Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

# «Розробка веб-застосунку на ASP.NET Core»

**Текстова частина до курсової роботи**
**за спеціальністю «Комп'ютерні науки» 122**

Керівник курсової роботи
старший викладач Борозенний С.О.
_____
(підпис)
"____" _____ 2021 р.

Виконав студент Шевченко В.О.
"____" _____ 2021 р.

**Київ 2021**

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
старший викладач
_____ Борозенний С.О.
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Шевченко Владиславі

4-го курсу факультету інформатики

ТЕМА: Розробка веб-застосунку на ASP.NET Core

Вихідні дані:

-
-

Зміст ТЧ до курсової роботи:

Вступ

Анотація

1. Аналіз предметної області
2. Вибір програмного забезпечення
3. Реалізація практичної частини

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

## Календарний план виконання курсової роботи

**Тема:** Розробка веб-застосунку на ASP.NET Core

**Календарний план виконання роботи:**

| № п/п | Назва етапу дипломного проекту (роботи) | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1. | Отримання завдання на курсову роботу | листопад 2020 р. | |
| 2. | Огляд літератури за темою роботи | листопад 2020 р. | |
| 3. | Вибір програмного забезпечення для роботи | грудень 2020 р. | |
| 4. | Розробка веб-застосунку (практичної частини роботи) | січень-лютий 2021 р. | |
| 5. | Написання текстової частини роботи | березень 2021 р. | |
| 6. | Підготовка слайдів для презентації та доповіді | березень 2021 р. | |
| 7. | Надання роботи науковому керівнику для перевірки | березень 2021 р. | |
| 8. | Корегування роботи за результатами попереднього захисту. | березень 2021 р. | |
| 9. | Остаточне оформлення пояснювальної роботи та слайдів. | квітень 2021 р. | |
| 10. | Захист курсової роботи | 19-25 квітня 2021 р. | |

Студентка Шевченко В.О._____

Керівник Борозенний С.О._____

“_____” _____

# Зміст

# Вступ

## Актуальність, наукове та практичне значення обраної теми

Одним з головних трендів останніх часів в українському медіапросторі є подкасти, які збирають чимало прослуховувань та завантажувань. Якщо в англомовному світі подкасти вже давно завоювали прихильність слухачів та сформували власний ринок, то в Україні він тільки розвивається. Наприклад, у США понад половина населення постійно слухає різноманітні подкасти [1].

Хоча український ринок подкастів лише формується, за останній рік було запущено купа цікавих проєктів. Наприклад, подкасти створюють чимало відомих медіа: НВ, Українська правда, The Village тощо. Крім цього, власні подкасти запускають організації, бізнес-школи та просто активні українці. Так, можна виділити подкасти від «Києво-Могилянської бізнес-школи» (Radio KMBS) та подкаст «Інше інтерв'ю, у якому радіоведучий Володимир Анфімов бере інтерв'ю у відомих українців.

Поява низки популярних подкастів спричинила велику зацікавленість до подкастів з боку рекламодавців. Адже за дослідженням Nielsen [2], реклама у подкастах в 4,4 рази ефективніша за інші традиційні види реклами (банерну, текстову та телевізійну). Крім того, згідно з дослідженням Edison Research [1], 60% опитуваних нададуть перевагу товару саме того бренду, про який вони почули у подкасті, якщо їм дадуть два товари з однаковою ціною.

Перепоною для більшої залученості рекламодавців у рекламу в подкастах є незручність її замовлення. Наприклад, якщо бренд влаштовує рекламну кампанію та прагне замовити рекламу у кількох подкастах, то йому потрібно шукати цікаві подкасти в інтернеті, шукати контакти кожного з подкастів, зв'язуватися з ними, дізнаватися аналітику прослуховувань та інші статистичні дані, а потім ще й вести перемовини.

Ми пропонуємо спростити цей процес за допомогою онлайн-платформи з купівлі реклами у подкастах, де ми зберемо найкращі шоу України. Рекламодавець заходить на платформу, за допомогою зручного пошуку та

фільтрів обирає цікаві подкасти, переглядає всі дані про цей подкаст (тема, опис програми, автор, попередні клієнти, вартість, дати виходу, статистику прослуховувань серед аудиторії тощо) та замовляє рекламу через форму, у якій вказує рекламне повідомлення, яке він хоче, щоб прозвучало у подкасті.

