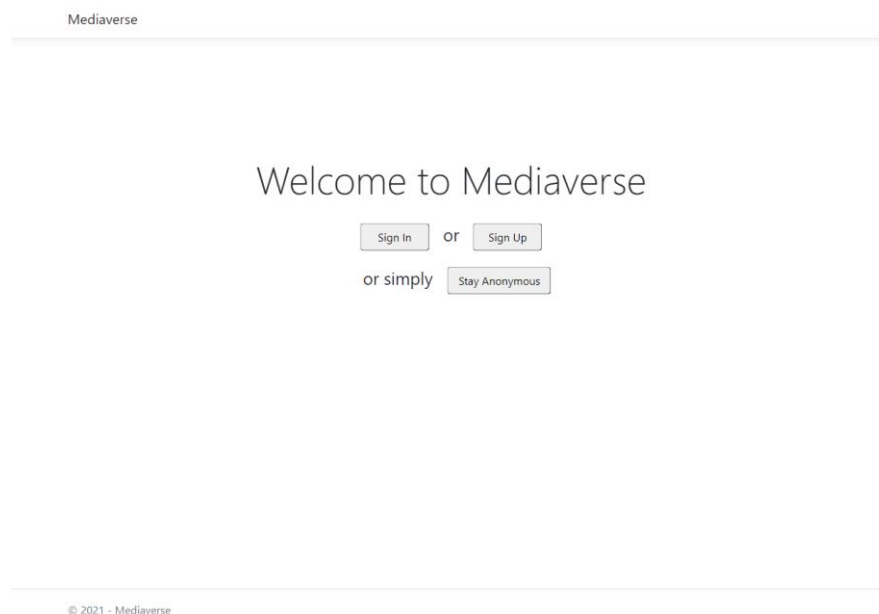
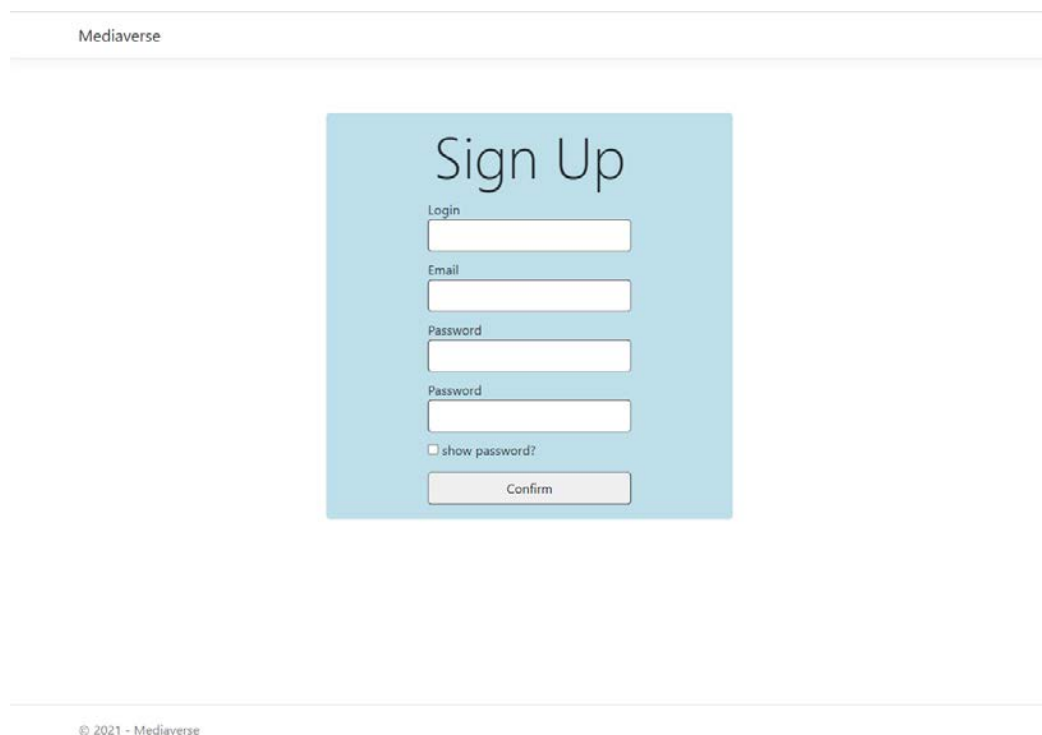


Рисунок 4.1 – Головна сторінка застосунку

Для реєстрації у системі користувач повинен вказати унікальний логін, електронну поштову адресу, пароль та його підтвердження. Після вказання необхідної інформації користувач має натиснути кнопку «Confirm». Якщо реєстрація успішна, система перенаправить користувача на сторінку керування кімнатами, інакше – відобразить повідомлення про помилку із деталями для користувача.

Сторінку реєстрації наведено на рис. 4.2.



Mediaverse

Sign Up

Login

Email

Password

Password

☐ show password?

Confirm

© 2021 - Mediaverse

Рисунок 4.2 – Сторінка реєстрації

Для аутентифікації у системі користувач має вказати логін або електронну поштову адресу та пароль від свого облікового запису. Після вказання необхідної інформації користувач має натиснути кнопку «Confirm». Якщо аутентифікація успішна, система перенаправить користувача на сторінку керування кімнатами, інакше – відобразить повідомлення про помилку із деталями для користувача.

Сторінку аутентифікації наведено на рис. 4.3.

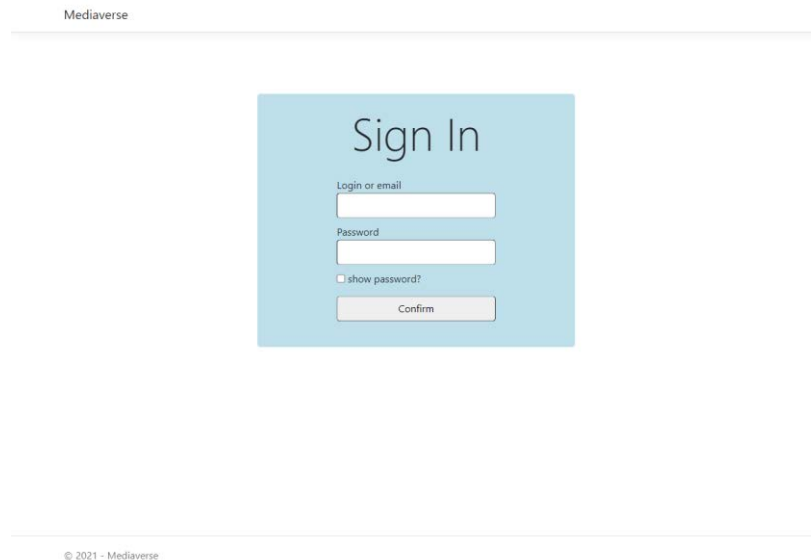


Рисунок 4.3 – Сторінка реєстрації

Сторінка керування кімнатами надає можливість створення нової кімнати або долучення до існуючої за спеціальним поширюваним кодом доступу або пошуком у секції відкритих кімнат. Для долучення до відкритої кімнати необхідно натиснути на відповідну панель.

Слід також зазначити, що відповідно до вимог до програмного забезпечення було реалізовано можливість приєднання до кімнати за посиланням.

Сторінку керування кімнатами продемонстровано на рис. 4.4.

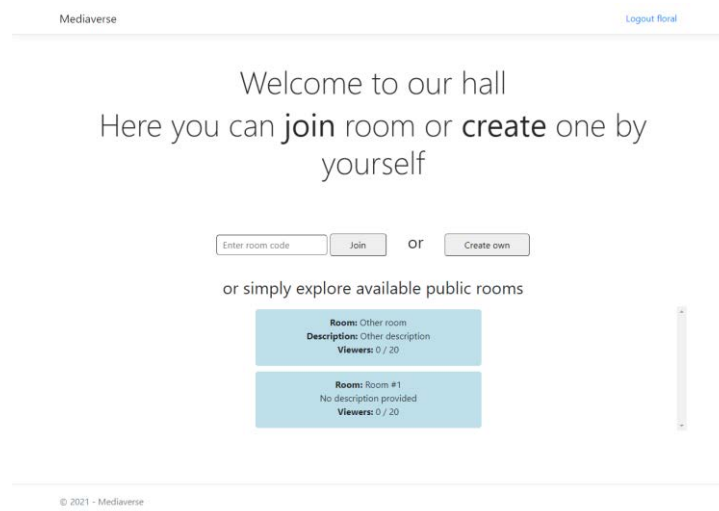
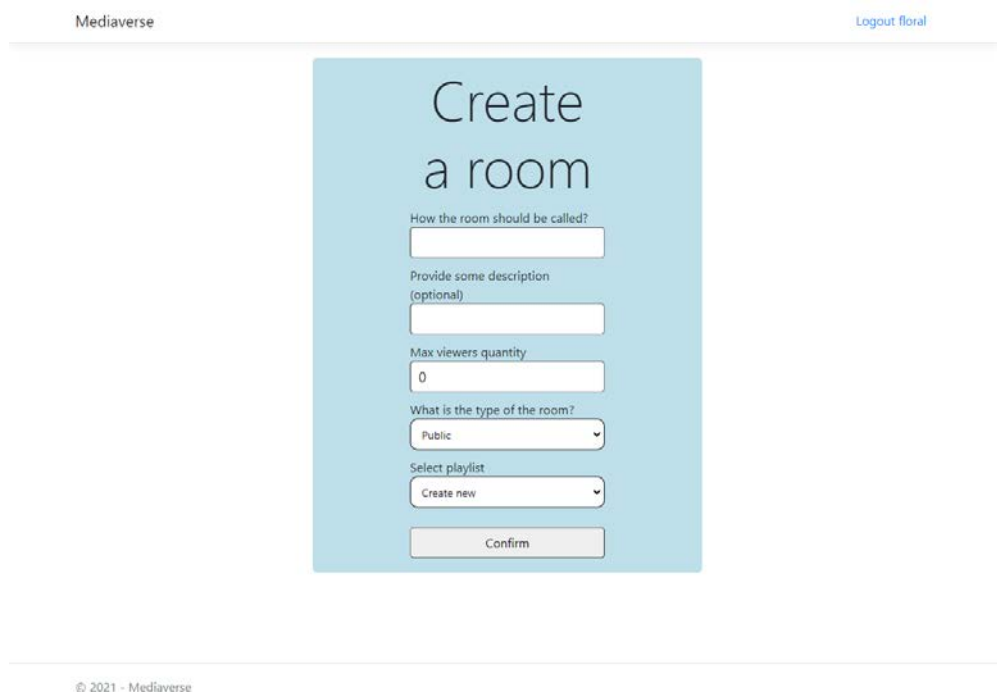


Рисунок 4.4 – Сторінка керування кімнатами

Для створення власної кімнати користувач має вказати назву кімнати, максимальну кількість глядачів, тип кімнати та обрати існуючий плейлист або створення нового. Також користувач може вказати невеликий опис, але це не є обов'язковим. Після вказання необхідної інформації користувач має натиснути кнопку «Confirm». Після успішного створення кімнати перегляду, як і після успішного приєднання, система перенаправляє користувача на сторінку кімнати.

Сторінку створення кімнати продемонстровано на рис. 4.5.



The screenshot shows a web interface for creating a room. At the top, there is a header with 'Mediaverse' on the left and 'Logout floral' on the right. The main content is a light blue box titled 'Create a room'. Inside this box, there are several input fields and dropdown menus: 'How the room should be called?' (text input), 'Provide some description (optional)' (text input), 'Max viewers quantity' (text input with '0' entered), 'What is the type of the room?' (dropdown menu with 'Public' selected), and 'Select playlist' (dropdown menu with 'Create new' selected). At the bottom of the form is a 'Confirm' button. Below the form box, there is a footer with '© 2021 - Mediaverse'.

