МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра програмування
(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА БАГАТОПЛАТФОРМНОГО ДОДАТКУ ДЛЯ ВІДТВОРЕННЯ ІНТЕРНЕТ-ТЕЛЕБАЧЕННЯ Й ІНТЕРНЕТ-РАДІО З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ .NET MAUI

Виконав: студент групи ПМІ-41

Спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Копитко Б.В.
(підпис) (прізвище та ініціали)

Керівник доц. Ярошко С.А.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
Кафедра програмування
Спеціальність 122 – комп'ютерні науки
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри

_____

" " 20 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Копитка Бориса Володимировича
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка багатоплатформного додатку для відтворення інтернет-телебачення й інтернет-радіо з використанням фреймворку .NET MAUI

керівник роботи Ярошко Сергій Адамович, кандидат фізико-математичних наук, доцент
,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від "13" вересня 2022 року протокол №15

2. Строк подання студентом роботи 13 червня 2023 року

3. Вихідні дані до роботи
Офіційні документації фреймворків. Блоги авторитетних розробників. Інтернет-форуми та обговорення. GitHub репозиторії з прикладами

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)
1) Визначити та чітко сформулювати проблему
2) Описати перелік необхідних моделей даних та навести таблиці з їхніми властивостями
3) Зробити вибір технологій для розробки та обґрунтувати його
4) Задокументувати етапи реалізації та показати архітектуру рішення, навести приклади програмного коду
5) Продемонструвати роботу готового рішення, відвторюючи реальні сценарії використання, та навести знімки екрана, де зображені графічні інтерфейси користувача

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1) зображення високорівневої архітектури .NET MAUI
2) зображення інтерфейсів модифікації компонентів, генерації проектів та скаффолдингу в Visual Studio 2022
3) зображення програмного коду застосунків рішення
4) зображення інтерфейсів додатків рішення

6. Консультанти розділів роботи

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

7. Дата видачі завдання 13 вересня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	**Назва етапів дипломної роботи**	**Строк виконання етапів роботи**	**Примітка**
1	*Визначення проблеми*	*жовтень*	*Виконано*
2	*Вибір технологій для розробки*	*листопад*	*Виконано*
3	*Створення веб-застосунку*	*грудень-січень*	*Виконано*
4	*Реалізація прикладного програмного інтерфейсу*	*січень*	*Виконано*
5	*Побудова клієнтського додатка*	*лютий-квітень*	*Виконано*
6	*Тестування рішення*	*травень*	*Виконано*
7	*Оформлення дипломної роботи*	*травень-червень*	*Виконано*

Студент _____ Копитко Б.В.
Керівник роботи _____ Ярошко С.А.

АВТОРЕФЕРАТ

Метою цієї дипломної роботи було вивчити новий фреймворк від компанії Microsoft для створення сучасних, багатоплатформних і нативних застосунків .NET MAUI, проаналізувати ринок додатків, що займаються дистрибуцією вмісту за допомогою послуги OTT, та отримані знання застосувати для поєднання існуючих, покращених та нових технологій платформи .NET на прикладі побудови рішення для отримання доступу до інтернет-потоків мультимедіа та їх відтворення шляхом використання прикладного програмного інтерфейсу для визначення системи аутентифікації та авторизації з ролями, фреймворку для формування веб-додатку для налаштування даних вмісту та користувачів, фреймворку для розробки API для забезпечення обміну та обробки інформації в системі та .NET MAUI для реалізації кросплатформного захищеного клієнтського додатку для перегляду вмісту користувачами, правильність роботи якого перевірити через виконання конкретних сценаріїв використання.

У розділі «МОДЕЛЬ ДАНИХ» було визначено спосіб відображення користувачів та їхніх ролей у рішенні, згенерованого ASP.NET Core Identity та доповненого власноруч відповідно до тематики роботи, а також інформативно обрамленого вмісту для відтворення.

«ВИКОРИСТАНИЙ НАБІР ТЕХНОЛОГІЙ» містить детальний опис та обґрунтування:

- застосування інструментів для розробки - IDE Microsoft Visual Studio 2022 Community, системи контролю версій Git, сервісу GitHub та його клієнта GitHub Desktop;
- ініціалізації, взаємодії та наповнення бази даних на основі СКБД Microsoft SQL Server 2022 за допомогою Entity Framework Core;
- застосування в якості фреймворку для створення веб-застосунку ASP.NET Core MVC;

- реалізації API шляхом використання «minimal»-версії ASP.NET Core Web API; .NET MAUI – головного об’єкта дослідження цієї роботи.

В розділі «ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ» ведеться мова про підготовку середовища для та про кожен аспект розробки застосунків: веб-додаток і БД – генерація та взаємодія зі сховищем даних, скаффолдинг системи аутентифікації та авторизації, розбір ролі кожного з елементів шаблону проектування Model-View-Controller; API – реалізація алгоритмів активації прив’язки клієнтських пристроїв і надсилання, обробки та прийому даних через GET- і PUT-запити протоколу HTTP; клієнтська програма – створення сервісів для взаємодії з віддаленою інформацією, проектування сторінок, виконання динамічного налаштування системи навігації Shell, конфігурація відображення колекцій вмісту та їх пошуку, робота система активації та відтворення мультимедійних інтернет-потоків даних з власними оптимізованими елементами керування.

«ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА» показує приклади реального використання додатків з рішення GUI у веб-аплікації (демонстрація роботи системи входу та налаштування облікового запису користувача; отримання кодів доступу до копій клієнтської програми та їх станів використання; перегляду, відбору, створення, редагування та видалення даних користувачів та вмісту) та в клієнтському застосунку (процес прив’язки та відв’язки пристрою до та від користувача відповідно; відтворення вмісту; роботи з боковим виринаючим меню, вкладками та пошуком, а найголовніше – можливість користування додатками як і під управлінням ОС Windows, так і під Android).

Отже, ціль даної роботи досягнуто, адже було оновлено знання ASP.NET Core MVC, вивчено ASP.NET Core Web API та .NET MAUI, сформульовано їх основі можливості та переваги, проаналізовано ринок OTT-додатків і поставлено вимоги до них, та, на основі цієї інформації, поєднано існуючі, покращені та нові продукти .NET на прикладі побудови рішення для надання доступу до відтворення інтернет-потоків мультимедіа та їх перегляду, покращивши взаємодію з Git і GitHub,

визначивши обов’язкові моделі даних, ініціалізувавши БД MSSQL одразу з інформацією через EF Core, використавши ASP.NET Core Identity та власну схему для керування користувачами та ролями в вході та розмежуванні прав доступу в рішенні відповідно, застосувавши ASP.NET Core MVC для формування веб-застосунку для перегляду та використання CRUD-операцій до даних вмісту й користувачів, ASP.NET Core Web API для розробки програмного інтерфейсу для обміну та модифікації даних вмісту й користувачів та .NET MAUI для реалізації багатоплатформного додатка для отримання доступу до відтворення вмісту клієнтами, забезпечивши такі необхідні властивості як зручність, простота, безпека, стабільність, швидкість і сучасність. Робота рішення випробувана на сценаріях реального використання додатків під ОС Windows і Android при використанні їх різними типами користувачів та дає правильні результати.

З результатами побудови рішення та історією їх розробки можна ознайомитись на GitHub-сховищі, яке знаходиться за наступним посиланням: https://github.com/BorysKopytko/OTT-Creator

ЗМІСТ

ВСТУП

У рейтингах, складених згідно останнього на даний момент опитування з метою визначення переліку найпопулярніших інструментів, які бажані до використання або вже використовуються для програмування серед розробників, проведеного Stack Overflow у травні 2022 року, де участь брало більш, ніж 70 тисяч респондентів, як і новачків, так і професіоналів, складові платформи .NET займають достатньо високі позиції, з чого можна зробити висновок, що вона заслуговує на вивчення для легкого створення безпечних, стабільних, швидких і сучасних застосунків для настільних персональних комп'ютерів і веб-додатків.

У процесі випуску нових версій .NET, до 5-ої включно, для розробки додатків мовою програмування C# за допомогою цієї платформи з'являлись фреймворки інтерфейсу користувача для різних типів пристроїв. Для ПК під керуванням сімейства операційних систем Windows це були Windows Forms, Windows Presentation Foundation та Windows UI, а для створення багатоплатформеного нативного UI водночас як і для Windows, так і для macOS, Android й iOS використовувався Xamarin.Forms. Незважаючи на дане різноманіття, на жаль, жоден з них не забезпечував реалізацію таких сучасних тенденцій розробки застосунків як єдиний проект, кросплатформений прикладний програмний інтерфейс для застосування можливостей пристроїв, .NET Hot Reload тощо.

Проте, ця ситуація змінилась на краще з виходом .NET версії 6, адже в її склад був доданий .NET Multi-platform App UI – новий фреймворк від Microsoft, який дозволяє сьогоденним розробникам реалізовувати звичні речі на кшталт побудови сучасних, багатоплатформенних, нативно скомпільованих додатків, зберігши та покращивши вже існуючі напрацювання минулих UI-фреймворків компанії та доповнивши це абстрагуванням додатків для різних ОС в одну загальну структуру з використанням специфічних для певних пристроїв компонентів і способів зберігання та обробки даних.

1 ФОРМУЛЮВАННЯ ЗАДАЧІ

Завданням даної праці було вивчити та проаналізувати структуру та фунціонал .NET MAUI, дослідити ринок додатків, що поширюють вміст за послугою «over-the-top», та на основі отриманої й опрацьованої інформації поєднати існуючі, вдосконалені та нові технології .NET на прикладі побудови рішення для надання доступу до та перегляду мультимедійних інтернет-потоків даних, використавши прикладний програмний інтерфейс для визначення та використання моделей користувачів та ролей в аутентифікації й авторизації в рішенні, застосувавши фреймворк для формування веб-застосунку для створення, редагування, видалення та перегляду даних вмісту й облікових записів користувачів, фреймворк для розробки прикладного програмного інтерфейсу для надсилання, обробки й отримання даних вмісту й облікових записів користувачів та .NET MAUI для реалізації багатоплатформного додатка для отримання доступу до перегляду вмісту кінцевими користувачами. Роботу рішення випробувати на конкретних сценаріях використання.

2 МОДЕЛЬ ДАНИХ

Перед безпосереднім початком побудови рішення, необхідно точно визначити моделі даних, які будуть присутні в ньому.

Проаналізувавши ринок подібних застосунків, особливо взявши до уваги додатки, які займають перші сходинки рейтингів магазинів програм для популярних ОС, ознайомившись з викладеними думками в статтях експертів галузі медіа та застосувавши власний досвід використання схожих сервісів, я описав необхідні наступні моделі даних.

2.1 Користувачі

Першою моделлю даних стали майбутні користувачі. Вони мають як і спільні, так і відмінні властивості та можливості, які даються на основі на 2 ролей: адміністратори та клієнти.

2.1.1 Спільні властивості та можливості

Усі користувачі мають однакову структуру, спочатку згенеровану API ASP.NET Core Identity, а, згодом, доповнену полями, необхідними для реалізації теми рішення. Дана модель представлена в таблиці 2.1.

Таблиця 2.1 Модель користувача

Поле	Тип	Опис
Id	nvarchar(450)	Ідентифікатор користувача
UserName	nvarchar(256)	Ім’я користувача
NormalizedUserName	nvarchar(256)	Нормалізоване ім’я користувача
Email	nvarchar(256)	Електронна пошта користувача
NormalizedEmail	nvarchar(256)	Нормалізована електронна пошта користувача

EmailConfirmed	bit	Стан підтвердження електронної пошти користувача
PasswordHash	nvarchar(MAX)	Хеш пароля користувача
SecurityStamp	nvarchar(MAX)	Штамп відповідності безпеки облікового запису користувача
ConcurrencyStamp	nvarchar(MAX)	Стан для запобігання конфлікту паралельних оновлень
PhoneNumber	nvarchar(MAX)	Номер телефону користувача
PhoneNumberConfirmed	bit	Стан підтвердження номеру телефона користувача
TwoFactorEnabled	bit	Стан двофакторної аутентифікації
LockoutEnd	datetimeoffset(7)	Час кінця блокування облікового запису користувача
LockoutEnabled	bit	Стан блокування облікового запису користувача
AccessFailedCount	int	Кількість невдалих спроб входу користувача
CodesAndUse	nvarchar(MAX)	Коди доступу до клієнтського застосунку та їхній стан використання
FavoriteContentItemsIDs	nvarchar(MAX)	Список ідентифікаторів улюбленого вмісту
IsAllowed	bit	Стан дозволу доступу до використання додатка клієнта

Користувачі мають змогу:

- входити та виходити з веб-застосунку;
- змінювати пароль облікового запису.

Крім того, існує система ролей, суть якої в наданні доступу до передбачених різними рівнями прав можливостей. Вигляд її одиниці представлений в таблиці 2.2.

Таблиця 2.2 Модель ролі

Поле	Тип	Опис
Id	nvarchar(450)	Ідентифікатор ролі
Name	nvarchar(256)	Ім’я ролі
NormalizedName	nvarchar(256)	Нормалізоване ім’я ролі
ConcurrencyStamp	nvarchar(MAX)	Стан для запобігання конфлікту паралельних оновлень

Для співставлення користувачів і ролей існує модель «користувач-роль», яка зображена в таблиці 2.3.

Таблиця 2.3 Модель «користувач-роль»

Поле	Тип	Опис
UserId	nvarchar(450)	Ідентифікатор користувача
RoleId	nvarchar(450)	Ідентифікатор ролі

2.1.2 Клієнт

Клієнт є споживачем вмісту, доповнений однойменною роллю і може:

- переглядати коди доступу до та перегляду вмісту з клієнтського застосунку та використовувати для їх для активації своєї копії додатка на 5 різних пристроях;
- отримувати та переглядати вміст за його типом і категорією з можливістю зупинки та продовження, регулюючи гучність та змінюючи співвідношення сторін екрана;
- додавати улюблений вміст в однойменний пункт в меню типів клієнтського застосунку;
- деактивовувати коди доступу для відв'язки поточних пристроїв і перенесення прив'язки на інші.

2.1.3 Адміністратор

Адміністратор здатен робити все, що й користувач, а також керувати даними, доповнений однойменною роллю і має можливості:

- переглядати, створювати, редагувати та видаляти вміст;
- переглядати, створювати, редагувати та видаляти користувачів.

2.2 Вміст

Другим об'єктом даних став вміст – інформативно обрамлені мультимедійні інтернет-потоки, які використовуються для отримання доступу до них та перегляду їх клієнтом через його копію застосунку. На основі нього формуються колекції, які впорядковані типами та категорією або вподобаннями з певного типу адміністратором і користувачем відповідно. Він володіє властивостями, представленими в таблиці 2.4.