Наша онлайн-платформа фокусуватиметься виключно на продажах реклами у вигляді рекламний вставок у подкаст. Тобто це буде 30-секунде або хвилинне аудіоповідомлення, яке прописує бренд, а автор подкасту зачитує його на початку або в середині випуску. Такий вид реклами обрано спеціально із урахуванням того, що він є найбільш актуальним, а також його найпростіше за все автоматизувати.

**Мета та завдання курсової роботи**

**Мета:** створення онлайн-платформи з купівлі реклами у подкасту, яка дозволить:

- оптимізувати процес купівлі реклами в українських подкастах
- зменшити час, який використовують рекламодавці для купівлі реклами
- зменшити час, який витрачають автори подкасту
- залучити більшу кількість рекламодавців
- підвищити зацікавленість до подкастів з боку рекламодавців
- сформувати ринок реклами у подкастах.

**Завдання:** розробити веб-застосунок для купівлі реклами, створити зручну систему вибору та купівлі подкастів для рекламодавців, сформувати базу даних актуальних українських подкастів, забезпечити перегляд історії замовлень адміністратором для різних подкастів.

**Об'єкт дослідження**

Опитування представників медіаринку (авторів подкасту та рекламодавців) та аналіз наявних сервісів для купівлі реклами у подкастах на іноземному ринку.

**Використане програмне забезпечення**

Для розробки веб-застосунку для купівлі реклами у подкастах була обрана технологія .NET Core з використанням мови програмування C#. Клієнт-серверну архітектуру забезпечує технологія ASP.NET Core із використанням патерну MVC (Model-View-Controller). Для зберігання даних використовується база даних на основі Microsoft SQL Server 2019. Для управління даними безпосередньо із застосунку використовується технологія Entity Framework Core. Для відображення користувацького інтерфейсу використовується Razor Pages та Bootstrap 4.

**Структура роботи**

Курсова робота містить вступ, три основні розділи, список використаної літератури та додатки.

Перший розділ складається з обгрунтування актуальності теми для дослідження, тестування наявних аналогів та аналізу їх переваг та недоліків. На основі проведеної роботи була сформульована мета та завдання розробки, перелік необхідного фунціональності для коректної роботи веб-застосунку, а також обрано програмне забезпечення для створення веб-застосунку.

У другому розділі розглядаються теоретичні питання розробки веб-застосунків на мові C# із використанням технологіях .NET Core. Крім цього, наводиться опис клієнт-серверної архітектури, патерну MVC, використаної бази даних та класифікації сайтів. У ньому також обгрунтовується вибір програмного забезпечення.

Третій розділ описує технічне завдання, вимоги до даних, структуру проєкту, безпосередньо реалізацію проєкту, розповідаються деталі розробки (написання класів та функцій), показуються скріншоти реалізації програми та тестування застосунку.

**Анотація**

Курсова робота присвячена створенню веб-застосунку для купівлі реклами у подкастах на базі платформи .NET Core від компанії Microsoft з використанням фреймворку ASP.NET Core, архітектурного патерну MVC (Model—View—Controller) та бази даних MSSQL Server.

# 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Актуальність задачі

За визначенням, подкаст — це аудіофайл, який розповсюджується на стрімінгових сервісах чи власних платформах [1]. Подкаст може включати інтерв'ю, розмовне шоу на різноманітні тематики, аудіосеріали, аудіовистави. Тобто будь-який розмовний жанр, який існує у світі. Якщо коротко, то подкасти — це новий YouTube серед аудіо.

Замість радіопередач, для яких потрібно мати радіоприймач та слухати можна лише у конкретний час, слухачі обирають подкасти, серед яких можуть обрати цікву для них тему, формат, а також слухати запис у час, коли їм зручно.

Основною перевагою подкастів над відео є зручність. Їх можна слухати в автомобілі дорогою на роботу, під час занять фітнесом чи приготуванні їжі, на прогулянці чи будь-де. Основними платформами, де можа слухати подкасти, є стандартні застосунки для iPhone («Подкасти») та Android (Google Podcasts). Також подкасти виходять на багатьох стрімінгових сервісах, найпопулярнішими серед яких є Youtube, Soundclous, Spotify тощо.

Кількість слухачів подкастів в Україні стрімко зростає, що підвищує рівень зацікавленості до цієї сфери з боку роботодавців.  Таким чином, все більше авторів подкасту потребують зрозумілого інструменту для управління рекламою у власних подкастах.