Рисунок 4.5 – Сторінка створення кімнати

На сторінці кімнати користувач отримує можливості пошуку контенту, додання та видалення контенту із обраного плейлиста, зміни або видалення активного плейлиста та керування відтворення контентом. При створенні нової кімнати система відображає повідомлення, що контент для відтворення не обрано. Для того, щоб почати, користувач повинен через пошук знайти необхідний контент, почати відтворення або додати до активного плейлисту та перейти до наступного елементу у переліку показу.

Сторінку кімнати після її створення продемонстровано на рис. 4.6.

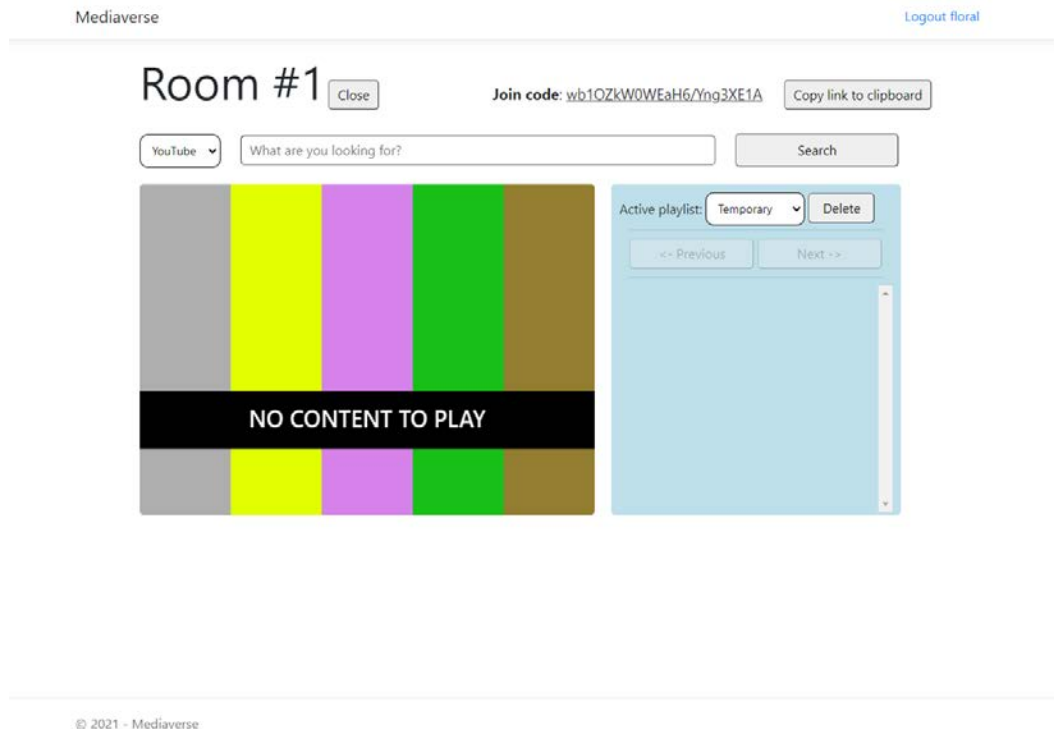


Рисунок 4.6 – Сторінка нової кімнати

Після виконання пошуку за ключовими словами знизу сторінки – у відповідній секції – будуть зображені прев'ю знайдених результатів. Як зазначалося вище, обраний контент можна додати до активного плейлиста або відразу почати перегляд натиснувши на зображення.

Після виконання пошуку за посиланням у тій же секції буде відображатися результат, але лише один, що відповідає запитуваному контенту.

Результат пошуку контенту за ключовим словом «Hello» продемонстровано на рис. 4.7.

Результат додавання контенту до плейлиста та початку відтворення із пошуку продемонстровано на рис. 4.8.

Here what we have found:

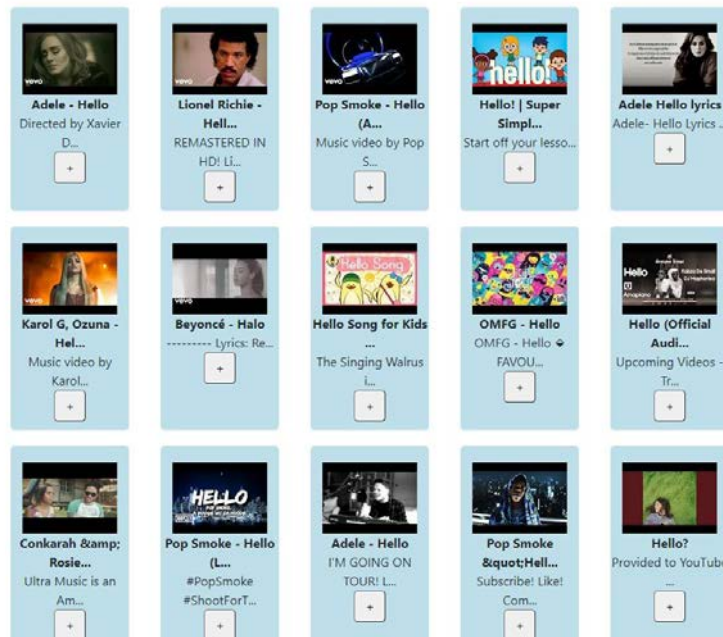


Рисунок 4.7 – Результати пошуку за ключовим словом

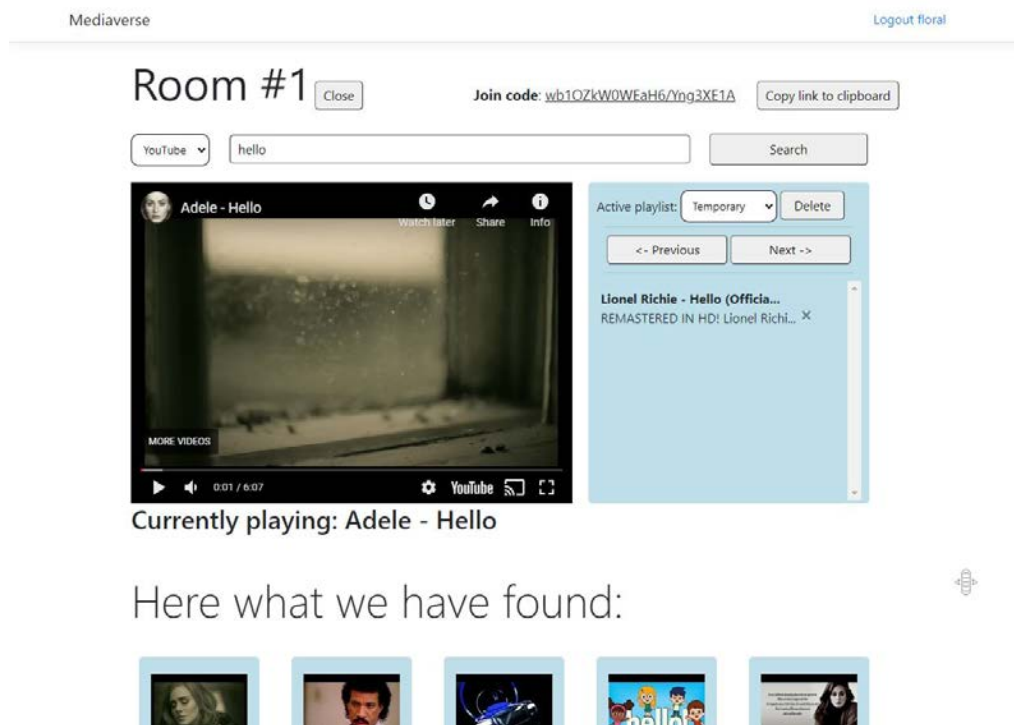


Рисунок 4.8 – Результати додавання контенту до плейлиста

4.4 Тестування програми

Для перевірки коректності виконання програмного відображення правил предметної області та контролю за відсутністю регресійних програмних помилок у розробленому програмному забезпеченні виконувалося автоматичне тестування. Воно виконувалося за допомогою модульних тестів, створених за допомогою потужного фреймворку для створення модульних тестів xUnit [22].

Автоматичне тестування відбувалося наступним чином: створюється необхідний доменний об'єкт, функціональність якого буде тестуватися, викликається потрібний метод та перевіряються результати виконання. Якщо тест пройшов успішно, функціональність виконується коректно.

На рис. 4.9 продемонстровано приклад розробленого автоматичного тесту.

```
[Theory]
[InlineData(nameof(GoodRoomsData))]
// floraldenis
public void Room_join(
    string roomName,
    Viewer host,
    RoomType type,
    Invitation invitation)
{
    var actualViewer = new Viewer(
        new UserProfile(
            id: Guid.NewGuid(),
            name: "someusername",
            isMember: false));

    var room = new Room(
        id: Guid.NewGuid(),
        roomName,
        host,
        type,
        invitation,
        playlistId: Guid.NewGuid());

    room.Join(actualViewer);

    Assert.False(room.IsVacated());
    Assert.NotEmpty(room.Viewers);
    Assert.Equal(actualViewer, room.Viewers.First());
    Assert.NotEqual(actualViewer, room.Host);
}
```

Рисунок 4.9 – Модульний тест для перевірки приєднання до кімнати

В результаті виконання роботи більшість критичних ділянок предметної області були покриті автоматичними тестами. На рис. 4.10 наведено результати автоматизованого тестування.

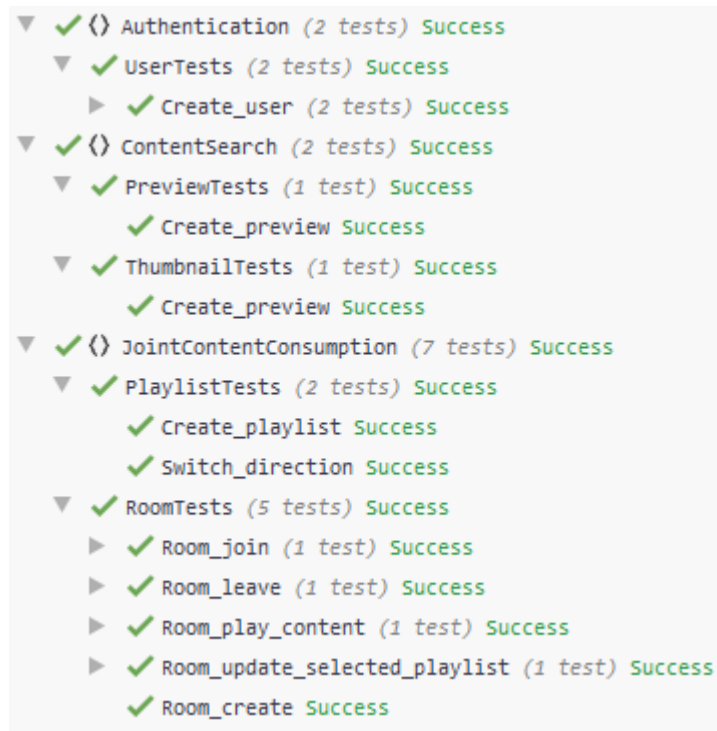


Рисунок 4.10 – Результати автоматизованого тестування сутностей домену

Для перевірки коректності роботи користувацького інтерфейсу та користувацьких сценаріїв описаних в прикладному шарі програми виконувалося ручне тестування.