Таблиця 2.4 Модель вмісту

Поле	Тип	Опис
Id	int	Ідентифікатор вмісту
Name	nvarchar(MAX)	Назва вмісту

Category	nvarchar(MAX)	Категорія вмісту
Type	nvarchar(MAX)	Тип вмісту
Image	nvarchar(MAX)	Зображення вмісту, яке використовується для заміни відео в вмісті без нього
CroppedImage	nvarchar(MAX)	Піктограма вмісту, яка використовується при відображені колекцій моделі
Stream	nvarchar(MAX)	Посилання на мультимедійний інтернет-потік, дані якого відтворюються в клієнтському додатку
HasVideo	bit	Стан наявності відео в вмісті

3 ВИКОРИСТАНИЙ НАБІР ТЕХНОЛОГІЙ

Варто виділити те, що для розробки в основному застосувались складові платформи .NET саме версії 7, яка є, на даний момент, її найновішою стабільною ітерацією, за рахунок чого можна випробувати її останні впровадження, перевірені на надійність роботи інженерами Microsoft та спільнотою навколо неї.

3.1 Інструменти для розробки

Перш за все, я сформував середовище для неї за допомогою якого я виконував дане завдання. До нього ввійшли:

- Microsoft Visual Studio 2022 Community версії 17.6.0 – остання на момент початку роботи над задачею поширеного та безкоштовного IDE для створення сучасних додатків для різних операційних ОС та веб-застосунків і хмарних сервісів, орієнтованого на новачків і студентів. Він підтримує розробку за допомогою C# та допомогою усіх фреймворків, вибраних для реалізації завдання цієї роботи, дозволяє продуктивно будувати програми за рахунок зручного редактора та відлагоджувача, дає змогу розширювати функціонал через встановлення надбудов тощо. Цей вибір є логічним, адже даний продукт, як і використані згодом в даній праці технології створені та підтримуються однією компанією, отже, мають бути максимально сумісні між собою;
- Git – популярна безкоштовна розподілена система контролю версій із відкритим вихідним кодом, призначена для швидкої та ефективної розробки проектів. Це дозволить створювати гілки під час розробки для поступового нарощення та вдосконалення функціоналу застосунків;
- GitHub – хостинговий Інтернет-сервіс, який є найвідомішим рішенням для зберігання та поширення коду додатків, адже забезпечує контроль доступу, відстежування помилок, організації завдань та створення документації. На нього публікуватиметься репозиторій з програмами рішення, де можна бути відслідкувати історію розробки;

- GitHub Desktop – додаток для Git, який заміняє написання та виконання консольних команд шляхом відображення інтуїтивно зрозумілого графічного інтерфейсу користувача та інтегрованою взаємодією з GitHub. Він служить проміжною ланкою між локальним і віддаленим репозиторіями.

3.2 База даних

Розпочавши створення рішення, першим постало питання зберігання інформації. Враховуючи те, що під час роботи застосунків відбувається взаємодія з чутливими відомостями, а саме з мультимедійними Інтернет-потоками даних, які передають як і безкоштовний, так і платний вміст, дотриманий авторським правом, то він повинен бути захищеним. Для цього необхідно розмістити базу даних на віддаленому сервері та надавати доступ до через API, в якому клієнт проходитиме аутентифікацію й авторизацію, а саме посилання на API ніде не публікувати для мінімізації випадкових та цілеспрямованих запитів поза клієнтським додатком.

3.2.1 Microsoft SQL Server 2022

Microsoft SQL Server 2022 – найновіша версія пропрієтарної системи керування БД. Я її обрав, тому що:

- вона є реляційною, що є найкращим варіантом при використанні попередньо чітко визначених моделей даних;
- має покращену продуктивність і безпеку за рахунок використання вбудованих алгоритмів стиснення даних і функцій шифрування;
- в неї наявні платні видання Enterprise і Standard для великого та малого бізнесів відповідно та безкоштовних Express і Developer для користувачів, яким потрібні основні можливості даної СКБД і розробників відповідно;
- очевидно, що для неї написана найрозгорнутіша документація по використанню з іншими продуктами Microsoft, що буде особливо корисно в цій праці тощо.

3.2.2 Entity Framework Core 7.0

Після визначення СКБД, щоб мати тестові дані для розробки програм, було обрано фреймворк з відкритим вихідним кодом об’єктно-реляційного відображення Entity Framework Core версії 7.0, адже він є невеликим за обсягом, розширюваним, багатоплатформним і дає змогу:

- розробникам .NET працювати з базою даних за допомогою об’єктів даної платформи, тим самим усуває потребу в більшості коду доступу до даних, який зазвичай потрібно писати;
- застосовувати різні підходи до створення сутностей і бази даних, у тому числі й «code-first» - створення БД і її наповнення з класів сутностей і контексту - представлення сеансу взаємодії із базою даних через міграції – функції для ініціалізації й оновлення БД;
- використовувати в зв’язці з іншими фреймворками платформи .NET, тому що постійно оновлюється і він сам, і його документація тощо.

3.3 Веб-застосунок

Для створення веб-додатків за допомогою платформи .NET можна використовувати різні технології. ASP.NET підтримує ряд моделей програмування з присутністю UI для їх створення, серед яких присутні і сторонні фреймворки та бібліотеки, і нативні:

- Web Forms;
- MVC;
- Core MVC;
- Core Razor Pages;
- Blazor тощо.

Порівнявши доступну інформацію про всі, за такими важливими критеріями як безпека, стабільність, швидкість, поширеність, актуальність, довгота існування та підтримки, розгорнутість документації та простота розробки, я прийшов до висновку, що в рамках даного проекту найкращим варіантом є застосування вже

тривалий час існуючого ASP.NET Core MVC, адже, як мінімум, Web Forms і звичайний MVC є застарілими технологіями, а особливості архітектури та роботи Razor Pages і Blazor призводять до меншого обсягу документації та показових прикладів використання.

Відповідно до вимог, сформульованих в завданні, і зробленого вибору фреймворку веб-застосунку, для впровадження аутентифікації та авторизації в ньому, очевидною є рекомендація використання ASP.NET Core Identity, що в достатньому обсязі задовольняє потреби системи безпеки рішення.

3.3.1 ASP.NET Core MVC

ASP.NET Core MVC – це багатоплатформний і високопродуктивний фреймворк з відкритим вихідним кодом для побудови сучасних, інтегрованих з хмарними технологіями веб-додатків.

Для безпеки системи та економії ресурсів користувача він генерує користувацький інтерфейс на сервері та використовує архітектурний паттерн Model-View-Controller, який розділяє проект на 3 основні групи компонентів: моделі, представлення та контролери. Запити користувачів скеровуються у контролер, який відповідає за роботу з моделлю для виконання дій користувача або отримання результатів, після чого він вибирає представлення для відображення з потрібними даними моделі користувачеві.

Також суттєвою перевагою використання цієї моделі програмування Інтернет-застосунків є те, що її шаблон проектування дозволяє легке доповнення та перехід від малих до великих за розмірами рішень за рахунок чіткого розподілу завдань між компонентами системи, що надає її гнучкості.

3.3.2 ASP.NET Core Identity

ASP.NET Core Identity – це поєднання API для аутентифікації та авторизації, та UI для реєстрації та входу й виходу з веб-застосунку, яке керує:

- користувачами;

- паролями;
- даними профілю;
- ролями;
- токенами;
- підтвердженням електронної пошти тощо.

3.4 API - ASP.NET Core Web API

Щоб встановити зв’язок між БД, розміщеною на віддаленому сервері, та веб чи багатоплатформним застосунками, необхідно створити API, який зазвичай вирішує завдання надсилання, обробки та отримання інформації, а в даній роботі буде також займатися активацією та деактивацією кодів доступу до застосунку клієнтів безпечно, стабільно та швидко. Для збільшення надійності рішення, необхідно реалізувати його окремо від веб-застосунку, щоб вони могли працювати незалежно один від одного, незважаючи на будь-які впливи на кожен з них по одному.

Існує 4 поширених варіанти роботи API:

- SOAP – клієнт і сервер обмінюються повідомленнями використовуючи XML, що є базовим і часто використовувалося в минулому;
- RPC – клієнт завершує функції на сервері, після чого сервер повертає клієнту вивід;
- Websocket – використовує JSON для передачі даних, підтримує двох-шляховий зв’язок між клієнтськими застосунками та сервером, може відправляти зворотні виклики під’єднаним до нього користувачам;
- REST – найбільш популярний та гнучкий, бо клієнт відправляє запити до сервера як дані, а сервер, в свою чергу, запускає внутрішні функції і повертає клієнту вивід.

Єдиним актуальним рішенням на даний момент є реалізація API на платформі .NET є ASP.NET Core WebAPI, адже воно відноситься до варіанту REST, а, отже,

використовує для обміну даними протокол HTTP та визначає такі необхідні в рамках цієї роботи функції для взаємодії з даними як GET, PUT, DELETE тощо.

ASP.NET Core підтримує створення API, використовуючи множину контролерів або єдиний файл. Вчергове проаналізувавши сформульовані вимоги до завдання, я обрав 2 вдосконалений варіант «minimal», який був спроектований для створення легкого HTTP API за рахунок малої кількості залежностей і файлів програми, що є ідеальною опцією для створення мікросервісів з CRUD-операціями над моделями даних. У цьому випадку обробка усіх запитів буде описана та відбуватися в вхідній точці застосунку – файлі Program.cs. Таким чином, це полегшує роботу розробника шляхом найбільш можливого спрощення структури проекту та відображенням всього його фунціоналу в одному місці.

3.5 Клієнтський застосунок

Останнім став вибір фреймворку, який водночас буде хорошим предметом досліджень через те, що є новим для платформи .NET і допоможе побудувати захищену для усіх учасників системи, зручну клієнтську програму з інтуїтивно зрозумілим графічним інтерфейсом для отримання доступу до та відтворення вмісту, яка буде вирізнятися стабільністю та швидкістю роботи.

Ним став .NET MAUI – багатоплатформний фреймворк для створення нативних додатків для різних типів пристроїв. Використовуючи його, можна реалізовувати застосунки для Windows, macOS, Android, iOS і Tizen – лідера ринку ОС для смарт-телевізорів від компанії Samsung (що розширює коло потенційних користувачів, особливо, коли суть додатка торкається теми моєї праці), з використанням єдиної спільної кодової бази з мінімальними індивідуальними налаштуваннями специфічних налаштувань притаманним певним платформам. Він має відкритий вихідний програмний код і є еволюцією Xamarin.Forms, яка дозволила побудову як і мобільних, так і настільних додатків. Його елементи керування були перебудовані з нуля, що дало суттєвий приріст у продуктивності.

Для підтвердження здатності даного описаного фреймворку до розширення було вирішено застосувати .NET MAUI Community Toolkit - набору звичних для побудови застосунків елементів для .NET MAUI, які розробники, як правило, відтворюють у багатьох програмах повторно, відповідно дана колекції спрощує їм роботу та економить витрати на розробку під .NET MAUI.

3.5.1 .NET MAUI

Писати код для .NET MAUI-застосунку можна на ПК та Mac 2 шляхами: розміткою XAML і мовою програмування C#, зберігаючи одні макет дизайну на усіх платформах (при використанні нативних для ОС стилів) і бізнес-логіку.

Даний фреймворк об'єднав багато ОС в єдиний прикладний програмний інтерфейс, що дозволяє писати однакові алгоритми одноразово для різних платформ. Це стало можливим за рахунок того, що .NET версії 6 чи новішої має імплементації платформи .NET для Windows, macOS, Android, iOS, які працюють з однієї бібліотекою базових класів (далі – BCL), яка в свою чергу, взаємодіє з відповідним середовищем виконання, специфічного для ОС. Для Windows-систем це .NET CoreCLR, а для non-Windows – Mono. Для кращого розуміння принципу роботи даного API, варто ознайомитись з рисунком 3.1, на якому видно, що для застосунку .NET MAUI пишеться код (1), який контактує з .NET MAUI API (2), який в свою чергу працює з API нативної ОС (3). Також, при потребі, можливе написання коду, який зв’язується напряму з нативними API, оминаючи API MAUI.

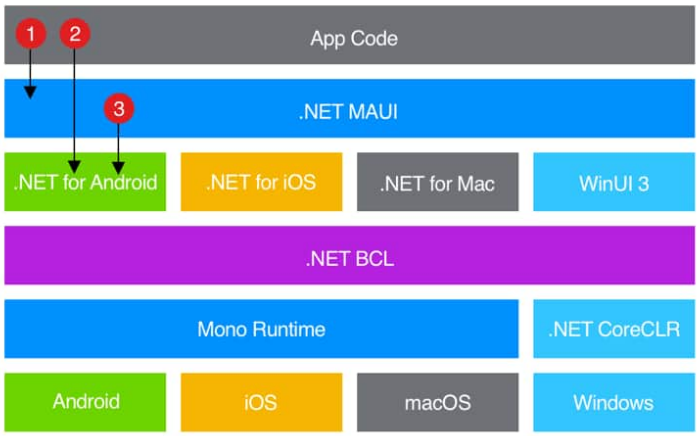

Рисунок 3.1 - Високорівневе подання типового додатку .NET MAUI

Багатоплатформність і нативність водночас досягаються компілюванням нативних застосунків для:

- Windows - використанням бібліотеки Windows UI 3;
- Android – переходом з C# у проміжну мову IL, а з неї під час виконання (JIT - just-in-time) до збірки;
- iOS – побудовою під час збірки (AOT – ahead-of-time), компільованого з C# до власного ARM-коду;
- macOS – через Mac Catalyst, шляхом конвертації та доповнення додатка для iOS.

Якщо розглядати .NET MAUI з точки зору змоги відображення та обробки даних (як і одинарних, так і колекцій), виконання дій, сигналізування діяльності тощо, то можна виділити наступні переваги:

- генерація сторінок та представлень різних типів, які різняться спрямуванням завдань, які за допомогою них вирішуються, підтримуючи застосуванням різноманітної об’єктів та способів малювання графіки та використання обробників для розширення базового функціоналу інтерфейсу;
- наявність API для доступу до специфічних можливостей різних пристроїв, наприклад, датчиків (GPS, компас, гіроскоп, акселерометр тощо), отримання та обробки звичайної та захищеної інформації у вигляді рядків чи файлів, відображення інформацій про девайс, робота з буфером обміну, функцій доступності та ін.;
- спрощений вибір налагоджувача і присутність .NET Hot Reload – функція, яка дозволяє змінювати вихідний код програми під час її виконання, що дуже зручно при роботі з великими рішеннями, які потребують затрати великих ресурсів при завантаженні спочатку тощо.