Наш веб-застосунок позбавить авторів від необхідності проведення переговорів з рекламодавцями, а рекламодавці зможуть за кілька кліків дізнатися всю цікаву для них інформацію про подкасти та придбати рекламу одразу у кількох подкастах. Отже, дана робота корисна для української медіасфери, бо спростить процес продажі, а також буде використана у комерційних цілях.

## 1.2. Аналіз наявних сервісів

Наразі українські подкасти співпрацюють з рекламодавцями виключно напрями. Це відбувається за допомогою електронної пошти чи повідомлень у Telegram або Facebook Messenger, що спричиняє труднощі. Адже для початку потрібно знайти контакти автора подкасту або відповідального за зв'язки з рекламодавцями цього подкасту. Враховуючи, що при прослуховуванні подкасту дана інформація не доступна (у найкращому випадку ми знаємо автора подкасту), рекламодавцям доводиться шукати автора у соцілальних мережах, де зв'язуватися з ним. Тож такий спосіб продажі реклами не є для нас конкурентом, адже спричиняє купу непотрібних дій для рекламодавця та потребує додаткового часу.

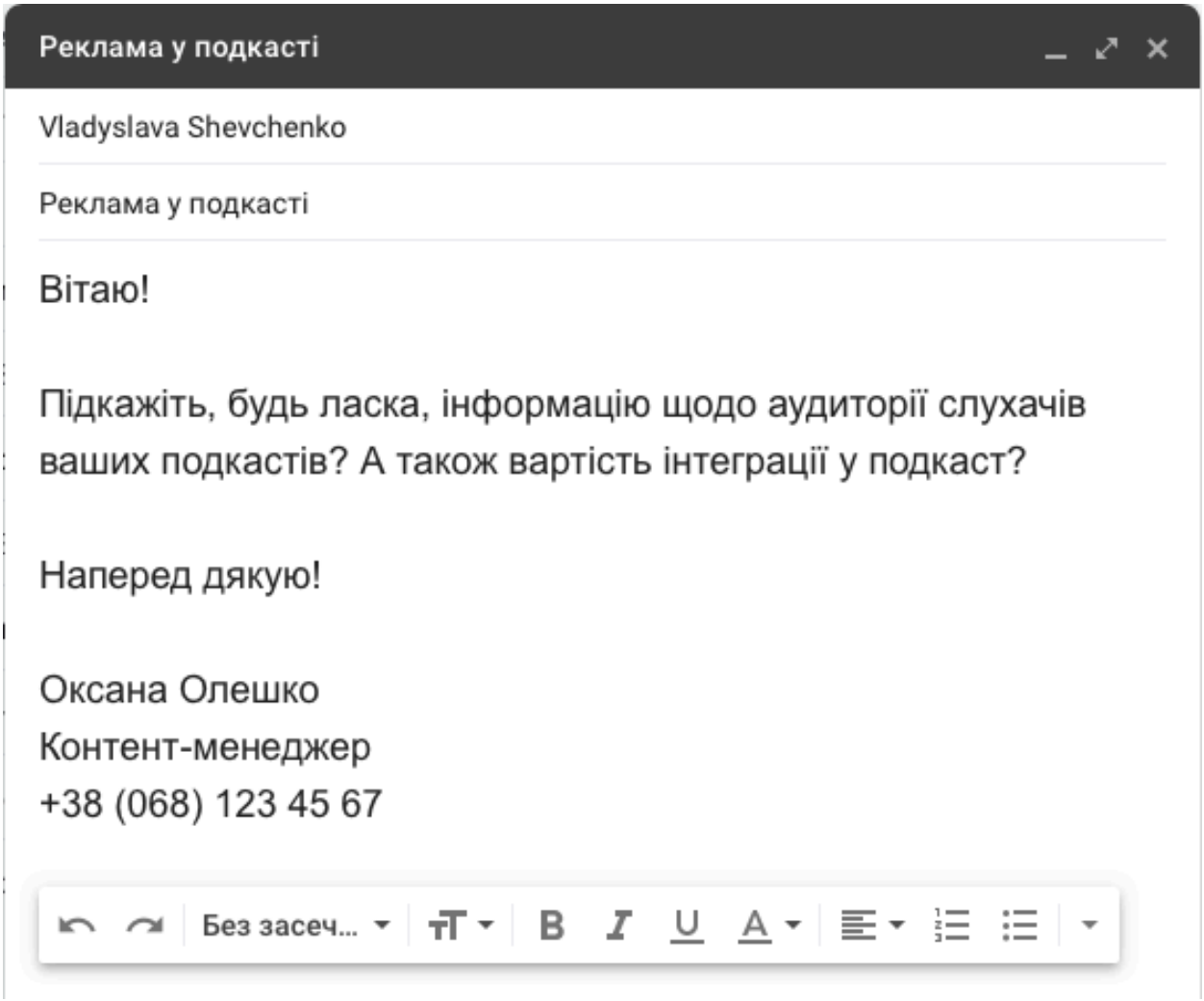