4.5 Порівняння результатів

За результатами розробки, експлуатації та тестування програмного забезпечення спільного відтворення медіаресурсів можна порівняти власне рішення із існуючими на ринку.

У табл. 4.1 приведено порівняння розробленої програми із існуючими рішеннями спільного відтворення медіаресурсів.

Таблиця 4.1 – Порівняння програмних рішень проблеми

Рішення Критерій	Watch2gether	Twoseven	Kast	Mediaverse
Велика кількість медіаресурсів	+	+-	+-	-
Наявність відкритих кімнат	-	-	-	+
Пошук за ключовими словами та посиланням	+	+-	+	+
Можливість використання без реєстрації	+	-	-	+
Можливість відтворення контенту з пристрою	-	+	-	-
Наявність власної відеотеки	+	-	+	-
Відображення програмних помилок зручне для користувача	-	-	+	+

Порівняти архітектуру та відповідність необхідним характеристикам існуючих рішень не є можливим через відсутність коду програм-конкурентів або загального опису обраних рішень у вільному доступі.

4.6 Висновки за розділом 4

Після експлуатації та тестування розробленого програмного забезпечення на відповідному етапі було проаналізовано відповідність системи функціональним вимогам та відповідно до цього порівняно її із існуючими рішеннями на ринку. В результаті порівняння можна сказати, що створена програма не є абсолютним лідером, проте володіє деякими важливими перевагами по відношенню до конкурентів, має високий потенціал ідеї, здатність та стійкість до розширення функціональності у майбутньому завдяки обраних архітектурним рішенням.

У самому розділі було вказано призначення, умови виконання програмного забезпечення та було детально описано особливості взаємодії із системою.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано предметну область спільного відтворення контенту, виявлено основні поняття та існуючі рішення проблеми.

Базуючись на результатах аналізу домену та висунутих функціональних вимогах, було спроектовано систему на основі багатосарової архітектури із використанням підходів «предметно-орієнтоване проектування» та «CQRS», що надало значну гнучкість при розробці програми, високу структурованість та концентрацію знань про домен. Крім цього для розробленого коду було створено множину автоматизованих модульних тестів, що контролюють коректність та актуальність програмних відображень правил предметної області.

Розроблене програмне забезпечення відповідає висунутим функціональним вимогам та здатне виконувати наступні операції:

- створення та видалення кімнат перегляду;
- приєднання до кімнат перегляду за кодом, посиланням або через пошук відкритих кімнат;
- створення, видалення та редагування вмісту переліків відтворення (плейлистів);
- синхронізоване відтворення контенту для спільноти людей;
- пошук контенту на медіаресурсах за ключовими словами або за посиланням;
- відтворення контенту із плейлиста або одразу із результатів пошуку;
- аутентифікація та реєстрація;
- можливість використання сервісу без проходження реєстрації.

В результаті тестування та експлуатації порівняння створеної програми із існуючими рішеннями зазначеної проблеми показало, що не

зважаючи на відсутність деякої некритичної функціональності, програма володіє перевагами та виступає активним конкурентом існуючим аналогам.

В рамках вдосконалення розробленого програмного забезпечення можна вказати наступні функціональні можливості:

- відтворення власного контенту з пристрою;
- розширення можливостей пошуку контенту за відтворенням його фрагменту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Content [Electronic resource]. – Access mode : <https://dictionary.cambridge.org/dictionary/english/content>.
2. Watch2gether [Electronic resource]. – Access mode : <https://w2g.tv/>.
3. Twoseven [Electronic resource]. – Access mode : <https://twoseven.xyz/>.
4. Kast [Electronic resource]. – Access mode : <https://kast.live/>.
5. NLayer Architecture [Electronic resource]. – Access mode : <https://aspnetboilerplate.com/Pages/Documents/NLayer-Architecture>.
6. Мартин Фаулер. Архитектура корпоративных программных приложений [Текст] : Пер. с англ. / М. Фаулер.— М. : Вильямс, 2006. – 544 с.
7. Еванс, Е. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем [Текст] : Пер. с англ. / Е. Еванс.— М. : Вильямс, 2011. – 448 с.
8. CQRS [Electronic resource]. – Access mode : <https://martinfowler.com/bliki/CQRS.html>.
9. A tour of the C# language [Electronic resource]. – Access mode : <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>.
10. What is Java? Definition, Meaning & Features of Java Platforms [Electronic resource]. – Access mode : <https://www.guru99.com/java-platform.html>.
11. .NET Core Overview [Electronic resource]. – Access mode : <https://www.tutorialsteacher.com/core/dotnet-core>.
12. ASP.NET Core Overview [Electronic resource]. – Access mode : <https://www.tutorialsteacher.com/core/aspnet-core-introduction>.
13. Entity Framework Core [Electronic resource]. – Access mode : <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>.
14. Data Transfer Object [Electronic resource]. – Access mode : <https://martinfowler.com/eaCatalog/dataTransferObject.html>.

15. IoC Container [Electronic resource]. – Access mode : <https://www.tutorialsteacher.com/ioc/ioc-container>.
16. Introduction to Identity on ASP.NET Core [Electronic resource]. – Access mode : <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>.
17. Server-Sent Events (SSE) support for ASP.NET Core [Electronic resource]. – Access mode : <https://www.tpeczek.com/2017/02/server-sent-events-sse-support-for.html>.
18. Lib.AspNetCore.ServerSentEvents [Electronic resource]. – Access mode : <https://github.com/tpeczek/Lib.AspNetCore.ServerSentEvents>.
19. Domain Events and Eventual Consistency [Electronic resource]. – Access mode : <https://www.infoq.com/news/2015/09/domain-events-consistency/>.
20. System.Transactions Namespace [Electronic resource]. – Access mode : <https://docs.microsoft.com/en-us/dotnet/api/system.transactions?view=netcore-3.1>.
21. TransactionScope Class [Electronic resource]. Access mode : <https://docs.microsoft.com/en-us/dotnet/api/system.transactions.transactionscope?view=netcore-3.1>.
22. About xUnit.net [Electronic resource]. – Access mode : <https://xunit.net/>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.1 Назва програми

Назва програми – «Mediaverse».

A.1.2 Призначення та область застосування

Програма призначена для спільного відтворення медіаресурсів для одночасного споживання контенту спільнотою людей.

Система реалізована у вигляді вебсайту.

Областю застосування є сфера онлайн-розваг для спільного проведення часу спільнотою людей.

A.2 Підстави для розробки

Підставою для розробки є завдання на диплому роботу на тему «Програмне забезпечення спільного відтворення медіаресурсів».

Актуальність роботи полягає в дослідженні архітектурних особливостей системи, що розроблюється та створені відповідного програмного забезпечення на основі результатів досліджень.

A.3 Призначення розробки

Програма призначена для спільного відтворення медіаресурсів для одночасного споживання контенту спільнотою людей.

A.4 Вимоги до програми

A.4.1 Вимоги до функціональних характеристик

Програмний продукт повинен забезпечувати виконання наступних функцій:

- створення та видалення кімнат перегляду;
- створення, редагування та видалення плейлистів;
- пошук контент на медіаресурсах;
- синхронне відтворення контенту;
- приєднання та вихід із кімнати;
- реєстрація та аутентифікація;
- використання сервісу без облікового запису.

A.4.2 Вимоги до надійності

Програма призначена для роботи на серверній частині, через що має забезпечувати безперебійну роботу протягом усього часу, що працює сервер.

A.4.3 Вимоги до технічних та програмних засобів

Для розгортання та виконання розробленого програмного забезпечення є необхідним персональний комп'ютер або сервер із наступними характеристиками:

- процесор з архітектурою x86 або x64;
- від 2-х Гб оперативної пам'яті;
- від 2-х Гб вільного місця для збереження файлів програми;
- наявність доступу до мережі Інтернет;

- встановлено одну з операційних систем: Windows 7 SP1+, Windows Server 2008 R2+, Mac OS X 10.11, Ubuntu 14.04+ або Red Hat Enterprise Linux 7;

- встановлено SDK та виконувальні файли .NET Core 3.1.

A.5 Вимоги до програмної документації

Склад програмної документації до програмного продукту:

- технічне завдання;
- пояснювальна записка.

A.6 Стадії і етапи розробки

Розробка має бути проведена в наступні стадії:

- створення та узгодження технічного завдання;
- аналіз предметної області та існуючих рішень;
- проектування архітектури системи;
- проектування БД;
- вибір технічних засобів;
- розробка програмного забезпечення.

A.7 Порядок контролю та приймання

Принципи контролю виконання роботи визначаються календарним планом, розташованим у завданні на дипломну кваліфікаційну роботу, де вказано назви етапів, строки їх виконання та результат.

ДОДАТОК Б
Текст програми

Б.1 Текст файла SignInCommand.cs

```
using System.ComponentModel.DataAnnotations;
using MediatR;
using Mediaverse.Application.Authentication.Common.Dtos;

namespace Mediaverse.Application.Authentication.Commands.SignIn
{
    public class SignInCommand : IRequest
    {
        public string LoginOrEmail { get; set; }
        public string Password { get; set; }
    }
}
```

Б.2 Текст файла SignInCommandHandler.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using MediatR;
using Mediaverse.Application.Authentication.Services;
using Mediaverse.Domain.Authentication.Entities;
using Mediaverse.Domain.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.Authentication.Commands.SignIn
{
    public class SignInCommandHandler : IRequestHandler<SignInCommand>
    {
        private readonly SignInManager<User> _signInManager;
        private readonly UserManager<User> _userManager;

        private readonly IEmailService _emailService;
        private readonly ILogger<SignInCommandHandler> _logger;

        public SignInCommandHandler(
            SignInManager<User> signInManager,
            UserManager<User> userManager,
            IEmailService emailService,
            ILogger<SignInCommandHandler> logger)
        {
            _signInManager = signInManager;
            _userManager = userManager;
            _emailService = emailService;
            _logger = logger;
        }

        public async Task<Unit> Handle(SignInCommand request,
            CancellationToken cancellationToken)
        {
            try
            {

```

```

        using var transaction = new
        TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        if (string.IsNullOrEmpty(request.LoginOrEmail)
            || string.IsNullOrEmpty(request.Password))
        {
            throw new InformativeException("Please provide valid
login/email and password");
        }

        string login = request.LoginOrEmail;
        if (!_emailService.IsValidEmail(request.LoginOrEmail))
        {
            var user = await
            _userManager.FindByEmailAsync(request.LoginOrEmail);
            login = user?.UserName ?? login;
        }

        var result = await _signInManager.PasswordSignInAsync(
            login,
            request.Password,
            isPersistent: false,
            lockoutOnFailure: false);

        if (!result.Succeeded)
        {
            throw new InformativeException("Email or password is
incorrect");
        }

        transaction.Complete();

        return Unit.Value;
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Could not sign in
{request.LoginOrEmail}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not sign in
{request.LoginOrEmail}");
        throw new InformativeException("Something went wrong. Please
retry");
    }
}
}

```

Б.3 Текст файла SignOutCommand.cs

```

using MediatR;

namespace Mediaverse.Application.Authentication.Commands.SignOut
{
    public class SignOutCommand : IRequest

```