Варто також відзначити, що Visual Studio дозволяє налаштування та застосування емулятора Android для відлагодження роботи застосунку під

управлінням даної ОС без використання фізичного пристрою з нею, чого не можна сказати про iOS чи macOS, які потребують для цього реального Mac.

3.5.2 .NET MAUI Community Toolkit

Дане розширення поділяється на 3 пакети, які різняться за призначенням:

- CommunityToolkit.Maui – надають до доступ до різних видів анімації, поведінки, перетворювачів і представлень;
- CommunityToolkit.Maui.Markup - реалізовують легкі за застосуванням допоміжні методи та класи для спрощення створення декларативних GUI у C#;
- CommunityToolkit.Maui.MediaElement – додає можливість відтворювати відео- та аудіо- файли.

Особливістю даного пакету є те, що він створений і підтримується спільнотою людей, сформованою навколо платформи .NET, відповідно кожен може подати свій варіант реалізації того чи іншого компоненту і його буде розглянуто авторитетними членами спільноти та інженерами .NET MAUI, щоб забезпечити повну сумісність між основними й додатковими інструментами.

4 ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

До початку безпосереднього програмування рішення, перш за все, я ознайомився з документацією кожного вибраного інструмента для розробки. Після цього, я встановив необхідне програмне забезпечення для виконання завдання даної роботи. Для створення додатків, використовуючи обрані фреймворки, Visual Studio була модифікована через Installer – в неї були додані компоненти, зображені на рисунку 4.1.

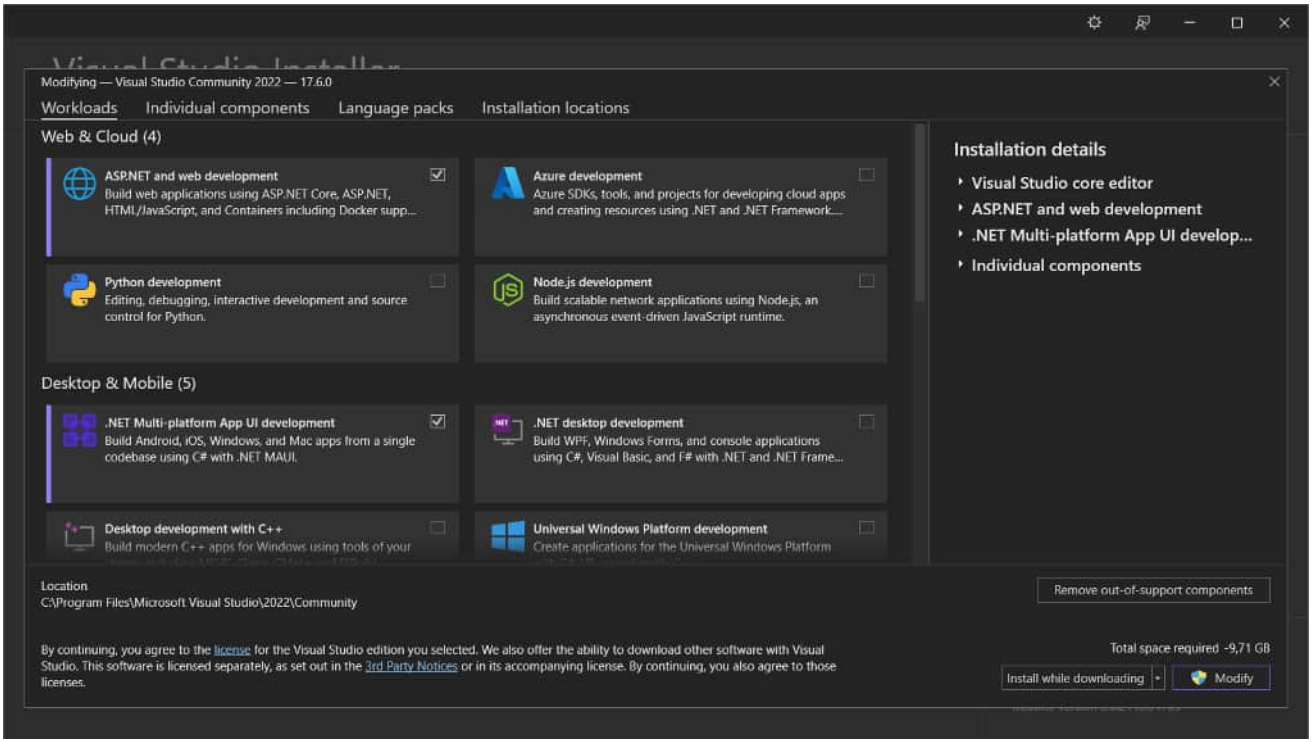

Рисунок 4.1 – вікно Visual Studio Installer з обраними для реалізації задачі складовими

Згодом, я створив локальний Git- і віддалений GitHub- репозиторії для контролю версій програмного коду проектів. У ньому я першочергово розмістив файли:

- README.md – для короткого опису властивостей і можливостей застосунків;
- .gitignore – для відсіювання різних даних, яка не варто розміщувати на віддаленому сховищі, наприклад, збірка чи кеш;

- ліцензії Apache 2.0 – дає дозвіл на комерційне використання, модифікацію, використання патенту та приватне користування й знімає відповідальність і гарантії за умови повідомлення про ліцензію та авторські права та документування внесених змін.

Потім, я створив рішення за допомогою нового порожнього шаблона в Visual Studio і нарешті розпочав його наповнення додатками, описаними далі.

4.1 Веб-застосунок і база даних

Спочатку, я згенерував базову структуру програми за допомогою шаблону ASP.NET Core MVC, застосувавши параметри вказані на рисунку 4.2.

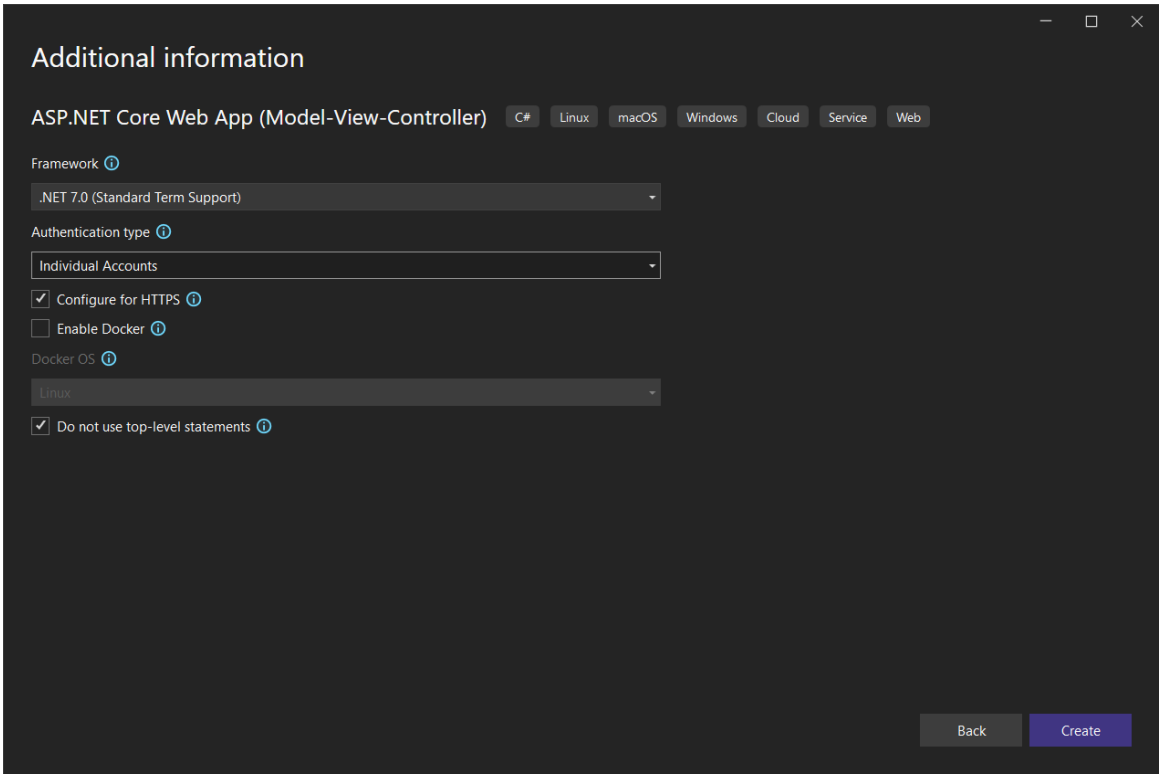

Рисунок 4.2 – Вікно створення проекту ASP.NET Core MVC

Першою гостро постала потреба в створенні БД для зберігання даних вмісту та користувачів. Першим кроком до цього стала імплементація моделі вмісту ContentItem (її структуру було розібрано у таблиці 2.4 цієї праці) у вигляді класу C#. Наступним було використано генератор системи аутентифікації та авторизації ASP.NET Core Identity, що водночас заощадило ресурси на розробці базових сторінок, створенню моделі користувача та класу ApplicationIdentityDbContext для

роботи EF Core з MSSQL. У ньому відбувається ініціалізація локальної бази даних через підхід «code-first» - формування її структури з попередньо визначених класів моделей і, в даному випадку, зі стандартними таблицями користувачів і ролей. Графічний інтерфейс описаного генератора представлений на рисунку 4.3,

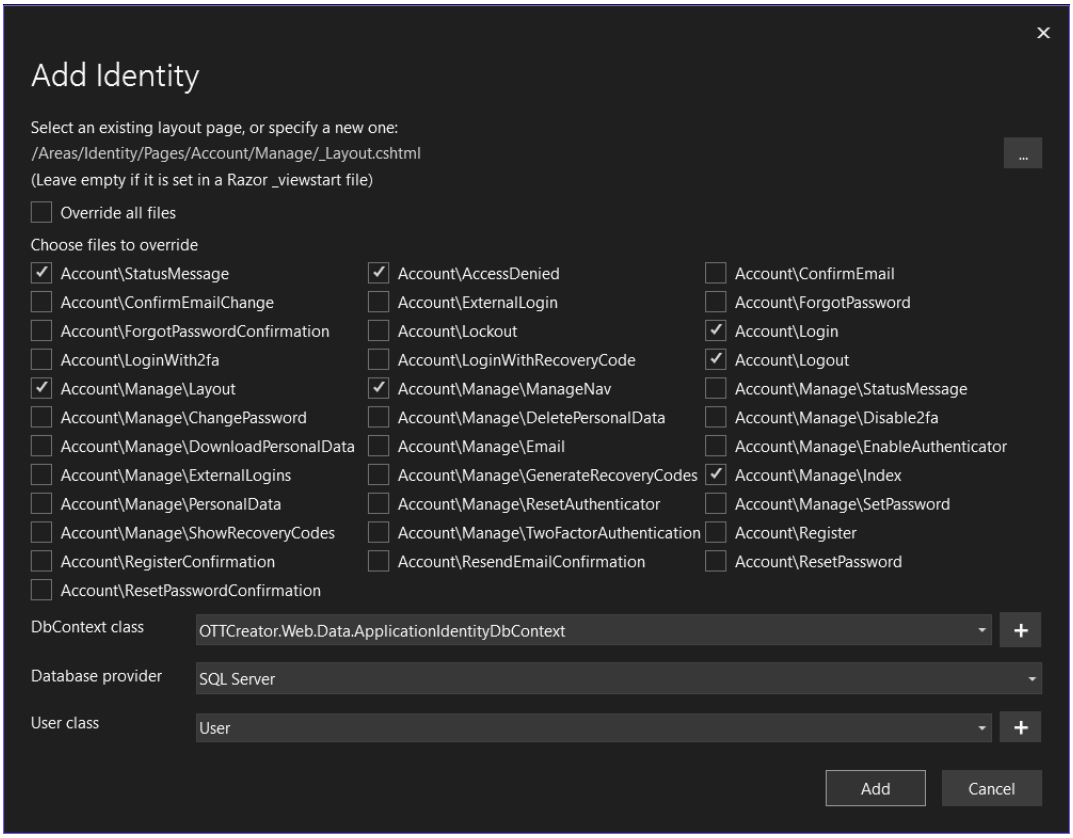

Рисунок 4.3 – Генератор системи аутентифікації та авторизації ASP.NET Core Identity

Враховуючи те, що Identity згенерував базовий клас User, я доповнив його додатковими обов’язковими полями, вже визначеними в таблиці 2.1. Клас Role є вбудованим в систему, тому, зважаючи на вимоги, його можна залишити таким, який він є.

Щоб побудувати БД з коду контексту, після цього треба застосувати міграції, які є початковою структурою або вдосконаленням БД і будуються залежно від наявності та ступеня оновленості існуючої БД. Це робиться в командному рядку командами Add-Migration “Назва міграції”, щоб створити міграцію та Update-Database, щоб або створити, або оновити існуючу БД.