Рис. 1.2.1. Приклад листа до автору подкасту із запитом реклами

На перший погляд, конкурентом нашої платформи можуть виступати піар-агенства, до яких може звернутися бренд або компанія із завданням влаштувати рекламну кампанію у подкастах. Це зекономить час та полегшить процес для рекламодавців.

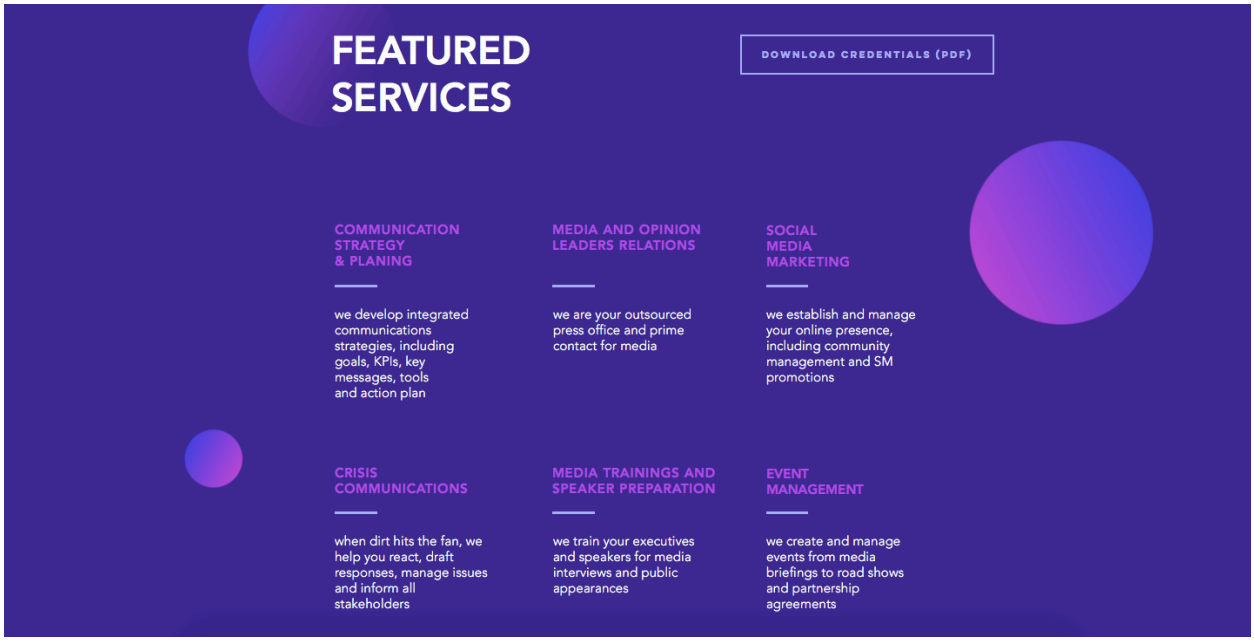


Рис. 1.2.2. Перелік послуг піар-агентства Newsfront [3]

Проте, насправді у такому випадку клієнтом вже стають рекламні піар-агенства, бо вони керують всіма рекламними проявами брендів, тож для них реклама у подкастах — це илше маленька частина завдань. Тож їм невигідно витрачати час воїх співробітників на пошук релевантних подкастів, а простіше звернутся до нашої онлайн-платформи.

Схожим чином на нашу онлайн-платформу працюють міжнародні онлайн-платформи, які пропонують рекламу у подкастах. Наприклад, сервіс Midroll [4] або AdvertiseCast [5]. Вони теж пропонують автоматизований продаж реклами у подкастах, але не є для нас конкурентами, адже працюють виключно на американському чи європейському ринку та не збираються ніяким чином заходити на український ринок.