```
    { }
}
```

Б.4 Текст файла SignOutCommandHandler.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using MediatR;
using Mediaverse.Domain.Authentication.Entities;
using Mediaverse.Domain.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.Authentication.Commands.SignOut
{
    public class SignOutCommandHandler : IRequestHandler<SignOutCommand>
    {
        private readonly SignInManager<User> _signInManager;
        private readonly ILogger<SignOutCommandHandler> _logger;

        public SignOutCommandHandler(
            SignInManager<User> signInManager,
            ILogger<SignOutCommandHandler> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        public async Task<Unit> Handle(SignOutCommand request,
            CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
                TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                await _signInManager.SignOutAsync();

                transaction.Complete();

                return Unit.Value;
            }
            catch (Exception exception)
            {
                _logger.LogError(exception, "Could not log out user");
                throw new InformativeException("Log Out attempt failed.
                Please retry.");
            }
        }
    }
}
```

Б.5 Текст файла SignUpCommand.cs

```
using System.ComponentModel.DataAnnotations;
using MediatR;
using Mediaverse.Application.Authentication.Common.Dtos;

namespace Mediaverse.Application.Authentication.Commands.SignUp
{
    public class SignUpCommand : IRequest<UserDto>
    {
        public string Login { get; set; }
        public string Password { get; set; }
        public string PasswordConfirmation { get; set; }
        public string Email { get; set; }
    }
}
```

Б.6 Текст файла SignUpCommandHandler.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.Authentication.Common.Dtos;
using Mediaverse.Domain.Authentication.Entities;
using Mediaverse.Domain.Authentication.Enums;
using Mediaverse.Domain.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.Authentication.Commands.SignUp
{
    public class SignUpCommandHandler : IRequestHandler<SignUpCommand, UserDto>
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;
        private readonly ILogger<SignUpCommandHandler> _logger;
        private readonly IMapper _mapper;

        public SignUpCommandHandler(
            UserManager<User> userManager,
            SignInManager<User> signInManager,
            ILogger<SignUpCommandHandler> logger,
            IMapper mapper)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
            _mapper = mapper;
        }
    }
}
```

```

        public async Task<UserDto> Handle(SignUpCommand request,
CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                if (string.IsNullOrEmpty(request.Email)
                    || string.IsNullOrEmpty(request.Login)
                    || string.IsNullOrEmpty(request.Password)
                    || string.IsNullOrEmpty(request.PasswordConfirmation))
                {
                    throw new InformativeException("Please provide valid
email, login and " +
                                                    "password with
confirmation");
                }

                if (request.Password != request.PasswordConfirmation)
                {
                    throw new InformativeException("Password and Confirmation
does not match");
                }

                var user = new User(UserType.Member)
                {
                    UserName = request.Login,
                    Email = request.Email,
                };

                var result = await _userManager.CreateAsync(user,
request.Password);
                if (result.Succeeded)
                {
                    await _signInManager.SignInAsync(user, isPersistent:
false);
                }
                else
                {
                    var errorMessages = result.Errors.Select(e =>
e.Description);
                    throw new InformativeException(string.Join("\n",
errorMessages));
                }

                transaction.Complete();

                return _mapper.Map<UserDto>(user);
            }
            catch (InformativeException exception)
            {
                _logger.LogError(exception, $"Could not sign up user
{request.Email}");
                throw;
            }
            catch (Exception exception)
            {
                _logger.LogError(exception, $"Could not sign up user
{request.Email}");
            }
        }
    }
}

```

```

        throw new InformativeException("Could not sign up user.
Please retry");
    }
}
}
}

```

Б.7 Текст файла SignUpAnonymousCommand.cs

```

using MediatR;
using Mediaverse.Application.Authentication.Common.Dtos;

namespace Mediaverse.Application.Authentication.Commands.SignUpAnonymous
{
    public class SignUpAnonymousCommand : IRequest
    { }
}

```

Б.8 Текст файла SignUpAnonymousCommandHandler.cs

```

using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.Authentication.Common.Dtos;
using Mediaverse.Application.Authentication.Services;
using Mediaverse.Application.Common.Services;
using Mediaverse.Domain.Authentication.Entities;
using Mediaverse.Domain.Authentication.Enums;
using Mediaverse.Domain.Authentication.Repositories;
using Mediaverse.Domain.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.Authentication.Commands.SignUpAnonymous
{
    public class SignUpAnonymousCommandHandler :
    IRequestHandler<SignUpAnonymousCommand>
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;
        private readonly INameGenerator _nameGenerator;
        private readonly ILogger<SignUpAnonymousCommandHandler> _logger;
        private readonly IMapper _mapper;

        public SignUpAnonymousCommandHandler(
            UserManager<User> userManager,
            SignInManager<User> signInManager,
            INameGenerator nameGenerator,
            ILogger<SignUpAnonymousCommandHandler> logger,
            IMapper mapper)
        {

```

```

    {
        _userManager = userManager;
        _signInManager = signInManager;
        _nameGenerator = nameGenerator;
        _logger = logger;
        _mapper = mapper;
    }

    public async Task<Unit> Handle(SignUpAnonymousCommand request,
CancellationToken cancellationToken)
    {
        try
        {
            using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

            string generatedPassword =
_nameGenerator.GenerateAnonymousPassword();
            var user = new User(UserType.Anonymous) {UserName =
_nameGenerator.GenerateAnonymousName()};

            var result = await _userManager.CreateAsync(user,
generatedPassword);
            if (result.Succeeded)
            {
                await _signInManager.SignInAsync(user, isPersistent:
false);
            }
            else
            {
                var errorMessages = result.Errors.Select(e =>
e.Description);
                throw new InvalidOperationException(string.Join("\n",
errorMessages));
            }

            transaction.Complete();

            return Unit.Value;
        }
        catch (InformativeException exception)
        {
            _logger.LogError(exception, "Could not sign up as
anonymous");
            throw;
        }
        catch (Exception exception)
        {
            _logger.LogError(exception, "Could not sign up as
anonymous");
            throw new InformativeException("Could not sign up as
anonymous. Please retry");
        }
    }
}

```

Б.9 Текст файла UserDto.cs

```
using System;

namespace Mediaverse.Application.Authentication.Common.Dtos
{
    public class UserDto
    {
        public Guid Id { get; set; }
        public string Nickname { get; set; }
    }
}
```

Б.10 Текст файла IEmailService.cs

```
namespace Mediaverse.Application.Authentication.Services
{
    public interface IEmailService
    {
        bool IsValidEmail(string email);
    }
}
```

Б.11 Текст файла INameGenerator.cs

```
using System;

namespace Mediaverse.Application.Authentication.Services
{
    public interface INameGenerator
    {
        string GenerateAnonymousName();
        string GenerateAnonymousPassword();
    }
}
```

Б.12 Текст файла IIdentifierProvider.cs

```
using System;

namespace Mediaverse.Application.Common.Services
{
    public interface IIdentifierProvider
    {
        Guid GenerateGuid();
        string GenerateToken(Guid guid);
    }
}
```

Б.13 Текст файла ContentIdDto.cs

```
using Mediaverse.Domain.ContentSearch.Enums;

namespace
Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos
{
    public class ContentIdDto
    {
        public string ExternalId { get; set; }
        public MediaContentSource ContentSource { get; set; }
        public MediaContentType ContentType { get; set; }
    }
}
```

PreviewDto.cs:

```
namespace
Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos
{
    public class PreviewDto
    {
        public ContentIdDto ContentId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public ThumbnailDto Thumbnail { get; set; }
    }
}
```

Б.14 Текст файла SearchResultDto.cs

```
using System.Collections.Generic;

namespace
Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos
{
    public class SearchResultDto
    {
        public IList<PreviewDto> Previews { get; set; }
    }
}
```

Б.15 Текст файла ThumbnailDto.cs

```
namespace
Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos
{
    public class ThumbnailDto
    {
        public long Width { get; set; }
        public long Height { get; set; }
    }
}
```

```

        public string Url { get; set; }
    }
}

```

Б.16 Текст файла GetRelevantContentQuery.cs

```

using System.Collections.Generic;
using MediatR;
using Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos;
using Mediaverse.Domain.ContentSearch.Enums;

namespace Mediaverse.Application.ContentSearch.Queries.GetRelevantContent
{
    public class GetRelevantContentQuery : IRequest<SearchResultDto>
    {
        public MediaContentSource SelectedSource { get; set; }
        public string QueryString { get; set; }

        public override string ToString() => $"{QueryString} from {SelectedSource.ToString()}";
    }
}

```

Б.17 Текст файла GetRelevantContentQueryHandler.cs

```

using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using AutoMapper;
using MediatR;
using Mediaverse.Application.ContentSearch.Queries.GetRelevantContent.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.ContentSearch.Enums;
using Mediaverse.Domain.ContentSearch.Repositories;
using Mediaverse.Domain.ContentSearch.Services;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.ContentSearch.Queries.GetRelevantContent
{
    public class GetRelevantContentQueryHandler :
        IRequestHandler<GetRelevantContentQuery, SearchResultDto>
    {
        private readonly IContentRepository _contentRepository;
        private readonly IQueryStringProcessor _queryStringProcessor;
        private readonly ILogger<GetRelevantContentQueryHandler> _logger;
        private readonly IMapper _mapper;

        public GetRelevantContentQueryHandler(
            IContentRepository contentRepository,
            IQueryStringProcessor queryStringProcessor,
            ILogger<GetRelevantContentQueryHandler> logger,
            IMapper mapper)
        {

```

```

        _contentRepository = contentRepository;
        _queryStringProcessor = queryStringProcessor;
        _logger = logger;
        _mapper = mapper;
    }
    public async Task<SearchResultDto> Handle(GetRelevantContentQuery
request, CancellationToken cancellationToken)
    {
        try
        {
            var contentQueryType =
            _queryStringProcessor.DefineQueryStringType(request.SelectedSource,
request.QueryString);

            string queryString = contentQueryType ==
ContentQueryType.ContentId
?
            _queryStringProcessor.ExtractExternalContentIdFromUrl(request.SelectedSource,
request.QueryString)
            : request.QueryString;

            var searchResult = await
            _contentRepository.SearchForContentAsync(
                request.SelectedSource,
                contentQueryType,
                queryString,
                cancellationToken);

            return new SearchResultDto
            {
                Previews = searchResult.MatchingContentPreviews != null
                ?
                _mapper.Map<IList<PreviewDto>>(searchResult.MatchingContentPreviews)
                :
                new List<PreviewDto>{_mapper.Map<PreviewDto>(searchResult.RequestedContent)}
            };
        }
        catch (InformativeException exception)
        {
            _logger.LogError(exception, $"Could not get relevant content
for {request}");
            throw;
        }
        catch (Exception exception)
        {
            _logger.LogError(exception, $"Could not get relevant content
for {request}", exception);
            throw new InformativeException("Could not get relevant
content for your request. Please retry");
        }
    }
}

```

Б.18 Текст файла GetRelevantContentQueryHandler.cs

```

using System;
using MediatR;

```

```

using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.AddContentToPlaylist
{
    public class AddContentToPlaylistCommand : IRequest<PlaylistDto>
    {
        public Guid CurrentRoomId { get; set; }
        public ContentIdDto ContentId { get; set; }
    }
}

```

Б.19 Текст файла AddContentToPlaylistCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Mediaverse.Domain.JointContentConsumption.ValueObjects;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.AddContentToPlaylist
{
    public class AddContentToPlaylistCommandHandler :
    IRequestHandler<AddContentToPlaylistCommand, PlaylistDto>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IPlaylistRepository _playlistRepository;
        private readonly IContentRepository _contentRepository;
        private readonly ILogger<AddContentToPlaylistCommandHandler> _logger;
        private readonly IMapper _mapper;

        public AddContentToPlaylistCommandHandler(
            IRoomRepository roomRepository,
            IPlaylistRepository playlistRepository,
            IContentRepository contentRepository,
            ILogger<AddContentToPlaylistCommandHandler> logger,
            IMapper mapper)
        {
            _roomRepository = roomRepository;
            _playlistRepository = playlistRepository;
            _contentRepository = contentRepository;
            _logger = logger;
            _mapper = mapper;
        }

        public async Task<PlaylistDto> Handle(AddContentToPlaylistCommand
request, CancellationToken cancellationToken)
        {
            try
            {