Для роботи з БД я взяв до уваги рекомендацію використовувати GenericRepository – абстракційний шар між даними та бізнес-логікою застосунку та UnitOfWork – паттерн об’єктно-реляційної поведінки для проведення транзакцій. Для цього слід водночас реалізовувати як і інтерфейси, так і класи, щоб мати змогу застосовувати Dependecy Injection – шаблон проектування, який дозволяє розмежовувати об’єкт і використання залежностей, тобто зробити його простішим до вбудовування. У результаті, я отримав інтерфейси та класи GenericRepository, UnitOfWork, UserRepository і ContentRepository. Їхня кількість спонукала мене об’єднати всі імплементовані репозиторії різних типів даних в один за допомогою RepositoryWrapper, який дає доступ до них всіх водночас. Повну суть його роботи зображено на рисунку 4.4.

```
public class RepositoryWrapper : IRepositoryWrapper
{
    private ApplicationIdentityDbContext _dbContext;

    private IUserRepository _userRepository;
    private IContentRepository _contentRepository;

    public IUserRepository UserRepository
    {
        get
        {
            if (_userRepository == null)
                _userRepository = new UserRepository(_dbContext);
            return _userRepository;
        }
    }

    public IContentRepository ContentRepository
    {
        get
        {
            if (_contentRepository == null)
                _contentRepository = new ContentRepository(_dbContext);
            return _contentRepository;
        }
    }

    public RepositoryWrapper(ApplicationIdentityDbContext applicationDBContext)
    {
        _dbContext = applicationDBContext;
    }

    public void Save()
    {
        _dbContext.SaveChanges();
    }

    public async Task SaveAsync()
    {
        await _dbContext.SaveChangesAsync();
    }
}
```

Рисунок 4.4 – Програмний код класу RepositoryWrapper

Далі, необхідним для завершення роботи над аспектом опрацювання інформації, стало впровадження моделей представлення. Вони можуть приховувати чутливі та об’єднувати звичайні дані з моделей для безпеки та швидкості відображення з контролера в представлення та навпаки. Зазвичай вони мають в складі себе анотації, які спрощують побудову інтерфейсу шляхом автоматичного відображення назв властивостей. Приклад реалізації є на рисунку 4.5.

```
public class UserViewModel
{
    public string Id { get; set; }

    [Display(Name = "Електронна пошта")]
    public string Email { get; set; }

    [Display(Name = "Номер телефону")]
    public string PhoneNumber { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Пароль")]
    public string Password { get; set; }

    [Display(Name = "Підтвердження пароля")]
    [DataType(DataType.Password)]
    public string ConfirmPassword { get; set; }

    public List<SelectListItem> Roles { get; set; } = new List<SelectListItem>
();
    [Display(Name = "Роль")]
    public string RoleId { get; set; }

    [Display(Name = "Оплачено")]
    public bool IsAllowed { get; set; }
}
```

Рисунок 4.5 – Модель представлення UserViewModel

Після повної реалізації моделей, БД та моделей представлень, прийшов час створення контролерів, які будуть займатися їхньою обробкою. Першим став той, який відповідає за взаємодію з Identity – UserController. У ньому визначені GET - функції для отримання порожніх та змістовних моделей представлення для перегляду й взаємодії, а POST - для створення, оновлення та видалення даних в БД після їхнього опрацювання. Вони поділені на окремі імплементації для специфічних для кожної ролі властивостей та можливостей. На рисунку 4.6 видно варіант написання коду для GET-дій, які захищені шляхом авторизації.

```
[Authorize(Roles = "Адміністратор")]
public async Task<IActionResult> Configure()
{
    var model = new List<UserListViewModel>();

    var administrators = await userManager.GetUsersInRoleAsync("Адміністратор");
    var users = await userManager.GetUsersInRoleAsync("Клієнт");

    foreach (var administrator in administrators)
        model.Add(new UserListViewModel { Id = administrator.Id,
            Email = administrator.Email, PhoneNumber = administrator.PhoneNumber,
            Role = "Адміністратор", IsAllowed = administrator.IsAllowed });
    foreach (var user in users)
        model.Add(new UserListViewModel { Id = user.Id,
            Email = user.Email, PhoneNumber = user.PhoneNumber,
            Role = "Клієнт", IsAllowed = user.IsAllowed });

    return View(model);
}
```

Рисунок 4.6 – GET-дія Configure для отримання списку моделей представлення

Аналогічний принцип роботи в ContentControler. Він є більш узагальненим прикладом контролерів, бо хоч і не працює з полями Identity, але обробляє велику кількість власних визначених даних і теж реалізовує CRUD-операції шляхом отримання даних з БД, перетворення в потрібну за наповненістю модель представлення та відправлення її на саме представлення, після завершення роботи з яким користувач спрямовує оновлену інформацію назад на контролер, який дістає з моделі представлення нові дані і вносить зміни до самої моделі та, на кінець, зберігає її до БД. Для демонстрації можливостей POST-дій наведений рисунок 4.7.

```
[HttpPost]
   public async Task<IActionResult> Create(ContentListViewModel model)
   {
       var contentItem = new ContentItem()
       {
           Name = model.Name,
           Category = model.Category,
           Type = model.Type,
           Image = model.Image,
           CroppedImage = model.CroppedImage,
           Stream = model.Stream,
           HasVideo = model.HasVideo
       };

       await repositoryWrapper.ContentRepository.AddAsync(contentItem);
       await unitOfWork.Commit();

       return RedirectToAction(nameof(Configure));
   }
```

Рисунок 4.7 POST-дія Create для збереження нового вмісту

HomeController – головний за замовчуванням у всіх застосунках ASP.NET Core MVC. У моєму йому надана роль переспрямування на домашню сторінку з описом додатка та, що важливіше, відображення кодів до доступу до клієнтського додатка та стан їх використання.

Завершивши побудову шляхи отримання та обробки інформації, залишилось її візуалізувати. В ASP.NET Core MVC це можливо завдяки Razor Pages – представлень, які крім традиційних у веб-розробці розмітки HTML5, мови стилю сторінок CSS3 та мови програмування JavaScript застосовують спеціальну розмітку Razor та C#, що полегшує відображення даних, особливо великих за розміром чи нестандартних за виглядом. В рамках цієї роботи вони виявились корисними при передачі моделей представлень рядками в таблицю. У початковому складі аплікації була одна домашня сторінка та багато для аутентифікації та авторизації (завдяки генеруванню Identity), які були модифіковані новою інформацією та перекладом з англійською на українську мову. Їх було доповнено новими представленнями, суть яких визначалась відповідно до можливостей контролерів, тобто перегляду – Configure, створення – Create, редагування – Edit, видалення – Delete, які створено по екземпляру для контролерів вмісту та користувачів, а також ще по одному для кожної ролі, адже в них різні властивості та варіанти використання. Для наочного розуміння надано рисунок 4.8.

Запуск веб-додатка на сервері відбувається шляхом завантаження програми у вигляді комадного рядка, який в свою чергу відображає події, які відбуваються під час роботи, залежно від того як його налаштувати. Мій налаштований для налагодження всіх аспектів застосунку, відповідно надає інформацію навіть про кожну операцію, яка відбувається в БД, що видно на рисунку 4.9.

```
@using OTTCreator.WebApp.ViewModels;
@model UserViewModel

@{
    ViewData["Title"] = "Редагувати користувача";
}

<form method="post">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="row">
        <div class="col-lg-6">
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PhoneNumber" class="control-label"></label>
                <input asp-for="PhoneNumber" class="form-control" />
                <span asp-validation-for="PhoneNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="IsAllowed" class="control-label"></label>
                <input asp-for="IsAllowed" class="form-check" name="IsAllowed" type="checkbox" value="true" />
                <span asp-validation-for="IsAllowed" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <input type="submit" value="Зберегти" class="btn btn-primary" />
            <button asp-controller="User" asp-action="Configure" class="btn btn-secondary">Скасувати</button>
        </div>
    </div>
</form>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.13.5/js/standalone/selectize.min.js"></script>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.13.5/css/selectize.bootstrap4.min.css" />
}
```

Рисунок 4.8 – Представлення EditUser, на яке посилається UserController

```
C:\Users\kopyt\Desktop\Qualification diploma (bachelor) work\OTT Creator\OTT-Creator\OTTCreator\OTTCreator.WebApp\bin\Debu...
      SELECT TOP(1) [a].[Id], [a].[AccessFailedCount], [a].[CodesAndUse], [a].[ConcurrencyStamp], [a].[Email], [a].[Emai
lConfirmed], [a].[FavoriteContentItemsIDs], [a].[IsAllowed], [a].[LockoutEnabled], [a].[LockoutEnd], [a].[NormalizedEmai
l], [a].[NormalizedUserName], [a].[PasswordHash], [a].[PhoneNumber], [a].[PhoneNumberConfirmed], [a].[SecurityStamp], [a
].[TwoFactorEnabled], [a].[UserName]
      FROM [AspNetUsers] AS [a]
      WHERE [a].[Id] = @__GetUserId_0
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__userId_0='?' (Size = 450)], CommandType='Text', CommandTimeout='30']
      SELECT [a0].[Name]
      FROM [AspNetUserRoles] AS [a]
      INNER JOIN [AspNetRoles] AS [a0] ON [a].[RoleId] = [a0].[Id]
      WHERE [a].[UserId] = @__userId_0
warn: Microsoft.AspNetCore.Watch.BrowserRefresh.BrowserRefreshMiddleware[3]
      Unable to configure browser refresh script injection on the response. Consider manually adding '<script src="/_fra
mework/aspnetcore-browser-refresh.js"></script>' to the body of the page.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__GetUserId_0='?' (Size = 450)], CommandType='Text', CommandTimeout='30']
      SELECT TOP(1) [a].[Id], [a].[AccessFailedCount], [a].[CodesAndUse], [a].[ConcurrencyStamp], [a].[Email], [a].[Emai
lConfirmed], [a].[FavoriteContentItemsIDs], [a].[IsAllowed], [a].[LockoutEnabled], [a].[LockoutEnd], [a].[NormalizedEmai
l], [a].[NormalizedUserName], [a].[PasswordHash], [a].[PhoneNumber], [a].[PhoneNumberConfirmed], [a].[SecurityStamp], [a
].[TwoFactorEnabled], [a].[UserName]
      FROM [AspNetUsers] AS [a]
      WHERE [a].[Id] = @__GetUserId_0
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__userId_0='?' (Size = 450)], CommandType='Text', CommandTimeout='30']
      SELECT [a0].[Name]
      FROM [AspNetUserRoles] AS [a]
      INNER JOIN [AspNetRoles] AS [a0] ON [a].[RoleId] = [a0].[Id]
      WHERE [a].[UserId] = @__userId_0
```

Рисунок 4.9 – Демонстрація роботи веб-застосунку в вигляді командного рядка

Остаточна архітектура даної частини рішення зображена на рисунку 4.10:

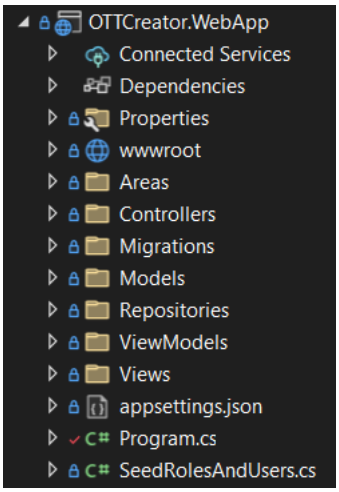

Рисунок 4.10 – Архітектура веб-застосунку

4.2 API

Я створив проект за шаблоном ASP.NET Core Web API, з параметрами зображеними на рисунку 4.11, вдосконаливши його використанням мінімальної версії, що вже аргументовано в пункті 3.4 даної роботи.

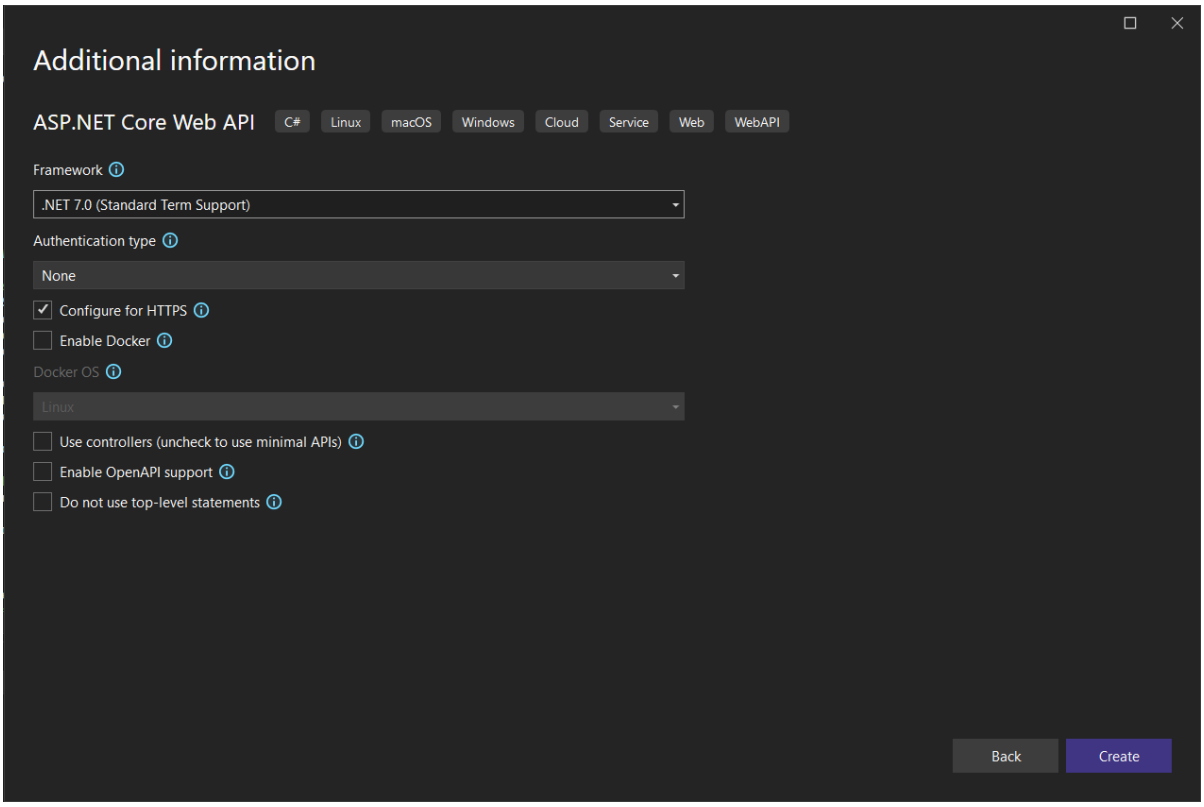

Рисунок 4.11 - Вікно створення проекту ASP.NET Core Web API

Єдиним файлом, який використовувався для виконання в даному застосунку є його вхідна точка – Program.cs. Але перед безпосередньою реалізацією запитів потрібно було вирішити як в нього аутентифіковувати користувачів, щоб зробити програмний інтерфейс захищеним. Для цього була придумана власна схема, яка передбачає вбудовування єдиного, постійно використовуваного GUID у всі копії клієнтських додатків та індивідуального для кожної з них. Саме за допомогою індивідуальних GUID відбувається активація, отримання доступу до та відтворення вмісту. Їх надається 5 кожному користувачеві для прив'язки будь-яких пристроїв. Відповідно, кожен запит в кінці себе приймає вбудований та зовнішній GUID, правильна комбінація яких й надає доступ до функцій API. Вбудований ключ в API розміщений в appsettings.json (адже й так відомий розробникам) та зчитується на початку виконання API перед початком обробки запитів.

В ньому було реалізовано такі GET-запити як:

- contentitems/{WebAPIkey}/{code} – отримання всього вмісту;
- types/{WebAPIkey}/{code} – отримання всіх типів вмісту;
- {type}/categories/{WebAPIkey}/{code} – отримання всіх категорій конкретного типу вмісту;
- {type}/{category}/contentitems/{WebAPIkey}/{code} – отримання всього вмісту конкретного категорії та типу;
- {type}/contentitems/favorites/{WebAPIkey}/{code} – отримання улюбленого вмісту конкретного типу;
- contentitems/{id}/{WebAPIkey}/{code} – отримання конкретного вмісту.