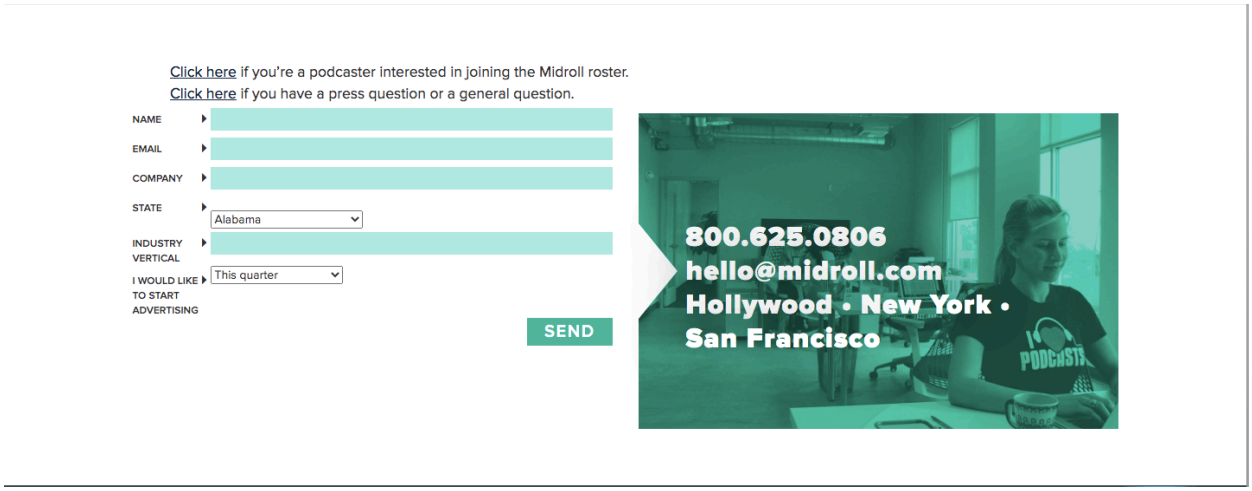


Рис. 1.2.3. Форма заповнення реклами на онлайн-платформі Midroll

Крім цього, основним фокусом для відбору подкастів у таких платформ є наявність великої аудиторії, що зменшує шанси на залучення рекламодавців у нішеві подкасти, які можуть мати меншу аудиторію, але цільову для певного бренду, що збільшить ефективність реклами у подкасту.

Наша онлайн-платформа стане єдиною платформою в Україні, яка пропонуватиме зручну та автоматизовану купівлю реклами у подкастах, як нішевих, так і популярних. Це прискорить формування ринку та залучить все більше людей до створення подкастів.

## 1.3. Постановка задачі

У веб-застосунку з купівлі реклами у подкасті мають бути реалізовані такі функції:

1. Інтерфейс веб-застосунку має бути не перевантаженим та оформлений у зрозумілій та сучасній верстці, аби увага відвідувачів була зосереджена на подкастах та оформленні замовлення.
2. Купівля реклами у подкастах може бути доступна лише для зареєстрованих користувачів.
3. Користувач має мати зрозумілий функціонал підбору подкастів: пошук потрібних подкастів, фільтрація подкастів за ціною та категорією

4. Можливість замовлення реклами у декількох подкастів одночасно

5. Можливість перегляду доступних подкастів на сайті без реєстрації

6. Зручний інтерфейс для адміністратора для додавання, редагування та видалення подкастів з бази

Проаналізувавши усі наявні сервіси, можна виокремити перелік переваг, які будуть наявні у нашого веб-застосунку:

- Онлайн-платформа дозволить рекламодавцю зможе замовити рекламу у будь-який зручний для нього час та не чекати довго відповіді від автора подкасту через email. Рекламодавець заповнює форму та отримує рекламу;
- Збирання в одному місці усіх українських подкастів, що дозволить рекламодавцю не пропустити жодної цікавої можливості для купівлі реклами;
- Інформація про усі українські подкасти допоможе відслідковувати кількість прослуховувань та аналізувати ринок українських подкастів;
- Зручний інтерфейс, який дозволить швидко та зручно знайти потрібні подкасти за допомогою фільтрів по ціні та темам;
- Економія часу для рекламодавців та авторів подкасту;
- Перегляд всієї необхідної інформації про подкаст (тема, опис програми, автор, попередні клієнти, вартість, дати виходу, статистику прослуховувань серед аудиторії тощо)
- Підвищення впізнаваності подкастів серед рекламодавців;
- Можливість отримання онлайн-консультації спеціаліста з реклами для обрання найкращих варіантів подкасту.