```

```

        using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        var room = await
_roomRepository.GetAsync(request.CurrentRoomId, cancellationToken)
        ?? throw new ArgumentException(
            $"Room {request.CurrentRoomId.ToString()}
could not be found");

        var playlist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
        ?? throw new InvalidOperationException(
            $"Playlist
{room.ActivePlaylistId.ToString()} " +
            $"could not be found");

        var contentId = _mapper.Map<ContentId>(request.ContentId);
        await _contentRepository.GetAsync(contentId,
cancellationToken);

        playlist.Add(contentId);
        await _playlistRepository.UpdateAsync(playlist,
cancellationToken);

        transaction.Complete();

        return _mapper.Map<PlaylistDto>(playlist);
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Could not add content
{request.ContentId} to active playlist of " +
            $"room
{request.CurrentRoomId.ToString()}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not add content
{request.ContentId} to active playlist of " +
            $"room {request.CurrentRoomId.ToString()}");
        throw new InformativeException("Could not add content to the
playlist. Please retry");
    }
}
}
}

```

Б.20 Текст файла ChangeActivePlaylistCommand.cs

```

using System;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.ChangeActivePlaylist
{
    public class ChangeActivePlaylistCommand : IRequest<PlaylistDto>

```

```

    {
        public Guid RoomId { get; set; }
        public Guid PlaylistId { get; set; }
    }
}

```

Б.21 Текст файла ChangeActivePlaylistCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.ChangeActivePlaylist
{
    public class ChangeActivePlaylistCommandHandler :
    IRequestHandler<ChangeActivePlaylistCommand, PlaylistDto>
    {
        private readonly IPlaylistRepository _playlistRepository;
        private readonly IRoomRepository _roomRepository;
        private readonly ILogger<ChangeActivePlaylistCommandHandler> _logger;
        private readonly IMapper _mapper;

        public ChangeActivePlaylistCommandHandler(
            IPlaylistRepository playlistRepository,
            IRoomRepository roomRepository,
            ILogger<ChangeActivePlaylistCommandHandler> logger,
            IMapper mapper)
        {
            _playlistRepository = playlistRepository;
            _roomRepository = roomRepository;
            _logger = logger;
            _mapper = mapper;
        }

        public async Task<PlaylistDto> Handle(ChangeActivePlaylistCommand
request, CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
                    ?? throw new ArgumentException("Room could not be
found");

                var newPlaylist = await
_playlistRepository.GetAsync(request.PlaylistId, cancellationToken)

```

```

        ?? throw new ArgumentException("New
playlist could not be found");

        room.UpdateSelectedPlaylist(newPlaylist);

        if (room.IsPlaylistSelected)
        {
            var previousPlaylist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
            ?? throw new
InvalidOperationException(
                "Previous active playlist
could not be found");

            if (previousPlaylist.IsTemporary)
            {
                await
_playlistRepository.DeleteAsync(previousPlaylist.Id, cancellationToken);
            }
        }

        await _roomRepository.UpdateAsync(room, cancellationToken);

        transaction.Complete();

        return _mapper.Map<PlaylistDto>(newPlaylist);
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Could not change active
playlist in room {request.RoomId.ToString()}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not change active
playlist in room {request.RoomId.ToString()}");
        throw new InformativeException("Could not change active
playlist in room. Please retry");
    }
}
}
}

```

Б.22 Текст файла ChangeContentPlayerStateCommand.cs

```

using System;
using MediatR;

namespace
Mediaverse.Application.JointContentConsumption.Commands.ChangeContentPlayerSt
ate
{
    public class ChangeContentPlayerStateCommand : IRequest
    {
        public Guid RoomId { get; set; }
        public string State { get; set; }
        public double CurrentPlaybackTimeInSeconds { get; set; }
    }
}

```

```
    }
}
```

Б.23 Текст файла ChangeContentPlayerStateCommandHandler.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using MediatR;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Enums;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.ChangeContentPlayerSt
ate
{
    public class PlayContentCommandHandler :
    IRequestHandler<ChangeContentPlayerStateCommand>
    {
        private readonly IRoomRepository _roomRepository;

        private readonly ILogger<PlayContentCommandHandler> _logger;

        public PlayContentCommandHandler(
            IRoomRepository roomRepository,
            ILogger<PlayContentCommandHandler> logger)
        {
            _roomRepository = roomRepository;
            _logger = logger;
        }

        public async Task<Unit> Handle(ChangeContentPlayerStateCommand
request, CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
                ?? throw new InformativeException("Could not find the
room");

                room.CurrentContent.PlayerState =
ConvertFromString(request.State);
                room.CurrentContent.PlayingTime =
(long)request.CurrentPlaybackTimeInSeconds;
                room.CurrentContent.LastUpdatedPlayingTime = DateTime.Now;

                await _roomRepository.UpdateAsync(room, cancellationToken);

                transaction.Complete();

                return Unit.Value;
            }
        }
    }
}
```

```

    }
    catch (Exception exception)
    {
        _logger.LogError(exception.ToString());
        throw;
    }
}

private ContentPlayerState ConvertFromString(string state)
{
    if (state == "playing")
    {
        return ContentPlayerState.Playing;
    }
    else
    {
        return ContentPlayerState.Paused;
    }
}
}
}

```

Б.24 Текст файла CloseRoomCommand.cs

```

using System;
using MediatR;

namespace Mediaverse.Application.JointContentConsumption.Commands.CloseRoom
{
    public class CloseRoomCommand : IRequest
    {
        public Guid RoomId { get; set; }
        public Guid MemberId { get; set; }
    }
}

```

Б.25 Текст файла CloseRoomCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using MediatR;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.JointContentConsumption.Commands.CloseRoom
{
    public class CloseRoomCommandHandler : IRequestHandler<CloseRoomCommand>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IPlaylistRepository _playlistRepository;
        private readonly ILogger<CloseRoomCommandHandler> _logger;
    }
}

```

```

public CloseRoomCommandHandler(
    IRoomRepository roomRepository,
    IPlaylistRepository playlistRepository,
    ILogger<CloseRoomCommandHandler> logger)
{
    _roomRepository = roomRepository;
    _playlistRepository = playlistRepository;
    _logger = logger;
}

public async Task<Unit> Handle(CloseRoomCommand request,
CancellationToken cancellationToken)
{
    try
    {
        using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
            ?? throw new ArgumentException("Room could not be
found");

        if (room.Host.Profile.Id != request.MemberId)
        {
            throw new InformativeException("Member does not have
permission");
        }

        if (room.IsPlaylistSelected)
        {
            var activePlaylist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
                ?? throw new
ArgumentException("Active playlist could not be found");

            if (activePlaylist.IsTemporary)
            {
                await
_playlistRepository.DeleteAsync(activePlaylist.Id, cancellationToken);
            }

            await
_roomRepository.DeleteAsync(room.Id,
cancellationToken);

            transaction.Complete();

            return Unit.Value;
        }
        catch (InformativeException exception)
        {
            _logger.LogError(exception, $"Could not close room
{request.RoomId.ToString()}");
            throw;
        }
        catch (Exception exception)
        {
            _logger.LogError(exception, $"Could not close room
{request.RoomId.ToString()}");

```

```

        throw new InformativeException("Could not close room. Please
retry");
    }
}
}
}

```

Б.26 Текст файла CreateRoomCommand.cs

```

using System;
using System.Collections.Generic;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using
Mediaverse.Application.JointContentConsumption.Queries.GetPlaylist.Dtos;
using Mediaverse.Domain.JointContentConsumption.Enums;

namespace Mediaverse.Application.JointContentConsumption.Commands.CreateRoom
{
    public class CreateRoomCommand : IRequest<RoomDto>
    {
        public Guid HostId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public RoomType Type { get; set; }
        public Guid PlaylistId { get; set; }
        public int MaxViewersQuantity { get; set; }

        public IList<SelectablePlaylistDto> AvailablePlaylists { get; set; }
    }
}

```

Б.27 Текст файла CreateRoomCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.Common.Services;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Entities;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.JointContentConsumption.Commands.CreateRoom
{
    public class CreateRoomCommandHandler :
    IRequestHandler<CreateRoomCommand, RoomDto>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IViewerRepository _viewerRepository;
    }
}

```

```

private readonly IPlaylistRepository _playlistRepository;
private readonly IIdentifierProvider _identifierProvider;
private readonly ILogger<CreateRoomCommandHandler> _logger;
private readonly IMapper _mapper;

public CreateRoomCommandHandler(
    IRoomRepository roomRepository,
    IViewerRepository viewerRepository,
    IPlaylistRepository playlistRepository,
    IIdentifierProvider identifierProvider,
    ILogger<CreateRoomCommandHandler> logger,
    IMapper mapper)
{
    _roomRepository = roomRepository;
    _viewerRepository = viewerRepository;
    _playlistRepository = playlistRepository;
    _identifierProvider = identifierProvider;
    _logger = logger;
    _mapper = mapper;
}

public async Task<RoomDto> Handle(CreateRoomCommand request,
CancellationTokens cancellationTokens)
{
    try
    {
        using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        var host = await _viewerRepository.GetAsync(request.HostId,
cancellationTokens)
            ?? throw new ArgumentException($"Host
{request.HostId.ToString()} could not be found");

        if (request.PlaylistId == default)
        {
            Guid generatedPlaylistId =
_identifierProvider.GenerateGuid();
            var playlist = new Playlist(generatedPlaylistId,
"Temporary", host) {IsTemporary = true};

            await _playlistRepository.AddAsync(playlist,
cancellationTokens);
            request.PlaylistId = generatedPlaylistId;
        }

        Guid generatedRoomId = _identifierProvider.GenerateGuid();
        string invitationToken =
_identifierProvider.GenerateToken(generatedRoomId);

        var room = new Room(
            generatedRoomId,
            request.Name,
            host,
            request.Type,
            new Invitation(invitationToken),
            request.PlaylistId,
            request.Description);

        await _roomRepository.AddAsync(room, cancellationTokens);
    }
}

```

```

        transaction.Complete();

        return _mapper.Map<RoomDto>(room);
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Could not create room with name
{request.Name} by user with ID " +
                                $"{request.HostId.ToString()}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not create room with name
{request.Name} by user with ID " +
                                $"{request.HostId.ToString()}");
        throw new InformativeException("Could not create room. Please
retry");
    }
}
}
}
}

```

Б.28 Текст файла DeletePlaylistCommand.cs

```

using System;
using System.Collections.Generic;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.DeletePlaylist
{
    public class DeletePlaylistCommand : IRequest<IList<PlaylistDto>>
    {
        public Guid MemberId { get; set; }
        public Guid RoomId { get; set; }
    }
}

```

Б.29 Текст файла DeletePlaylistCommandHandler.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.Common.Services;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Entities;

```