Спосіб написання обробки GET-запиту на прикладі {type}/{category}/contentitems/{WebAPIkey}/{code} є на рисунку 4.12.

```
app.MapGet("{type}/{category}/contentitems/{WebAPIkey}/{code}", async (string type, string category, string WebAPIkey,
string code, ApplicationIdentityDbContext db) =>
{
    if (WebAPIkey == WebAPIKey)
    {
        var user = await GetUserAsync(db, code);
        if (user != null && user.IsAllowed)
            return await db.ContentItems.Where(c => c.Type == type && c.Category == category).ToListAsync();
    }
    return null;
});
```

Рисунок 4.12 – Обробка GET-запиту {type}/{category}/contentitems/{WebAPIkey}/{code}

Також були розроблені наступні PUT-запити:

- contentitems/{id}/favorite/{WebAPIkey}/{code} для додавання та видалення конкретного вмісту до та з «улюбленого» відповідно для конкретного користувача;
- activate/{activateOrDeactivate}/{WebAPIkey}/{code} – для реалізації описаної мною схеми активації та деактивації клієнтської програми.

Варіант побудови обробки PUT-запиту на прикладі contentitems/{id}/favorite/{WebAPIkey}/{code} продемонстровано на рисунку 4.13.

```
app.MapPut("contentitems/{id}/favorite/{WebAPIkey}/{code}", async (int id, string WebAPIkey, string code,
ApplicationIdentityDbContext db) =>
{
    if (WebAPIkey == WebAPIKey)
    {
        var contentItem = await db.ContentItems.FindAsync(id);
        if (contentItem is null) return Results.NotFound();
        var user = await GetUserAsync(db, code);
        if (user != null && user.IsAllowed)
        {
            if (user.FavoriteContentItemsIDs.Contains(id))
                user.FavoriteContentItemsIDs.Remove(id);
            else
                user.FavoriteContentItemsIDs.Add(id);
            db.Entry(user).State = EntityState.Modified;
            await db.SaveChangesAsync();

        }
        return Results.NoContent();
    }
    return Results.Unauthorized();
});
```

Рисунок 4.13 – Обробка PUT-запиту contentitems/{id}/favorite/{WebAPIkey}/{code}

Даний програмний інтерфейс, аналогічно до веб-додатка, функціонує в вигляді командного рядка та інформує про події в системі, залежно від налаштувань

рівня сповіщень, які в мене максимально деталізовані в зв’язку з відлагодженням, що видно з наведеного рисунку 4.14.

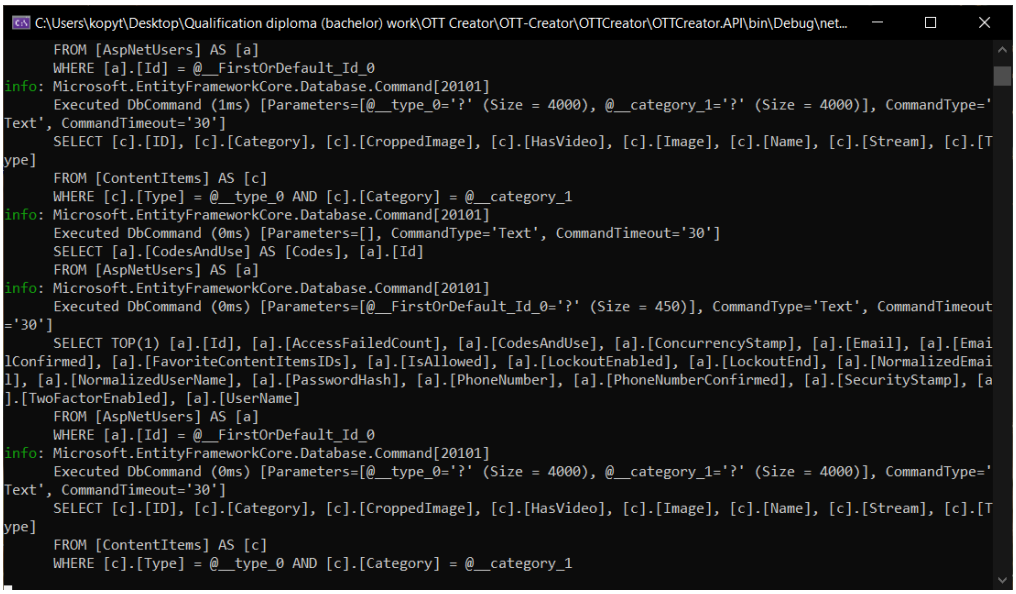

Рисунок 4.14 – Демонстрація роботи API в вигляді командного рядка

Архітектура даного програмного інтерфейсу насправді виправдовує назву «мінімальної», що простежується на рисунку 4.15.

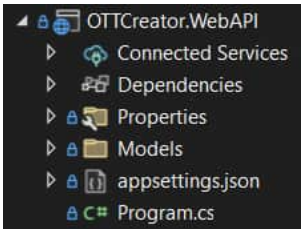

Рисунок 4.15 – Архітектура API

4.3 Клієнтський застосунок

Для генерації початкової архітектури даної частини рішення, застосував шаблон Visual Studio під назвою .NET MAUI App - основний об’єкт дослідження даної праці. Головна конфігурація застосунку відбувається в файлі класу MauiProgram.cs, де в методі CreateMauiApp відбувається під’єднання компонентів, шрифтів, ведення журналу подій, налаштування служб тощо.

Роботу над ним почав з того, що для зв’язку з вже імплементованим API необхідно було розробити сервіс, який оброблятиме надсилання та отримання

відповідей, зважаючи на індивідуальні особливості платформи, на якій додаток виконується. Для зручності його було поділено на 2 менших:

- HttpsClientHandlerService – повертає нативний обробник повідомлень для non-Windows систем, який використовується для їх прийому та віддачі;
- ContentService – безспосередньо створює запити, серіалізує дані в JSON і надсилає їх разом та навпаки при отриманні відповідей.

Приклади формування GET- і PUT-запитів до API в ContentService продемонстровані на рисунках 4.16 і 4.17 відповідно.

```
public async Task<List<ContentItem>> GetContentItemsAsync()
        {
            var WebAPIkey = await SecureStorage.Default.GetAsync("WebAPIKey");
            var code = await SecureStorage.Default.GetAsync("Code");
            var contentItems = new List<ContentItem>();

            var uri = new Uri(string.Format($"{Constants.WebAPIUrl}/contentitems/{WebAPIkey}/{code}",
string.Empty));
            try
            {
                var response = await client.GetAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    var content = await response.Content.ReadAsStringAsync();
                    contentItems = JsonSerializer.Deserialize<List<ContentItem>>(content, serializerOptions);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@"\tError {0}", ex.Message);
            }

            return contentItems;
        }
```

Рисунок 4.16 – Приклад формування GET-запиту в ContentService

```
        public async Task SaveContentItemFavoriteAsync(int id)
        {
            var WebAPIkey = await SecureStorage.Default.GetAsync("WebAPIKey");
            var code = await SecureStorage.Default.GetAsync("Code");

            var uri = new Uri(string.Format($"{Constants.WebAPIUrl}/contentitems/{id}/favorite/{WebAPIkey}/{code}",
string.Empty));

            try
            {
                var json = JsonSerializer.Serialize(id, serializerOptions);
                var content = new StringContent(json, Encoding.UTF8, "application/json");

                var response = await client.PutAsync(uri, content);

                if (response.IsSuccessStatusCode)
                    Debug.WriteLine(@"\tContentItem successfully saved.");
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@"\tError {0}", ex.Message);
            }
        }
```

Рисунок 4.17 – Приклад формування PUT-запиту в ContentService

Отримуючи та оновлюючи дані захищено та віддаленно, наступним постало питання відображення результатів даних операцій. Його було вирішено, створивши розмітку XAML для 3 необхідних власних типів сторінок на основі класу ContentPage, які відображають:

- CategoryPage – колекцію даних конкретних типу та категорії та пошук по всьому вмісту, його розмітка подана на рисунку 4.18;
- ContentItemPage – при відсутності коду прив’язки користувача до копії додатка - екран активації, а при наявності - обраний в CategoryPage вміст;
- SupportPage – телефон, пошту та сайт адміністраторів і код, яким пристрій прив’язаний до користувача та кнопку для його відв’язки, чим і поясюється вся її суть.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="OTTCreator.Client.Pages.CategoryPage"
             xmlns:controls="clr-namespace:OTTCreator.Client.Controls">
    <Shell.SearchHandler>
        <controls:ContentItemSearchHandler Placeholder="Введіть назву"
                                           ShowsResults="true">
            <controls:ContentItemSearchHandler.ItemTemplate>
                <DataTemplate>
                    <Grid Padding="10"
                          ColumnDefinitions="0.2*,0.8*">
                        <Image Source="{Binding CroppedImage}"
                               HeightRequest="50"
                               WidthRequest="50"
                               Margin="10"/>
                        <Label Grid.Column="1"
                               Text="{Binding Name}"
                               VerticalOptions="Center"
                               HorizontalOptions="Center"
                               Margin="10"/>
                    </Grid>
                </DataTemplate>
            </controls:ContentItemSearchHandler.ItemTemplate>
        </controls:ContentItemSearchHandler>
    </Shell.SearchHandler>
    <ScrollView Padding="10">
        <FlexLayout x:Name="ContentItemList"
                    Wrap="Wrap"
                    AlignItems="Start"
                    AlignContent="Start"/>
    </ScrollView>
</ContentPage>
```

Рисунок 4.18 – Розмітка CategoryPage

Також в комплекті з шаблоном генерування проекту йдуть сторінки App – точка входу в застосунок і AppShell - імплементація однієї з переваг даного фреймворку, а саме Shell – вбудованої навігаційної системи, яка дозволяє організувати навігаційну системи за рахунок:

- опису візуальної ієрархії всього додатка в одному місці;
- інтуїтивно зрозумілого інтерфейсу;
- шляхів навігації на базі URI до будь-якої сторінки;
- підтримки вбудовування пошуку.

При потребі тонких налаштувань, для .NET MAUI можна створити й власну систему навігації з нуля, використовуючи ContentPage (застосовний і в Shell для відображення вмісту сторінки), FlyoutPage, NavigationPage та TabbedPage.

Повертаючись до теми автоматичної генерації, в розмітці дизайну AppShell я розмістив верхню (header) та нижню (footer) частини для показу логотипу програми і інформації про авторські права, доповнив статичними пунктами «Поточне відтворення», «Улюблений вміст» і «Підтримка», а в code-behind – файлі коду виконання, пов'язаного з розміткою, в моєму випадку AppShell, в якому й відбувається ініціалізація системи переходів між сторінками, я написав функцію, яка через ContentService отримує всі необхідні дані для динамічної побудови інтерфейсу за допомогою FlyoutItem, TabBar і ShellContent, призначені для проектування бокового виринаючого меню, панелі вкладок зверху чи знизу (залежно від ОС) та адаптивної складової UI відповідно – елементи навігації, вміст яких надалі можна оновлювати далі безпосередньо на ContentPage. Для кращого розуміння, код AppShell.cs є на рисунку 4.19.

```
using OTTCreator.Client.Pages;
using OTTCreator.Client.Services;

namespace OTTCreator.Client;

public partial class AppShell : Shell
{
    ContentService contentService;

    public AppShell()
    {
        InitializeComponent();

        contentService = new ContentService();
        var task = Task.Run(GenerateUI);
        task.Wait();

        RegisterRoutes();
    }

    private async Task GenerateUI()
    {
        var types = await contentService.GetTypesAsync();

        var categoryPageDataTemplate = new DataTemplate(typeof(CategoryPage));

        foreach (var type in types)
            FavoriteContentFlyoutItem.Items.Add(new ShellContent() { Title = type, ContentTemplate =
categoryPageDataTemplate });

        foreach (var type in types)
        {
            var UIType = new FlyoutItem() { Title = type };
            if (type == "Телеканали")
                UIType.Icon = "tv_icon.png";
            else if (type == "Радіостанції")
                UIType.Icon = "radio_icon.png";
            var categories = await contentService.GetCategoriesAsync(type);
            foreach (var category in categories)
                UIType.Items.Add(new ShellContent() { Title = category, ContentTemplate = categoryPageDataTemplate });
            Shell.Items.Add(UIType);
        }
    }