# 2. Теоретичні відомості про обрані технології

## 2.1. Класифікація сайтів

Інтернет та веб-застосунки — це невід'ємна частина життя людини у XXI столітті. Ми працюємо, дивимося фільми і серіали, замовляємо доставки та купуємо товари на сайтах. Враховуючи величезну кількість різноманітних сайтів, їх прийнято розрізняти за кінцевою метою створення [6].

1. Сайт-візитка — створюється для компаній з метою розповіді про власні послуги в інтернеті. Зазвичай, сайт-візитка містить кілька основних сторінок та розділів: «Про нас», «Послуги», «Вартість», «Як зв'язатися».
2. Корпоративний сайт — використовується для надання необхідної інформації про компанію для клієнтів, партнерів, медіапредставників. Сайт здебільшого містить актуальні новини та прес-релізи, детальну інформацію про компанію, перелік послуг, що надаються, перелік клієнтів. Основна мета — вигідне представлення компанії в онлайн-просторі.
3. Інтернет-магазин — сайт, який допомагає користувачу обрати та придбати необхідні товари чи послуги, а продавцю — продати їх. Зазвичай, інтернет-магазин несе відповідальність за своєчасне виконння послуг, що замовив покупець.
4. Інформаційний сайт — сайт, який розміщує новини та матеріали інформаційного характеру. Такі сайти допомагають користувачам знайти відповіді на питання, що їх цікавлять, та дізнатися останні новини світу.
5. Блог — особистий сайт конкретної людини, яка заповнює його виходячи зі своїх бажань та цілей. Як правило, містить кілька сторінок та нескладний інтерфейс.

6. Форум — сайт, на якому люди формуються у певні чати за своїми інтересами та спілкуються між собою. Таким чином, форуми мають контент, який генерують самостійно користувачі.

Враховуючи наші потреби, веб-застосунок буде схожий за принципом на онлайн-платформу з купівлі реклами (аналог онлайн-магазину).

## 2.2. Клієнт-серверна архітектура сайту

Для створення веб-застосунку була використана клієнт-серверна архітектура, яка має три основних компоненти [7]:

1. клієнт — інтерфейс, який надає можливість взаємодіяти з даними
2. сервер — програма, яка забезпечує зв'язок між клієнтом та способом зберігання даних і додаткову логіку;
3. мережа — забезпечує обмін інформацією між клієнтом та сервером

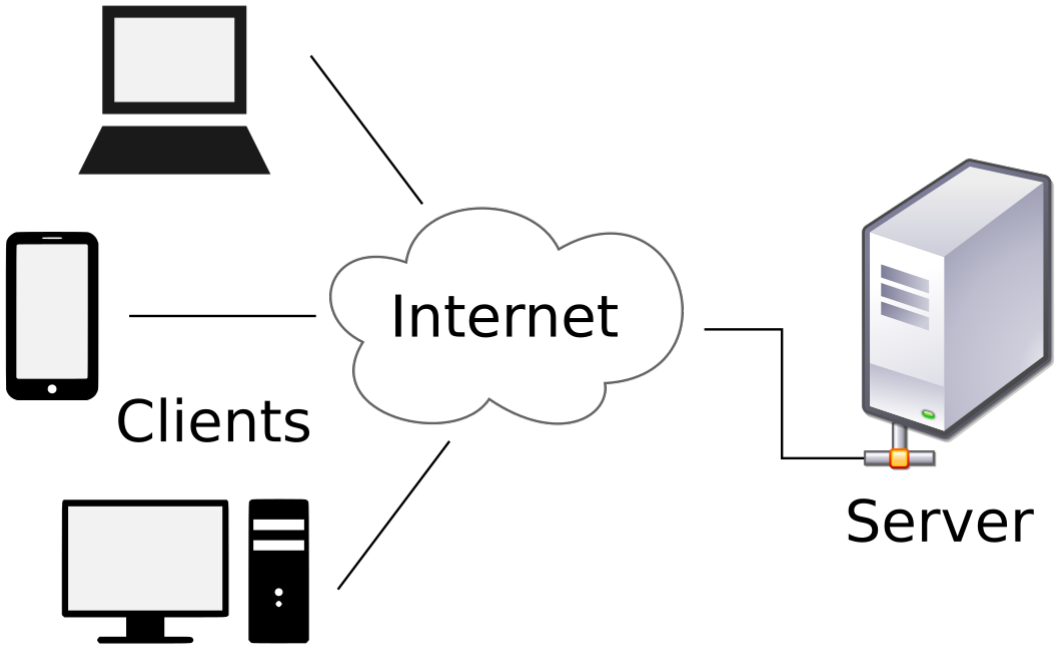


Рис. 2.2.1. Схема комп’ютерної мережі клієнтів, що спілкуються із сервером через Інтернет

## 2.2. .NET Core

Розробка веб-застосунку була реалізована за допомогою платформи .NET Core із використанням технології ASP.NET Core та мови програмування C#.