```

using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.DeletePlaylist
{
    public class DeletePlaylistCommandHandler :
    IRequestHandler<DeletePlaylistCommand, IList<PlaylistDto>>
    {
        private readonly IPlaylistRepository _playlistRepository;
        private readonly IViewerRepository _viewerRepository;
        private readonly IRoomRepository _roomRepository;
        private readonly IIdentifierProvider _identifierProvider;
        private readonly ILogger<DeletePlaylistCommandHandler> _logger;
        private readonly IMapper _mapper;

        public DeletePlaylistCommandHandler(
            IPlaylistRepository playlistRepository,
            IViewerRepository viewerRepository,
            IRoomRepository roomRepository,
            IIdentifierProvider identifierProvider,
            ILogger<DeletePlaylistCommandHandler> logger,
            IMapper mapper)
        {
            _playlistRepository = playlistRepository;
            _viewerRepository = viewerRepository;
            _roomRepository = roomRepository;
            _identifierProvider = identifierProvider;
            _logger = logger;
            _mapper = mapper;
        }

        public async Task<IList<PlaylistDto>> Handle(DeletePlaylistCommand
request, CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
                    ?? throw new ArgumentException("Room could not be
found");

                var playlist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
                    ?? throw new ArgumentException("Playlist could
not be found");

                var viewer = await
_viewerRepository.GetAsync(request.MemberId, cancellationToken)
                    ?? throw new ArgumentException("Viewer could not
be found");

                if (!playlist.Owner.Equals(viewer))
                {
                    throw new InformativeException("Playlist does not belong
to you");
                }
            }
        }
    }
}

```

```

        await _playlistRepository.DeleteAsync(playlist.Id,
cancellationTokens);

        var remainingPlaylists = await
_playlistRepository.GetAllByViewerAsync(request.MemberId, cancellationTokens);
        if (remainingPlaylists == null || remainingPlaylists.Count ==
0)
        {
            var newPlaylistId = _identifierProvider.GenerateGuid();
            var newPlaylist = new Playlist(newPlaylistId,
"Temporary", viewer);

            await _playlistRepository.AddAsync(newPlaylist,
cancellationTokens);

            room.UpdateSelectedPlaylist(newPlaylist);
        }
        else
        {
            room.UpdateSelectedPlaylist(remainingPlaylists.First());
        }

        await _roomRepository.UpdateAsync(room, cancellationTokens);
        transaction.Complete();

        return _mapper.Map<IList<PlaylistDto>>(remainingPlaylists);
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Could not delete playlist
{request.RoomId.ToString()} by member " +
request.MemberId.ToString());
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not delete playlist
{request.RoomId.ToString()} by member " +
request.MemberId.ToString());
        throw new InformativeException("Could not delete playlist.
Please retry");
    }
}
}
}

```

Б.30 Текст файла JoinRoomCommand.cs

```

using System;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace Mediaverse.Application.JointContentConsumption.Commands.JoinRoom
{
    public class JoinRoomCommand : IRequest<RoomDto>
    {

```

```

        public Guid ViewerId { get; set; }
        public string RoomToken { get; set; }
    }
}

```

Б.31 Текст файла JoinRoomCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.JointContentConsumption.Commands.JoinRoom
{
    public class JoinRoomCommandHandler : IRequestHandler<JoinRoomCommand,
RoomDto>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IViewerRepository _viewerRepository;
        private readonly ILogger<JoinRoomCommandHandler> _logger;
        private readonly IMapper _mapper;

        public JoinRoomCommandHandler(
            IRoomRepository roomRepository,
            IViewerRepository viewerRepository,
            ILogger<JoinRoomCommandHandler> logger,
            IMapper mapper)
        {
            _roomRepository = roomRepository;
            _viewerRepository = viewerRepository;
            _logger = logger;
            _mapper = mapper;
        }

        public async Task<RoomDto> Handle(JoinRoomCommand request,
CancellationTokens cancellationTokens)
        {
            try
            {
                using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var viewer = await
_viewerRepository.GetAsync(request.ViewerId, cancellationTokens)
                ?? throw new ArgumentException("Viewer could not
be found");

                var room = await _roomRepository.GetAsync(request.RoomToken,
cancellationTokens)
                ?? throw new InformativeException("Room could not
be found");
            }
        }
    }
}

```

```

        room.Join(viewer);

        await _roomRepository.UpdateAsync(room, cancellationToken);

        transaction.Complete();

        return _mapper.Map<RoomDto>(room);
    }
    catch (InformativeException exception)
    {
        _logger.LogError(exception, $"Viewer
{request.ViewerId.ToString()} could not join the " +
                                $"room {request.RoomToken}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Viewer
{request.ViewerId.ToString()} could not join the " +
                                $"room {request.RoomToken}");
        throw new InformativeException("Failed to join the room.
Please retry");
    }
}
}
}

```

Б.32 Текст файла LeaveRoomCommand.cs

```

using System;
using MediatR;

namespace Mediaverse.Application.JointContentConsumption.Commands.LeaveRoom
{
    public class LeaveRoomCommand : IRequest
    {
        public Guid ViewerId { get; set; }
        public Guid RoomId { get; set; }
    }
}

```

Б.33 Текст файла LeaveRoomCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Microsoft.Extensions.Logging;

namespace Mediaverse.Application.JointContentConsumption.Commands.LeaveRoom

```

```

{
    public class LeaveRoomCommandHandler : IRequestHandler<LeaveRoomCommand>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IViewerRepository _viewerRepository;
        private readonly ILogger<LeaveRoomCommandHandler> _logger;
        private readonly IMapper _mapper;

        public LeaveRoomCommandHandler(
            IRoomRepository roomRepository,
            IViewerRepository viewerRepository,
            ILogger<LeaveRoomCommandHandler> logger,
            IMapper mapper)
        {
            _roomRepository = roomRepository;
            _viewerRepository = viewerRepository;
            _logger = logger;
            _mapper = mapper;
        }

        public async Task<Unit> Handle(LeaveRoomCommand request,
            CancellationToken cancellationToken)
        {
            try
            {
                using var transaction = new
                TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var viewer = await
                _viewerRepository.GetAsync(request.ViewerId, cancellationToken)
                ?? throw new ArgumentException("Viewer could not
                be found");

                var room = await _roomRepository.GetAsync(request.RoomId,
                cancellationToken)
                ?? throw new ArgumentException("Room could not be
                found");

                room.Leave(viewer);

                if (room.IsVacated())
                {
                    await
                    _roomRepository.DeleteAsync(room.Id,
                    cancellationToken);
                }
                else
                {
                    await
                    _roomRepository.UpdateAsync(room,
                    cancellationToken);
                }

                transaction.Complete();

                return Unit.Value;
            }
            catch (InformativeException exception)
            {
                _logger.LogError(exception, $"Viewer
                {request.ViewerId.ToString()} could not leave " +
                $"the
                {request.RoomId.ToString()}");
            }
        }
    }
}

```

```

        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Viewer
{request.ViewerId.ToString()} could not leave " +
        $"the room {request.RoomId.ToString()}");
        throw new InformativeException("Failed to leave the room.
Please retry");
    }
}
}
}
}
}

```

Б.34 Текст файла PlaySpecificContentCommand.cs

```

using System;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.PlaySpecificContent
{
    public class PlaySpecificContentCommand : IRequest
    {
        public Guid RoomId { get; set; }
        public ContentIdDto ContentId { get; set; }
    }
}

```

Б.35 Текст файла PlaySpecificContentCommandHandler.cs

```

using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Mediaverse.Domain.JointContentConsumption.ValueObjects;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.PlaySpecificContent
{
    public class PlaySpecificContentCommandHandler :
    IRequestHandler<PlaySpecificContentCommand>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IContentRepository _contentRepository;
        private readonly ILogger<PlaySpecificContentCommandHandler> _logger;
        private readonly IMapper _mapper;
    }
}

```

```

public PlaySpecificContentCommandHandler(
    IRoomRepository roomRepository,
    IContentRepository contentRepository,
    ILogger<PlaySpecificContentCommandHandler> logger,
    IMapper mapper)
{
    _roomRepository = roomRepository;
    _contentRepository = contentRepository;
    _logger = logger;
    _mapper = mapper;
}

public async Task<Unit> Handle(PlaySpecificContentCommand request,
CancellationToken cancellationToken)
{
    try
    {
        using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

        var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
            ?? throw new ArgumentException("Room could not be
found");

        var contentId = _mapper.Map<ContentId>(request.ContentId);
        var content = await _contentRepository.GetAsync(contentId,
cancellationToken)
            ?? throw new ArgumentException("Content could not be
found");

        room.PlayContent(contentId);

        await _roomRepository.UpdateAsync(room, cancellationToken);

        transaction.Complete();

        return Unit.Value;
    }
    catch (InformativeException exception)
    {
        _logger.LogWarning(exception, $"Could not play content
{request.ContentId} in room {request.RoomId}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogWarning(exception, $"Could not play content
{request.ContentId} in room {request.RoomId}");
        throw new InformativeException("Could not play requested
content. Please retry");
    }
}
}

```

Б.36 Текст файла RemoveContentFromPlaylistCommand.cs

```
using System;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.RemoveContentFromPlay
list
{
    public class RemoveContentFromPlaylistCommand : IRequest<PlaylistDto>
    {
        public Guid CurrentRoomId { get; set; }
        public ContentIdDto ContentId { get; set; }
    }
}
```

Б.37 Текст файла RemoveContentFromPlaylistCommandHandler.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.JointContentConsumption.Common.Dtos;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Mediaverse.Domain.JointContentConsumption.ValueObjects;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.RemoveContentFromPlay
list
{
    public class RemoveContentFromPlaylistCommandHandler :
    IRequestHandler<RemoveContentFromPlaylistCommand, PlaylistDto>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IPlaylistRepository _playlistRepository;
        private readonly ILogger<RemoveContentFromPlaylistCommandHandler>
_logger;
        private readonly IMapper _mapper;

        public RemoveContentFromPlaylistCommandHandler(
            IRoomRepository roomRepository,
            IPlaylistRepository playlistRepository,
            ILogger<RemoveContentFromPlaylistCommandHandler> logger,
            IMapper mapper)
        {
            _roomRepository = roomRepository;
            _playlistRepository = playlistRepository;
            _logger = logger;
            _mapper = mapper;
        }
    }
}
```

```

        public async Task<PlaylistDto>
        Handle(RemoveContentFromPlaylistCommand request, CancellationToken
        cancellationToken)
        {
            try
            {
                using var transaction = new
                TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

                var room = await
                _roomRepository.GetAsync(request.CurrentRoomId, cancellationToken)
                ?? throw new ArgumentException(
                    $"Room {request.CurrentRoomId.ToString()}
                could not be found");

                var activePlaylist = await
                _playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
                ?? throw new InvalidOperationException(
                    $"Playlist
                {room.ActivePlaylistId.ToString()} " +
                    "could not be found");

                var contentId = _mapper.Map<ContentId>(request.ContentId);

                if (room.CurrentContent.ContentId.Equals(contentId))
                {
                    throw new InformativeException("Could not remove
                currently playing content. " +
                    "Please change playing
                content and retry");
                }

                activePlaylist.Remove(contentId);
                await _playlistRepository.UpdateAsync(activePlaylist,
                cancellationToken);

                transaction.Complete();

                return _mapper.Map<PlaylistDto>(activePlaylist);
            }
            catch (InformativeException exception)
            {
                _logger.LogError(exception, $"Could not remove content
                {request.ContentId} from active playlist of " +
                    $"room
                {request.CurrentRoomId.ToString()}");
                throw;
            }
            catch (Exception exception)
            {
                _logger.LogError(exception, $"Could not remove content
                {request.ContentId} from active playlist of " +
                    $"room {request.CurrentRoomId.ToString()}");
                throw new InformativeException("Could not remove content from
                the playlist. Please retry");
            }
        }
    }
}