    private void RegisterRoutes()
    {
        Routing.RegisterRoute("ContentItemPage", typeof(ContentItemPage));
        Routing.RegisterRoute("CategoryPage", typeof(CategoryPage));
        Routing.RegisterRoute("SupportPage", typeof(SupportPage));
    }
}
```

Рисунок 4.19 – Код AppShell.cs

Також би хотілось виділити те, що була використана можливість під’єднання пошуку по всьому вмісту шляхом наслідування SearchHandler, доповненням його об’єктами для відображення та його вбудовуванням в AppShell. Особливістю є те, що показ результатів під час кожної зміни запиту оновлюється і його можна урізноманітнити елементами графіки.

Розміщення множини ImageButton, які представляють елементи колекції вмісту, отриманого знову ж таки з ContentService, в вигляді піктограм, які здійснюють за посиланням перехід на ContentItemPage, записуючи в SecureStorage

– сховище чутливих даних в форматі «ключ-значення» ідентифікатор вмісту, який необхідно відтворити, в CategoryPage відбувається за допомогою шаблону FlexLayout, який може:

- розташовувати дочірні елементи горизонтально та вертикально в стеку;
- перенести дочірні елементи, якщо їх занадто багато, щоб помістится в один рядок або стовпець;
- керувати орієнтацією екрана та адаптуватися до різних розмірів екрана, що особливо важливо при багатоплатформності тощо.

Сторінка, яка втілює основну ідею даного застосунку, ContentItemPage – представляє собою Grid, елементи якого за допомогою властивостей ZIndex і IsVisible, з’являються чи зникають на тих чи інших планах, до якого входять:

- Image, який замінює відеоряд у вмісті без нього;
- MediaElement – компонент, який входить до складу CommunityToolkit.Maui, призначений для відтворення локальних та потокових відео- та аудіо- файлів;
- StackLayout для формування власної заміни вбудованих в MediaElement елементів керування з метою забезпечення зручності користування саме потоковими мультимедійними даними, до набору якого входять ImageButton з піктограмами та можливостями зміни співвідношення, зменшення та збільшення гучності, відтворення та зупинки, додавання в «Улюблений вміст»;
- черговий FlexLayout з метою реалізації екрана захисту вмісту від неаутентифікованих і неавторизованих користувачів, на якому знаходяться інструкція з активації, поле вводу та кнопка збереження коду доступу.

У результаті була отримана наступна архітектура програми, яку видно на рисунку 4.20:

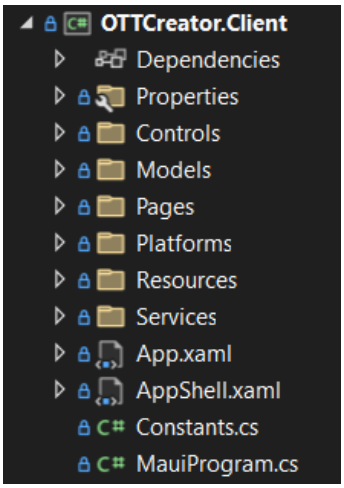

Рисунок 4.20 – Архітектура клієнтського застосунку

5 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

5.1 Веб-застосунок

Перші 2 облікові записи для адміністратора та клієнта генеруються при першому запуску веб-додатка для того, щоб внести їхні ролі в Identity. Дані входу знаходяться в файлі SeedRolesAndUsers.cs кореня проекту веб-програми.

5.1.1 Гість

Вперше переходячи за посиланням веб-аплікації, будь-який користувач є гостем, адже не аутентифікувався та авторизувався. Його зустрічає головна сторінка, фрагмент якої проілюстрований на рисунку 5.1. Її суть – коротко передати інформацію про публічні можливості всієї системи.

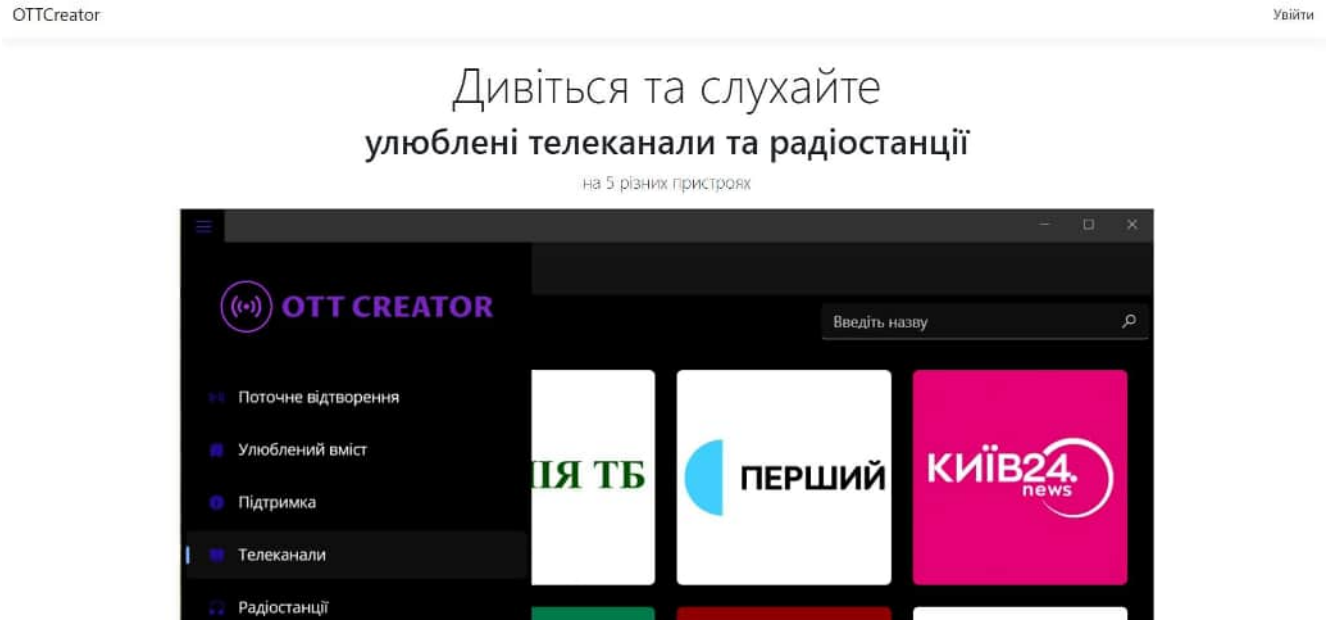

Рисунок 5.1 – Фрагмент головної сторінки веб-застосунку

Щоб отримати доступ до можливостей, які дає веб-додаток, гість повинен увійти в нього, ввівши коректні дані облікові дані, тим самим підтвердивши свою належність до всієї систем та роль у ній, шляхом натиснення відповідної кнопки в правому верхньому куті, після чого він буде переспрямований на сторінку входу, частина процесу логування на якій зображена на рисунку 5.2. Залежно від того, хто користувач - адміністратор чи клієнт, йому будуть відкриті спільні або відмінні функції.

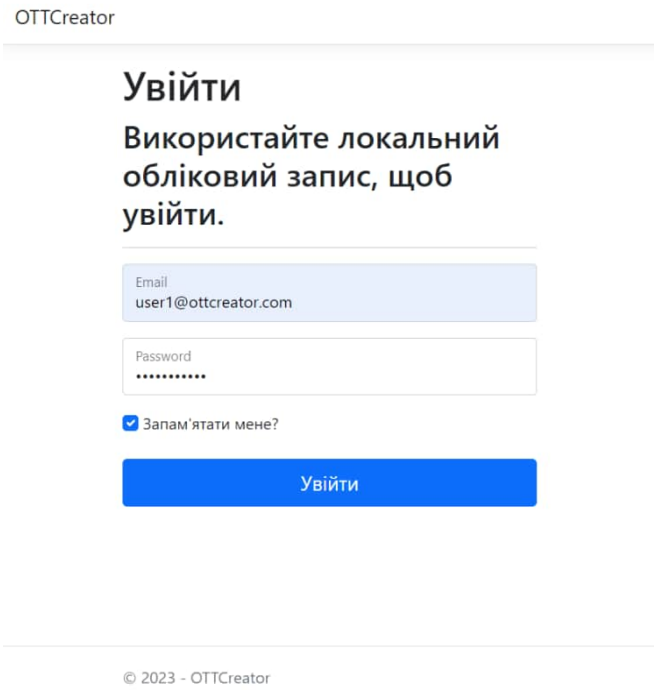

Рисунок 5.2 – Частина процесу входу в веб-додаток

5.1.2 Клієнт

Після успішної аутентифікації та авторизації клієнт знову потрапляє на головну сторінку, проте вже доповнену наданим йому функціоналом – отримання кодів доступу до його програми та перегляду стану їхнього використання, керування власним обліковим записом та вихід з веб-аплікації.

При відкриті пункту навігаційного меню «Коди та використання» з’являється таблиця з однойменними даними для 5 пристроїв, які використовуються для копіювання GUID для активації прив’язки копії клієнтського застосунку та відстеження поточного стану застосування ключа, щоб можна побачити на рисунку 5.3.

Код	Використовується
418fb98b-2341-4196-a130-57be7e9b4c0f	Так
2f986eed-aac7-4b28-8c05-712c06051378	Так
ad6da660-cf02-4fad-b6a0-ddbc43d761ba	Так
6b093e32-3f94-47e2-a6ed-f68b44e7c4cd	Ні
721818e5-2ad2-4e10-853a-06bfd4972102	Ні

Рисунок 5.3 – Частина сторінки «Коди та використання» веб-програми

Натиснувши на «Обліковий запис» клієнт переходить на сторінку «Керувати», яка відображає перший елемент бокового меню - розділ «Профіль», де він може бачити свій логін, який є копією адреси електронної пошти, та переглядати, додавати та змінювати номер телефону, що показано на рисунку 5.4. Скориставшись боковим меню та перейшовши до розділу «Пароль», користувачеві надається можливість змінити пароль шляхом введення та надсилання старого, нового та підтвердження нового паролів.

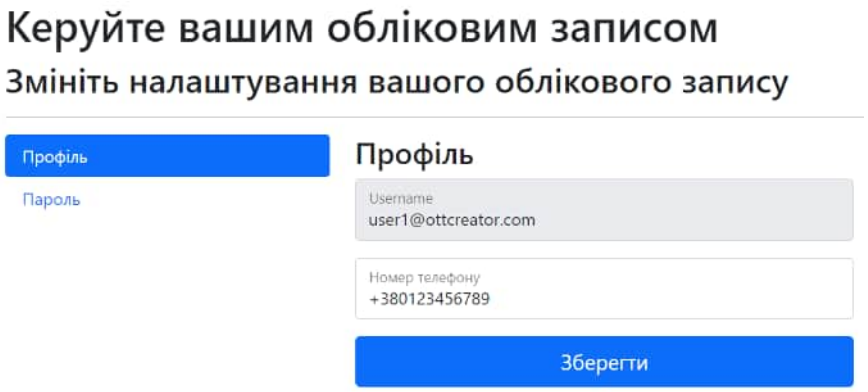

Рисунок 5.4 – Фрагмент розділу «Профіль» сторінки «Керувати» веб-аплікації

5.1.3 Адміністратор