.NET Core — це модульна платформа, яку створила компанія Microsoft. Вона призначена для розробки програмного забезпечення із відкритим кодом [8].

ASP.NET Core — це фреймворк від компанії Microsoft, який використовує платформу .NET Core. Він призначений для створення різних сучасних веб-застосунків: від невеликих блогі до великих інтернет-магазинів [9].

Застосунки ASP.NET Core працюють на Windows, Mac і Linux. Він був спроектований для забезпечення оптимізованого середовища розробки для застосунків, які розгортаються в хмарі або запускаються локально.

Основні переваги ASP.NET Core:

- відкритий висхідний код;
- підтримка національних алфавітів за допомогою використання Unicode
- висока продуктивність завдяки відкомпільованому коду
- можливість запуску веб-застосунків на всіх пристроях (смартфони, планшети, ноутбуки, ПК);
- кросплатформенність — можливість роботи веб-застосунку на Windows, Linux и macOS
- інтеграція з сервісами Microsoft
- функціональність Razor Pages
- можливість застосування типових шаблонів авторизації та автентифікації користувача
- підтримка різноманітних сервісів, які дозволяють реалізувати типову логіку

Клієнт-серверну архітектуру забезпечує використанням патерну MVC (Model-View-Controller).

## 2.3. Шаблон проєктування MVC

Шаблон проєктування MVC (Model — View — Controller) був вперше описаний у 1970-х роках Трюгве Реенскаугом, який працював у дослідному центрі компаніх Xerox. Його основна мета — розділити логіку програми (Model) від її візуалізації (View) [10].

MVC має три окремих компоненти:

1. Model — клас, який реагує на команди Controller та представляє дані за запитом користувача.
2. Controller — клас, який обробляє запити з клієнта до сервера та навпаки, а також повертає дані у вигляді View.
3. View — інтерфейс, який бачить користувач (зазвичай, відображається у вигляді html-коду)

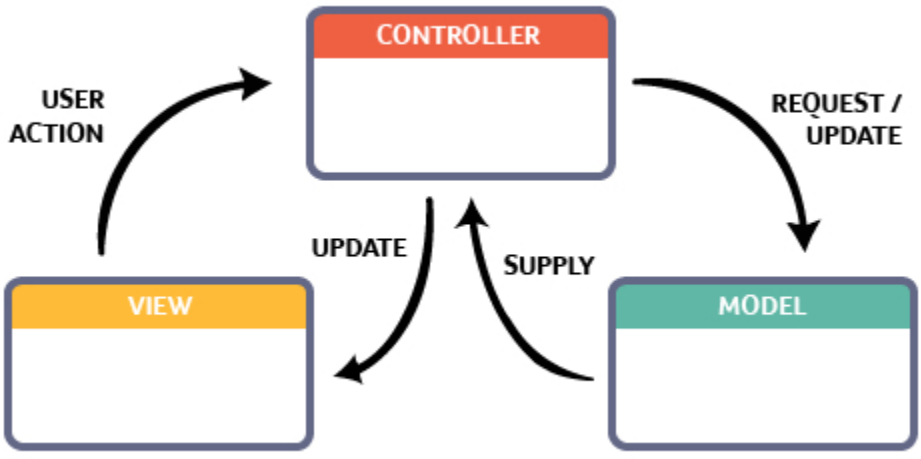


Рис. 2.3.1. Схема роботи шаблону проєктування MVC

Такий підхід дозволяє використовувати повторно код, наприклад, з іншим інтерфейсом. Також шаблон MVC допомагає виконувати такі задачі:

1. Під'єднувати кілька View до однієї Model, не змінюючи її реалізацію. Зокрема, дані можна представити у вигляді діаграми та таблиць
2. Змінювати реакції на дії користувача (натискання на клавіатурі чи ведення на тачпаді), не змінюючи View, а лише використовуючи інший Controller.
3. Тестувати окремі частини проєкту, а не увесь. Це дозволяє швидше знайти помилки та покращити якість реалізації програми.
4. Диференціювати роботу розробників логіки програми та розробників веб-інтерфейсу.