```

Б.38 Текст файла SavePlaylistCommand.cs

```
using System;
using MediatR;

namespace
Mediaverse.Application.JointContentConsumption.Commands.SavePlaylist
{
    public class SavePlaylistCommand : IRequest
    {
        public Guid RoomId { get; set; }
        public Guid ViewerId { get; set; }
    }
}
```

Б.39 Текст файла SavePlaylistCommandHandler.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using AutoMapper;
using MediatR;
using Mediaverse.Application.Common.Services;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Entities;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Mediaverse.Domain.JointContentConsumption.ValueObjects;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.SavePlaylist
{
    public class SavePlaylistCommandHandler :
    IRequestHandler<SavePlaylistCommand>
    {
        private readonly IPlaylistRepository _playlistRepository;
        private readonly IViewerRepository _viewerRepository;
        private readonly IRoomRepository _roomRepository;
        private readonly IIdentifierProvider _identifierProvider;
        private readonly ILogger<SavePlaylistCommandHandler> _logger;
        private readonly IMapper _mapper;

        public SavePlaylistCommandHandler(
            IPlaylistRepository playlistRepository,
            IViewerRepository viewerRepository,
            IRoomRepository roomRepository,
            IIdentifierProvider identifierProvider,
            ILogger<SavePlaylistCommandHandler> logger,
            IMapper mapper)
        {
            _playlistRepository = playlistRepository;
            _viewerRepository = viewerRepository;
            _roomRepository = roomRepository;
        }
    }
}
```

```

        _identifierProvider = identifierProvider;
        _logger = logger;
        _mapper = mapper;
    }

    public async Task<Unit> Handle(SavePlaylistCommand request,
CancellationToken cancellationToken)
    {
        try
        {
            using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

            var room = await _roomRepository.GetAsync(request.RoomId,
cancellationToken)
                ?? throw new ArgumentException("Room could not be
found");

            var viewer = await
_viewerRepository.GetAsync(request.ViewerId, cancellationToken)
                ?? throw new ArgumentException("Viewer could not
be found");

            if (!room.Viewers.Contains(viewer))
            {
                throw new InvalidOperationException("Viewer does not
belong to this room");
            }

            var activePlaylist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationToken)
                ?? throw new ArgumentException("Playlist
could not be found");

            if (activePlaylist.Owner == viewer)
            {
                if (!activePlaylist.IsTemporary)
                {
                    throw new InvalidOperationException("Playlist is
added already");
                }

                activePlaylist.IsTemporary = false;
            }
            else
            {
                Guid newPlaylistId = _identifierProvider.GenerateGuid();
                activePlaylist = CreatePlaylistCopy(newPlaylistId,
viewer, activePlaylist);
            }

            await _playlistRepository.AddAsync(activePlaylist,
cancellationToken);

            transaction.Complete();

            return Unit.Value;
        }
        catch (InformativeException exception)
        {

```

```

        _logger.LogError(exception, $"Could not add playlist
{request.RoomId.ToString()} " +
                                $"to host of room
{request.RoomId.ToString()}");
        throw;
    }
    catch (Exception exception)
    {
        _logger.LogError(exception, $"Could not add playlist
{request.RoomId.ToString()} " +
                                $"to host of room
{request.RoomId.ToString()}");
        throw new InformativeException("Could not add playlist.
Please retry");
    }
}

private Playlist CreatePlaylistCopy(Guid newPlaylistId, Viewer
newPlaylistOwner, Playlist sourcePlaylist) =>
    new Playlist(newPlaylistId, sourcePlaylist.Name,
newPlaylistOwner, sourcePlaylist);
}
}

```

Б.40 Текст файла ContentDto.cs

```

using Mediaverse.Application.JointContentConsumption.Common.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent.Dtos
{
    public class ContentDto
    {
        public ContentIdDto Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public ContentPlayerDto Player { get; set; }
    }
}

```

Б.41 Текст файла ContentPlayerDto.cs

```

namespace
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent.Dtos
{
    public class ContentPlayerDto
    {
        public long Width { get; set; }
        public long Height { get; set; }
        public string Html { get; set; }
        public string State { get; set; }
        public double PlayingTime { get; set; }
    }
}

```

Б.42 Текст файла SwitchContentCommand.cs

```
using System;
using MediatR;
using
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent.Dtos;
using Mediaverse.Domain.JointContentConsumption.Enums;

namespace
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent
{
    public class SwitchContentCommand : IRequest
    {
        public Guid RoomId { get; set; }
        public SwitchContentDirection Direction { get; set; }
    }
}
```

Б.43 Текст файла SwitchContentCommandHandler.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Transactions;
using MediatR;
using Mediaverse.Domain.Common;
using Mediaverse.Domain.JointContentConsumption.Enums;
using Mediaverse.Domain.JointContentConsumption.Repositories;
using Mediaverse.Domain.JointContentConsumption.ValueObjects;
using Microsoft.Extensions.Logging;

namespace
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent
{
    public class SwitchContentCommandHandler :
    IRequestHandler<SwitchContentCommand>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IPlaylistRepository _playlistRepository;
        private readonly ILogger<SwitchContentCommandHandler> _logger;

        public SwitchContentCommandHandler(
            IRoomRepository roomRepository,
            IPlaylistRepository playlistRepository,
            ILogger<SwitchContentCommandHandler> logger)
        {
            _roomRepository = roomRepository;
            _playlistRepository = playlistRepository;
            _logger = logger;
        }

        public async Task<Unit> Handle(SwitchContentCommand request,
            CancellationToken cancellation_token)
        {

```

```

        try
        {
            using var transaction = new
TransactionScope(TransactionScopeAsyncFlowOption.Enabled);

            var room = await _roomRepository.GetAsync(request.RoomId,
cancellationTokens)
                ?? throw new InvalidOperationException($"Room
{request.RoomId.ToString()} does not exist");

            var playlist = await
_playlistRepository.GetAsync(room.ActivePlaylistId.Value, cancellationTokens)
                ?? throw new
InvalidOperationException($"Playlist {room.ActivePlaylistId.ToString()} " +
$"does
not exist");

            ContentId contentId;
            if (playlist.Contains(room.CurrentContent.ContentId))
            {
                contentId = request.Direction ==
SwitchContentDirection.Next
                    ?
playlist.GetNextContent(room.CurrentContent?.ContentId)
                    :
playlist.GetPreviousContent(room.CurrentContent?.ContentId);
            }
            else
            {
                contentId = playlist.FirstOrDefault()?.ContentId;
            }

            room.CurrentContent = new CurrentContent(
                contentId,
                ContentPlayerState.Paused,
                playingTime: 0,
                lastUpdatedPlayingTime: DateTime.Now);

            await _roomRepository.UpdateAsync(room, cancellationTokens);

            transaction.Complete();

            return Unit.Value;
        }
        catch (InformativeException exception)
        {
            _logger.LogError(exception, $"Could not play next content
from active playlist of " +
                $"room
{request.RoomId.ToString()}");
            throw;
        }
        catch (Exception exception)
        {
            _logger.LogError(exception, $"Could not play next content
from active playlist of " +
                $"room {request.RoomId.ToString()}");
            throw new InformativeException("Could not play next content
from the playlist. Please retry");
        }
    }
}

```

```
    }
}
```

Б.44 Текст файла ContentIdDto.cs

```
using Mediaverse.Domain.JointContentConsumption.Enums;

namespace Mediaverse.Application.JointContentConsumption.Common.Dtos
{
    public class ContentIdDto
    {
        public string ExternalId { get; set; }
        public MediaContentSource ContentSource { get; set; }
        public MediaContentType ContentType { get; set; }
    }
}
```

Б.45 Текст файла PlaylistDto.cs

```
using System;
using System.Collections.Generic;
using
Mediaverse.Application.JointContentConsumption.Queries.GetPlaylist.Dtos;

namespace Mediaverse.Application.JointContentConsumption.Common.Dtos
{
    public class PlaylistDto
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public IList<PlaylistItemDto> Items { get; set; }

        public IList<SelectablePlaylistDto> AvailablePlaylists { get; set; }
    }
}
```

Б.46 Текст файла PlaylistItemDto.cs

```
namespace Mediaverse.Application.JointContentConsumption.Common.Dtos
{
    public class PlaylistItemDto
    {
        public ContentIdDto ContentId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
    }
}
```

Б.47 Текст файла RoomDto.cs

```

using System;
using
Mediaverse.Application.JointContentConsumption.Commands.SwitchContent.Dtos;

namespace Mediaverse.Application.JointContentConsumption.Common.Dtos
{
    public class RoomDto
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public Guid ActivePlaylistId { get; set; }
        public ContentIdDto ContentId { get; set; }
        public string Token { get; set; }
        public int MaxViewersQuantity { get; set; }
        public int CurrentViewersQuantity { get; set; }
    }
}

```

Б.48 Текст файла GetAvailablePlaylistsQuery.cs

```

using System;
using System.Collections.Generic;
using MediatR;
using
Mediaverse.Application.JointContentConsumption.Queries.GetPlaylist.Dtos;

namespace
Mediaverse.Application.JointContentConsumption.Queries.GetAvailablePlaylists
{
    public class GetAvailablePlaylistsQuery :
    IRequest<IList<SelectablePlaylistDto>>
    {
        public Guid HostId { get; set; }
    }
}