Можливості адміністратора включають в себе як і клієнтські, так і власні: перегляд, створення, редагування та видалення вмісту та аналогічні види операцій відносно користувачів.

Перейшовши на сторінку «Вміст», адміністратор бачить таблицю з колонками назви, категорії, типу, зображення, логотипу, потоку, наявності відео з останнім стовпцем, який дає можливість переходу до представлень, відповідальних за CRUD-операції. У неї також вбудовані інструменти пошуку, фільтрації, сортування за колонками рядків та пагінації сторінок даних, частину чого видно на рисунку 5.5, а типової форми для всіх операцій, яка відрізняється текстом, кольором та дією кнопки, наприклад «Зберегти» – 5.6.

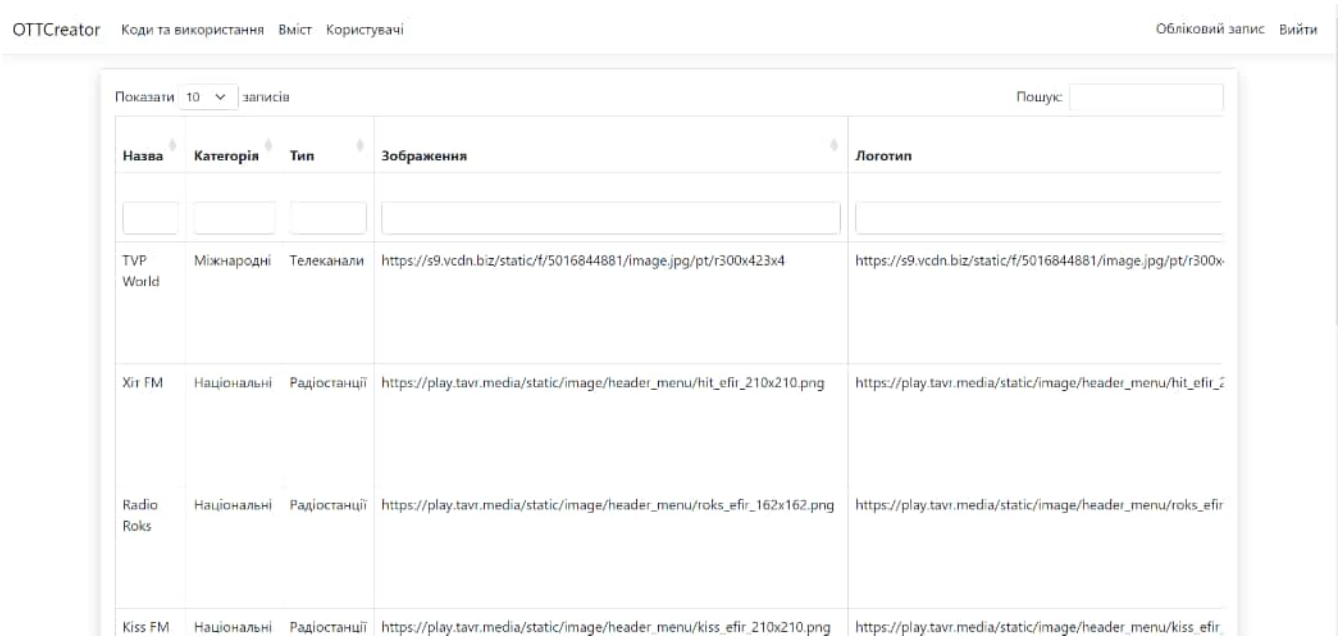

Рисунок 5.5 – Частина сторінки «Вміст» веб-застосунку

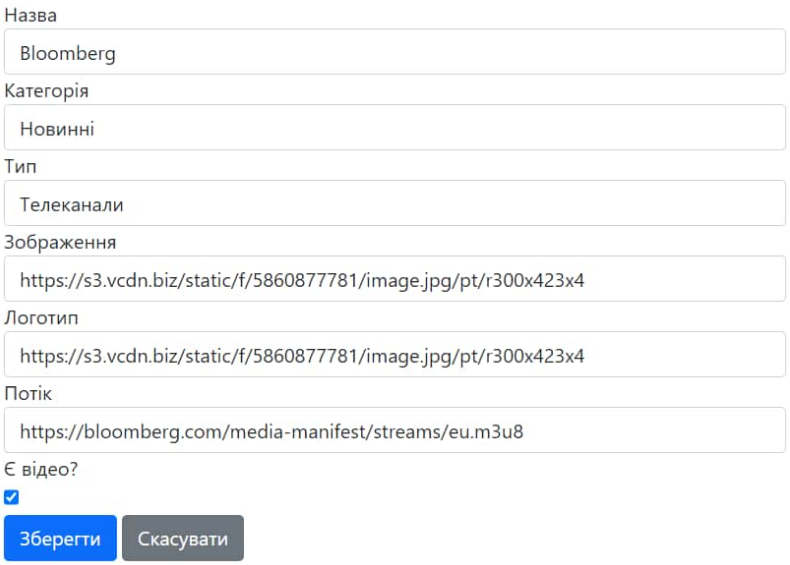

Рисунок 5.6 – Фрагмент представлення «Зберегти» сторінки «Вміст» веб-додатка

Аналогічна взаємодія відбувається при навігації до сторінки «Користувачі», де таблиця має інформативні колонки електронної пошти, номера телефону, ролі, стану оплати. Для кращого розуміння надані рисунки 5.7 і 5.8.

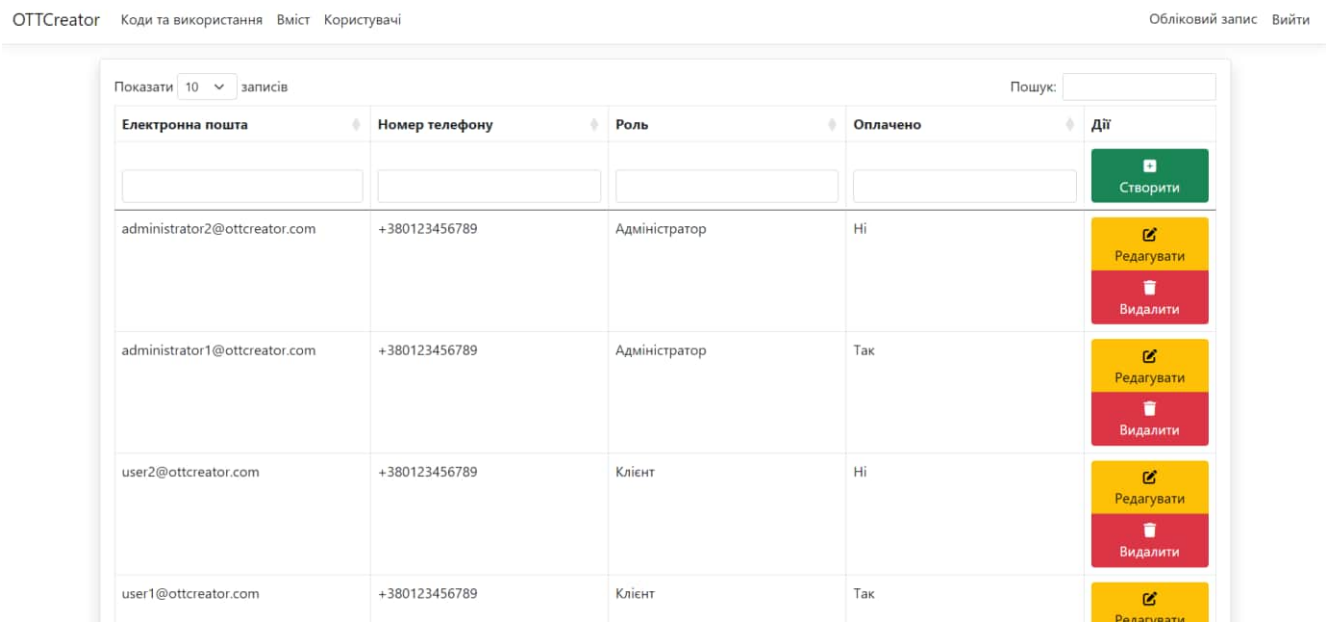

Рисунок 5.7 -Фрагмент сторінки «Вміст» веб-програми

Рисунок 5.8 - Фрагмент представлення «Зберегти» сторінки «Користувачі» веб-аплікації

5.2 Клієнтський застосунок

Клієнтський додаток працює під управлінням ОС Windows і Android.

Перед безпосереднім початком перегляду вмісту треба прив’язати копію клієнтської програми до користувача. Інструкція з реалізації цього викладена на екрані, що зображений на рисунках 5.9 та 5.10 для Windows і Android відповідно.

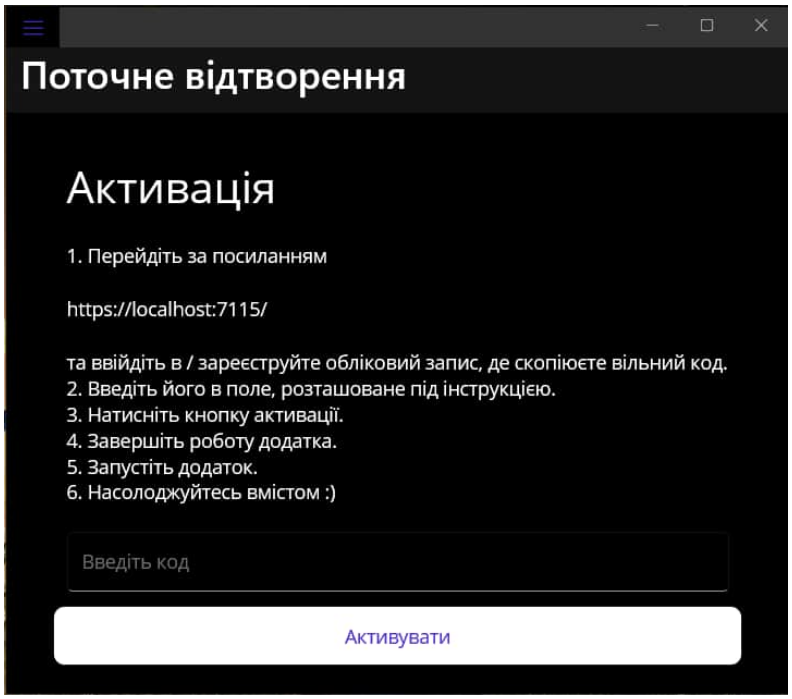

Рисунок 5.9 – Екран активації клієнтського застосунка для Windows

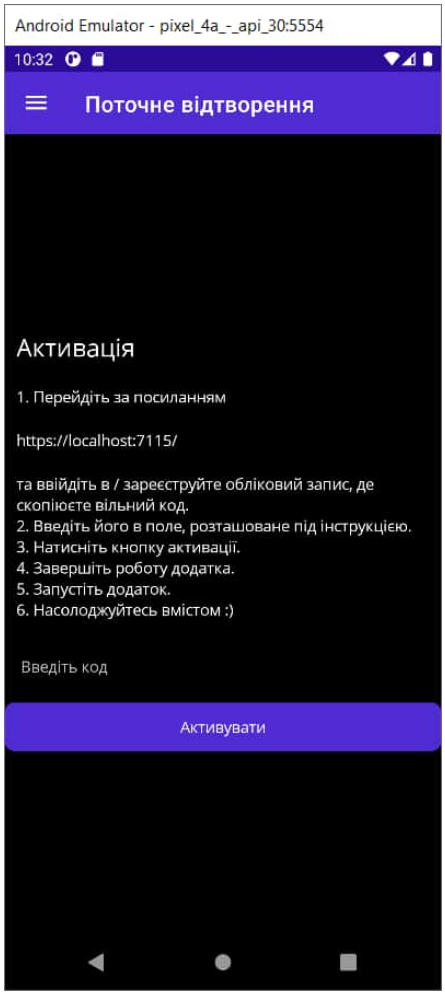

Рисунок 5.10 – Екран активації клієнтського застосунка для Android

Після успішної прив’язки пристрою до користувача, відбудеться автоматичний перехід до пункту бокового виринаючого меню «Поточне відтворення» з назвою вмісту в якості заголовка. В даному пункті можна підлаштовувати перегляд чи прослуховування вмісту до потреб клієнта шляхом натиснення один раз на вміст, внаслідок чого з’явиться панель керування, яка дозволяє змінювати співвідношення екрана між 4 режимами: по центру, заповнити, заповнити або припасувати з врахуванням початкового співвідношення, зменшувати та збільшувати гучність, починати або зупиняти відтворення, додавати в «Улюблений вміст». Останні відтворений вміст, гучність та співвідношення екрана автоматично зберігаються. Вигляд під час роботи проілюстровано на рисунках 5.11 та 5.12 для Windows і Android відповідно.

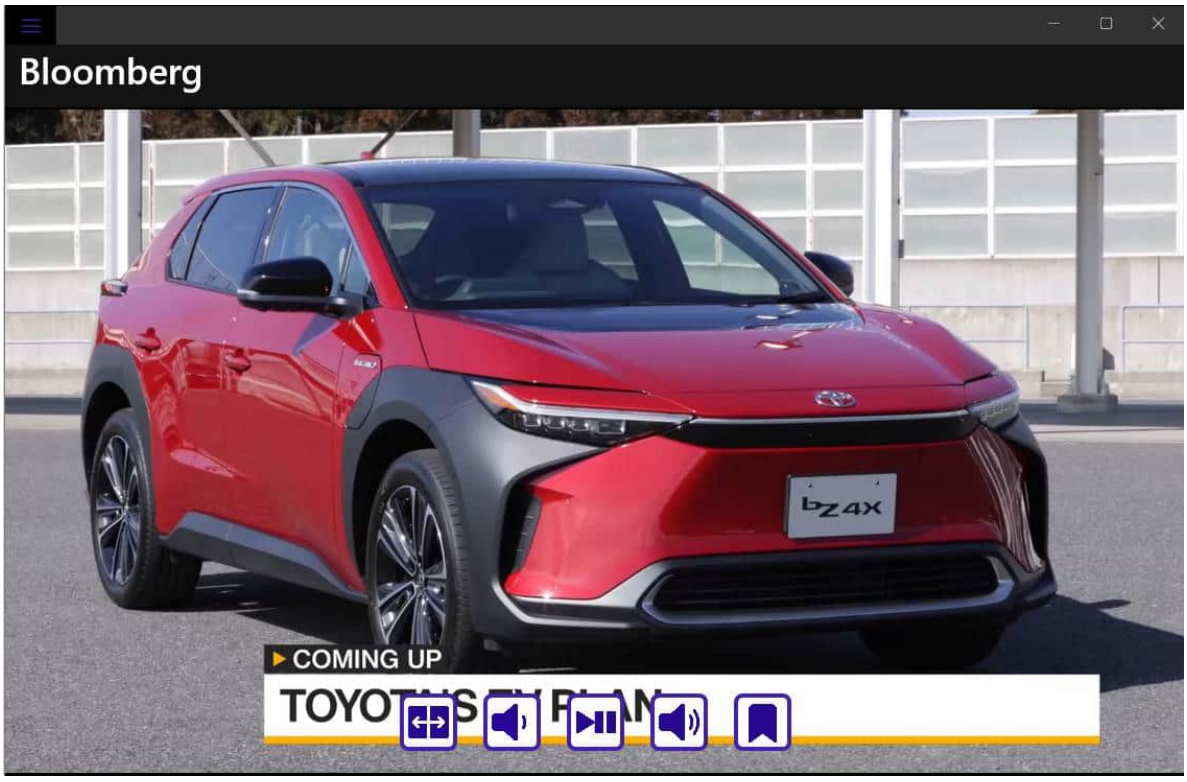

Рисунок 5.11 – Сторінка «Поточне відтворення» клієнтської програми для Windows

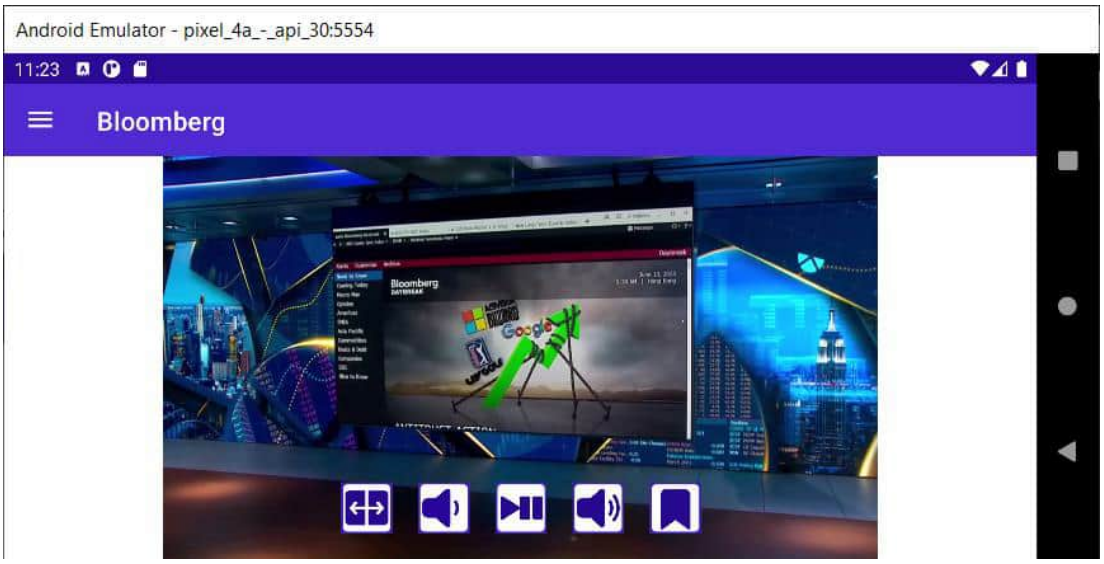

Рисунок 5.12 – Сторінка «Поточне відтворення» клієнтської програми для Android

Далі, можна ознайомитись з боковим виринаючим меню, яке містить логотип застосунку, статичні («Улюблений вміст») та динамічні типи («Телеканали» та «Радіостанції») вмісту, застереження авторських прав, натиснувши на піктограму «бургера» в верхньому лівому куті та обравши потрібний його пункт, що показано на рисунках 5.13 та 5.14 для Windows і Android відповідно.

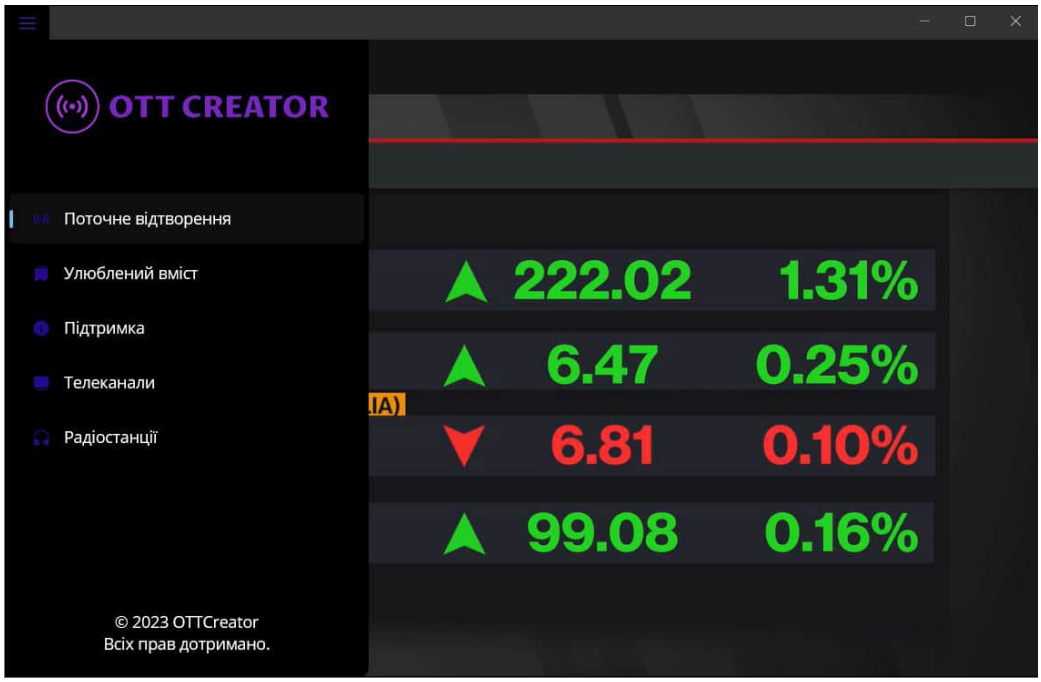

Рисунок 5.13 – Приклад відкритого бокового виринаючого меню клієнтської аплкікації для Windows

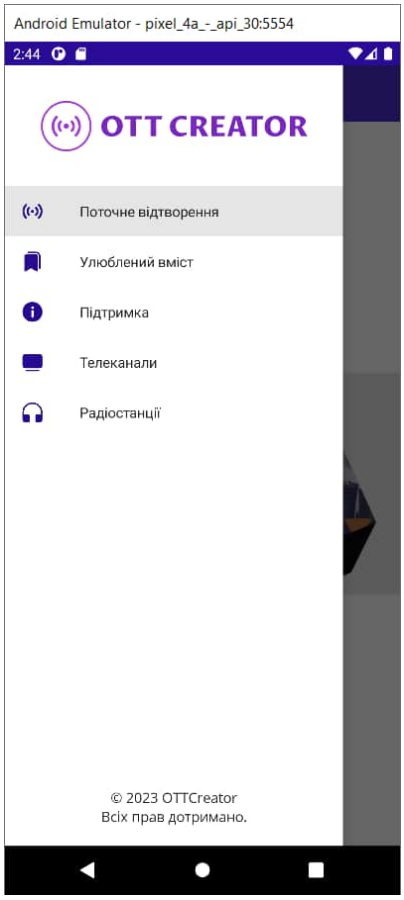

Рисунок 5.14 – Приклад відкритого бокового виринаючого меню клієнтської аплікації для Android

Якщо ми перейдемо до одного з типів, то ми потрапимо до першої категорії конкретного типу. Перемикання між іншим категоріями одного типу відбувається за допомогою вкладок, розміщених вгорі на Windows та знизу на Android. Внутрішньо, всі вони влаштовані за однаковим принципом – відображенні колекції відповідно вмісту в вигляді піктограм в адаптивному до характеристик екрана контейнері. Альтернативно, на кожній з них є вбудована можливість пошуку по всьому змісту за текстом. Приклад використання сторінки категорії ми бачимо на рисунках 5.15 та 5.16 для Windows і Android відповідно.

Майже за тим самим принципом працює пункт «Улюблений вміст», єдині відмінності роботи якого полягають в тому, що в ньому групування відбувається не за категоріями, а типами та, що стає зрозумілим з назви, він акумулюється не адміністратором, а клієнтом.

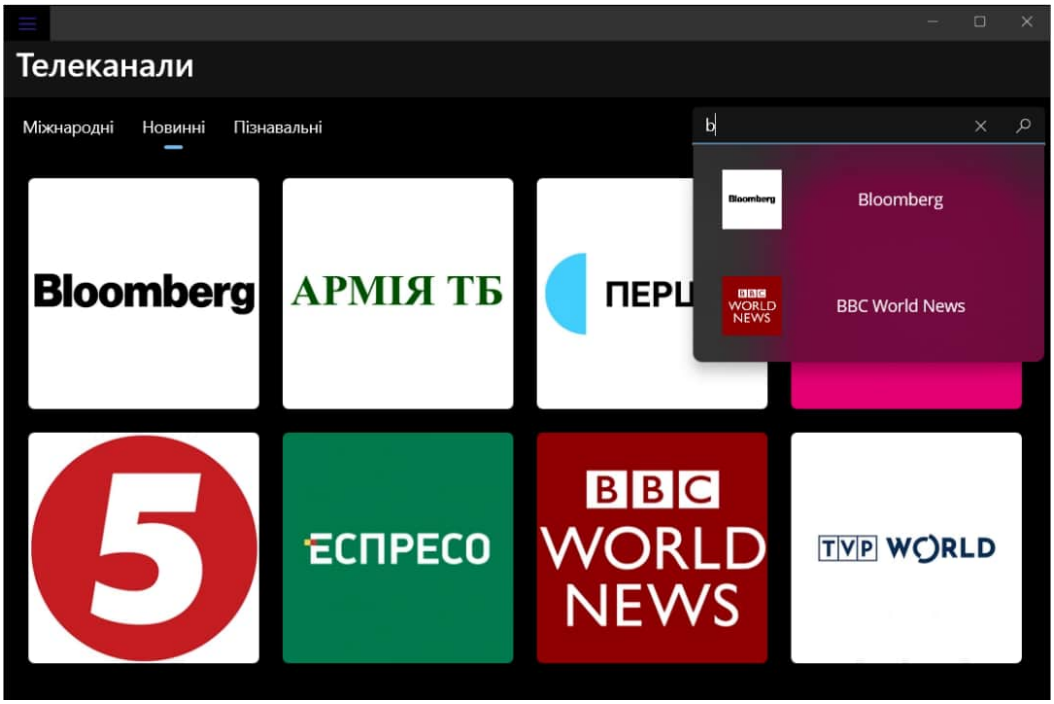

Рисунок 5.15 – Приклад використання сторінки категорії клієнтського застосунку для Windows

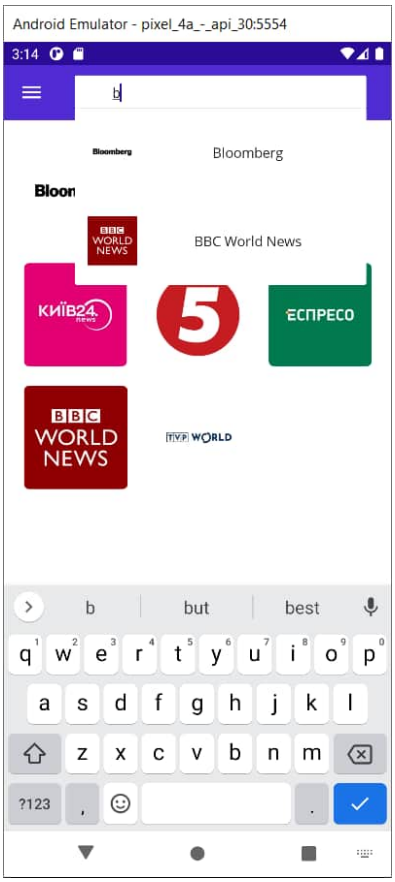

Рисунок 5.16 – Приклад використання сторінки категорії клієнтського застосунку для Android

І на останок - якщо в клієнта виникнуть труднощі в користуванні додатком, він може звернутися за контактними телефоном, адресою електронної пошти та веб-сайтом, вказаними на сторінці, доступній за пунктом меню «Підтримка», щоб йому допомогли її вирішити. Крім того, якщо виникне потреба відв’язати вказаний там же ж код доступу до даної копії клієнтської програми, то це робиться в одне натиснення кнопки «Деактивувати», після чого додаток можна буде закрити. Дизайн даної сторінки є на рисунках 5.17 та 5.18 для Windows і Android відповідно.

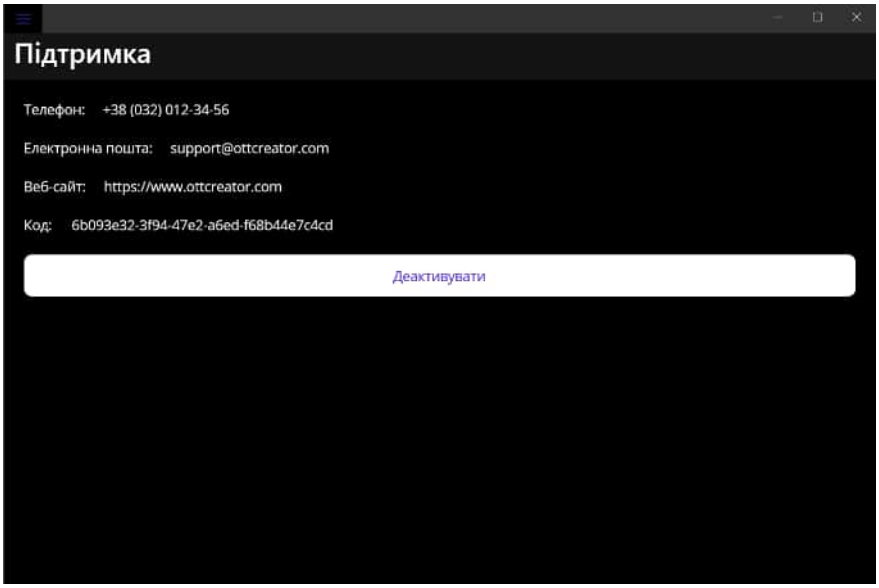

Рисунок 5.17 – Сторінка «Підтримка» в клієнтському додатку для Windows