```

ДОДАТОК В
Слайди презентації

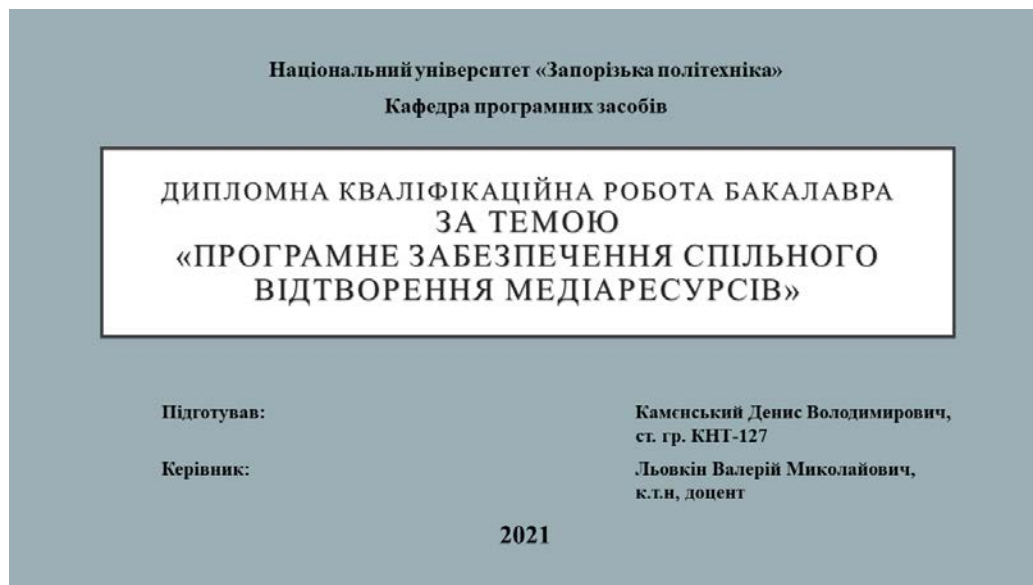


Рисунок В.1 – Слайд №1

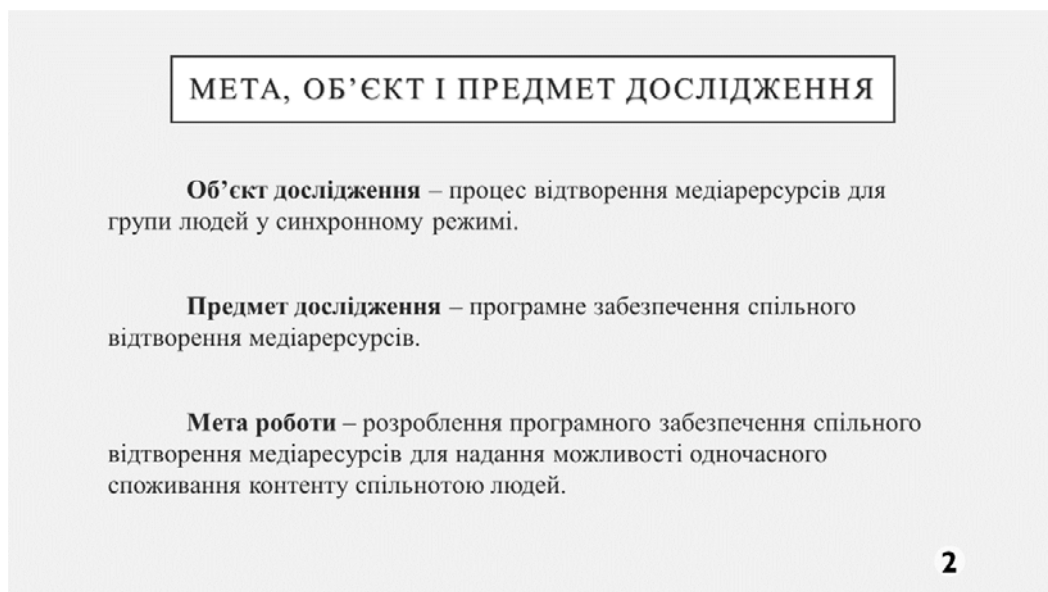


Рисунок В.2 – Слайд №2

ЗАВДАННЯ ДО РОБОТИ

- визначення архітектурних та технічних вимог до програмного забезпечення спільного відтворення медіаресурсів;
- вибір архітектурних підходів та технічних засобів розробленої системи відповідно до вимог;
- проектування програмного забезпечення;
- реалізація та впровадження розробленої системи;
- тестування та аналіз на відповідність поставленим вимогам.

3

Рисунок В.3 – Слайд №3

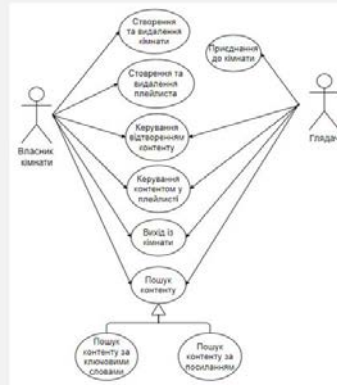
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

	Watch2gether	Twoseven	Kast
Велика кількість медіаресурсів	+	+/-	+/-
Наявність відкритих кімнат	-	-	-
Пошук за ключовими словами та посиланням	+	+/-	+
Можливість використання без реєстрації	+	-	-
Можливість відтворення контенту з пристрою	-	+	-
Наявність власної відеотеки	+	-	+
Відображення програмних помилок зручне для користувача	-	-	+

4

Рисунок В.4 – Слайд №4

ДІАГРАМА ПРЕЦЕДЕНТІВ



5

Рисунок В.5 – Слайд №5

БАГАТОШАРОВА АРХІТЕКТУРА



6

Рисунок В.6 – Слайд №6

ОРГАНІЗАЦІЯ ШАРІВ СИСТЕМИ					
Організація шару предметної області			Організація шару прикладної логіки		
	Transaction Script	Domain Model		Сервісний підхід	CQRS
Складність впровадження	низька	висока	Базовий принцип	прикладна логіка розміщується у спеціальних службових класах; методи описують сценарії використання	прикладна логіка поділяється за типом на модифікацію і «читання» об'єктів; кожен сценарій розміщується у відповідному класі
Складність підтримки на початку	низька – на початку розробки системи, висока – із розвитком системи	помірна			
Складність впровадження нової функціональності	низька – на початку розробки системи, висока – із розвитком системи	помірна	Складність впровадження	низька	помірна
Концентрація знань про предметну область	низька	висока	Розділення моделей запису та читання	у класичному варіанті - немає	можливе
Узгодженість даних	помірна – на початку розробки системи, низька – із розвитком системи	висока	Продуктивність	помірна	при впровадженні розділу моделей запису та читання - висока
Структурованість	помірна	висока	Структурованість	помірна	висока

7

Рисунок В.7 – Слайд №7



8

Рисунок В.8 – Слайд №8

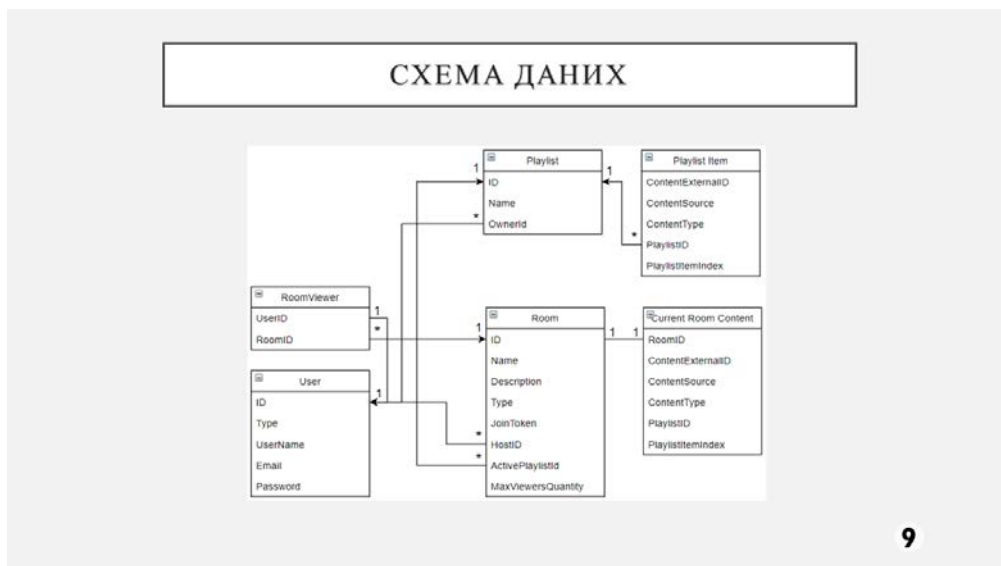


Рисунок В.9 – Слайд №9

ВИБІР МОВИ ПРОГРАМУВАННЯ		
	C#	Java
Простота роботи із примітивними типами даних	++	+
Наявність API для роботи із колекціями	+	+
Підтримка показників на методи	+ (делегати)	+ (за рахунок функціональних інтерфейсів)
Підтримка концепції властивостей	+	-
Поліморфізм	вимагає використання відповідних ключових слів	за замовченням
Можливість наділяти перелічення поведінкою	+ (лише за допомогою бібліотек)	+
Простота роботи із асинхронним кодом	++ (async/await)	+ (вимагає роботи з потоками)
Структуризація програми (розділення на субпроєкти)	++	+

У результаті порівняння обрано мову програмування C#

10

Рисунок В.10 – Слайд №10

ДІАГРАМА СТАНІВ КІМНАТИ



11

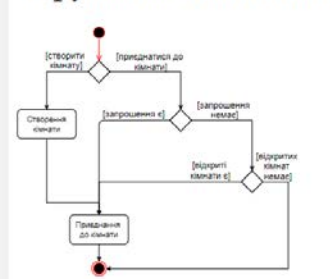
Рисунок В.11 – Слайд №11

ДІАГРАМИ ДІЯЛЬНОСТІ

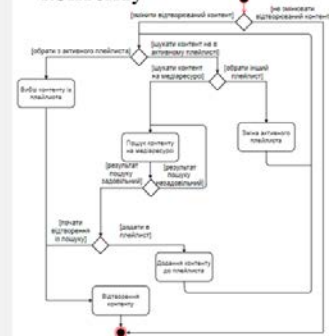
Ідентифікація користувача



Керування кімнатами



Пошук та відтворення контенту



12

Рисунок В.12 – Слайд №12

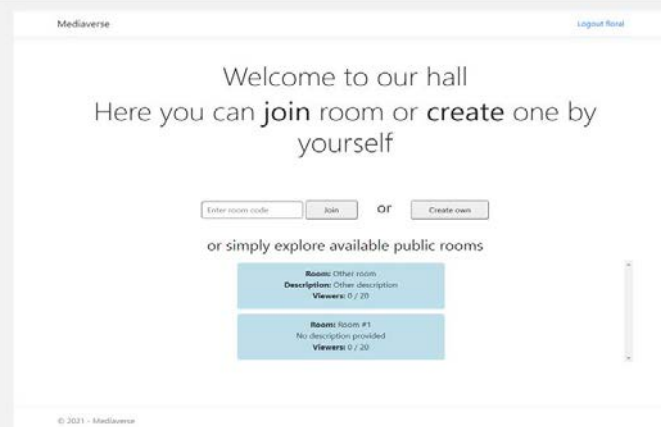
ДІАГРАМИ КЛАСІВ



13

Рисунок В.13 – Слайд №13

РОЗРОБЛЕНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

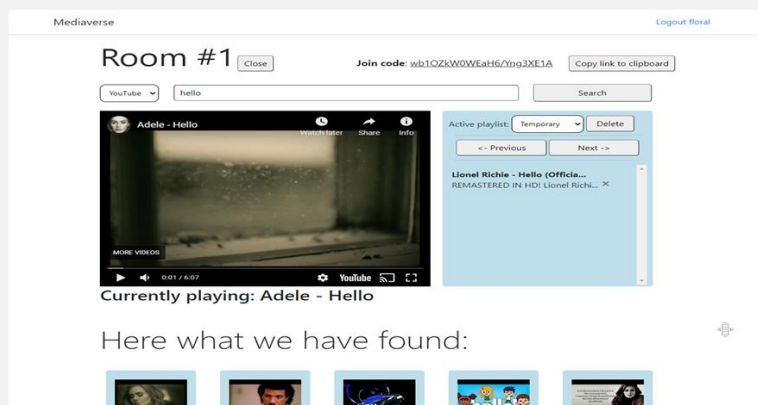


Сторінка керування кімнатами

14

Рисунок В.14 – Слайд №14

РОЗРОБЛЕНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ



Сторінка кімнати перегляду

15

Рисунок В.15 – Слайд №15

ПОРІВНЯННЯ ПРОГРАМИ ІЗ ІСНУЮЧИМИ РІШЕННЯМИ

	Watch2gether	Twoseven	Kast	Mediaverse
Велика кількість медіаресурсів	+	+-	+-	-
Наявність відкритих кімнат	-	-	-	+
Пошук за ключовими словами та посиланням	+	+-	+	+
Можливість використання без реєстрації	+	-	-	+
Можливість відтворення контенту з пристрою	-	+	-	-
Наявність власної відеотеки	+	-	+	-
Відображення програмних помилок зручне для користувача	-	-	+	+

16

Рисунок В.16 – Слайд №16

ВИСНОВКИ

- проаналізовано предметну область спільного відтворення контенту, виявлено основні поняття та існуючі рішення проблеми;
- досліджено особливості архітектурних вимог до розроблюваної системи;
- базуючись на результатах аналізу домену та висунутих функціональних вимогах, було спроектовано систему на основі багатопарової архітектури із використанням підходів «предметно-орієнтовне проектування» та «CQRS».

17

Рисунок В.17 – Слайд №17