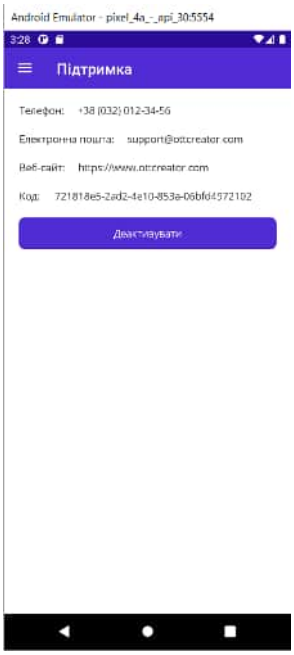

Рисунок 5.18 – Сторінка «Підтримка» в клієнтському додатку для Android

ВИСНОВКИ

У ході виконання даної роботи було поновлено попередні знання використання ASP.NET Core MVC, вивчено та проаналізовано структуру та фунціонал ASP.NET Core Web API та .NET MAUI, виділено їх основі можливості та переваги, досліджено ринок додатків, що поширюють вміст за послугою «over-the-top», сформульовано чіткі вимоги до подібних застосунків, та на основі отриманої й опрацьованої інформації поєднано існуючі, вдосконалені та нові технології .NET на прикладі побудови рішення для надання доступу до та перегляду мультимедійних інтернет-потоків даних, покращивши вміння використання Git і GitHub, спроектувавши необхідні моделі даних, згенерувавши та наповнивши початковими даними БД MSSQL за допомогою EF Core, використавши ASP.NET Core Identity та власну схему для визначення та використання користувачів та ролей в аутентифікації й авторизації в рішенні, застосувавши ASP.NET Core MVC для формування веб-застосунку для створення, редагування, видалення та перегляду даних вмісту й облікових записів користувачів, ASP.NET Core Web API для розробки прикладного програмного інтерфейсу для надсилання, обробки й отримання даних вмісту й облікових записів користувачів та .NET MAUI для реалізації багатоплатформного додатка для отримання доступу до перегляду вмісту кінцевими користувачами, забезпечивши такі важливі аспекти як зручність, простота, безпека, стабільність, швидкість і сучасність. Робота рішення випробувана на конкретних сценаріях використання додатків під управлінням Windows та Android при використанні їх гістьми, клієнтами та адміністраторами та дає коректні результати.

Повні результати побудови рішення та історію їх розробки можна переглянути на GitHub-репозиторії за посиланням: https://github.com/BorysKopytko/OTT-Creator

ПЕРСПЕКТИВИ РОЗВИТКУ

Після завершення виконання основного завдання даної роботи, у вільний час я провів додатковий аналіз ринку схожих додатків і прийшов до висновку, що можливості розробленого рішення можна значно розширити за рахунок:

- додавання нових типів вмісту, наприклад фільмів, серіалів, музики, подкастів і курсів;
- покращення існуючих типів вмісту за рахунок введення їм нових властивостей;
- додавання EPG – електронного програмного гіду;
- введення гнучкої системи рекомендацій вмісту сформованих бібліотекою машинного навчання ML.NET;
- збільшення переліку підтримуваних ОС, включивши до нього iOS, mac і Tizen;
- перенесення аутентифікації й авторизації напряму до клієнтського застосунку й створення локальної програми для адміністрування за допомогою WinUI3;
- перенесення клієнтського застосунку на MVVM – шаблон проектування, який ділить аплікацію на 3 шари, де модель – відповідає за дані, view – за представлення, а view model – служить проміжною ланкою, що суттєво полегшує оновлення даних в реальному часі за допомогою MVVM Toolkit – сучасної, швидкої, модульної бібліотеки для побудови цього;
- реалізація системи тарифів та оплати тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stack Overflow Developer Survey 2022 - [Електронний ресурс]. – 2022. – Режим доступу : https://survey.stackoverflow.co/2022/
2. WinForms vs WPF vs UWP: Expectations vs. Reality - [Електронний ресурс]. – 2022. – Режим доступу : https://blogs.embarcadero.com/winforms-vs-wpf-vs-uwp-expectations-vs-reality/
3. .NET Multi-platform App UI documentation - [Електронний ресурс]. – 2023. – Режим доступу : https://learn.microsoft.com/en-us/dotnet/maui/
4. Українські OTT: як обрати собі телебачення в інтернеті [Електронний ресурс]. / О. Білоусенко; – 2022. – Режим доступу : https://ms.detector.media/onlain-media/post/28830/2022-01-19-ukrainski-ott-yak-obraty-sobi-telebachennya-v-interneti/
5. Introduction to Identity on ASP.NET Core - [Електронний ресурс]. / R. Anderson; – 2022. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio
6. Scaffold Identity in ASP.NET Core projects - [Електронний ресурс]. / R. Anderson; – 2023. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-7.0&tabs=visual-studio
7. Custom User Management in ASP.NET Core MVC with Identity - [Електронний ресурс]. / M. Murugan; – 2022. – Режим доступу : https://codewithmukesh.com/blog/user-management-in-aspnet-core-mvc/
8. ASP.NET Core MVC with EF Core - tutorial series - [Електронний ресурс]. – 2023. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/?view=aspnetcore-7.0
9. Overview of ASP.NET Core MVC - [Електронний ресурс]. / S. Smith; – 2022. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0

10. What Is An API (Application Programming Interface)?- [Електронний ресурс]. – Режим доступу : https://aws.amazon.com/what-is/api/
11. Tutorial: Create a minimal API with ASP.NET Core - [Електронний ресурс]. / R. Anderson, T. Dykstra. – 2023. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio
12. .NET Multi-platform App UI (.NET MAUI) Community Toolkit documentation- [Електронний ресурс]. – 2022. – Режим доступу : https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/
13. Play Audio and Video in .NET MAUI apps with the new MediaElement- [Електронний ресурс]. / G. Versluis; – 2023. – Режим доступу : https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0