Для створення веб-застосунків із технологією ASP.NET Core найчастіше використовують мову програмування C# через такі переваги:

1. підтримка об'єктно-орієнтованого підходу;
2. мова компільованого типу;
3. наявність великої кількості бібліотек та шаблонів, написаних мовою C#;
4. наявність зрозумілої документації;
5. сувора типізація.

У якості середовища програмування було обрано Visual Studio 2019 від компанії Microsoft. Це функціональне IDE якраз для розробки застосунків на платформі .NET, яке забезпечує зручну навігацію по коду, пошук помилок, підказки щодо виправлення тощо.

Для зберігання обрано базу даних MS SQL. Microsoft SQL Server — це реляційна система управління базами даних, розроблена корпорацією Microsoft [11].

## 2.4. Entity Framework

Під час розробки веб-застосунку для взаємодії з базою даних була використана технологія Entity Framework [12]. Перевагою цієї технології є те,

що розробник працює з базою даних, використовуючи поняття сутності, об'єкта та методу замість таблиці та запитів.

Щоб застосовувати Entity Framework з MS SQL потрібно встановити бібліотеку Microsoft.EntityFrameworkCore.SQLServer.

Технологія Entity Framework Core — це обгортка реляційної бази даних, яка дозволяє представити базу даних в об'єктно-орієнтованій парадигмі. Цю технологію створила компанія Microsoft для розробників застосунків, які запускаються на платформі .NET Core.

Завдяки технології Entity Framework Core застосунки, написані з використанням платформи .NET Core, дозволяють зберігати дані, використовуючи реляційну модель побудови даних.

Реалізовуючи технологію Entity Framework Core Microsoft полегшила завдання розробникам по роботі з базою даних. Читання, зберігання, зміна, видалення та запит даних відбувається за допомогою Entity Framework.

Базу даних, яку використовує Entity Framework, представляє собою реляційну модель. Презентація даних у реляційній моделі являє собою відображення таблиць з рядками. Робота з даними реалізовується через технологію запитів, які у свою чергу реалізовуються за допомогою діалектів мови SQL. Отже, технологія Entity Framework реалізовє підхід перетворення абстрактних об'єктів у форму зберігання даних, прийнятій в реляційній моделі, а саме: рядки, таблиці та запити до них.

Одним із розширень технології Entity Framework є технологія Entity Framework Core, яка є оновленою технологією Entity Framework. Сумісність технології Entiti Framework Core та серверів баз даних покладається на версію серверу, на якому здійснюється робота з базою даних.

Реалізація підходу завдяки якому Entity Framework Core отримує дані з бази даних являє собою реалізацію SQL-команд, якіс створюють запити до серверу бази даних та отримають дані, що представляють собою об'єкти .NET.

Переваги технології Entity Framework Core:

- незалежність технологія Entity Framework Core та реалізації .NET;

- відкритий вихідний код;
- створення бази даних реалізовується написанням об'єктно-орієнтованого коду, тобто на основі абстракції класу технології Entity Framework Core самостійно створює реляційну базу даних
- технологія Entity Framework Core реалізована таким чином, що взаємодія з базою даних не залежить від реляційної СУБД;
- механізм реалізації SQL-запитів до бази даних у технології Entity Framework Core реалізований через вбудовану мову LINQ;
- підтримка запитів з параметрами та роботи з процедурами;
- реалізація CRUD-команд: create, read, update, delete;
- реалізація підтримки роботи не тільки з типами даних, які є вбудованими, але і підтримка реалізації з типами даних, які можна запрограмувати.

## 2.5. Razor Pages

Однією зі складових паттерну MVC є View, тобто візуалізація та взаємодія з даними для користувача. Технологія реалізації View у ASP.NET Core отримала назву Razor Pages [13]. Технологія Razor Pages реалізована у вигляді html-розмітки із використанням стилів (CSS, bootstrap).

Підхід, що використовується при написанні View, а саме відображення контенту за допомогою розмітки та робота з ним за допомогою backend-класу), надає перевагу в тому, що ми позбавляємося від необхідності використання прошарку сторінки, яка реалізовує модель даних.

У підсумку, внутрішня реалізації механізму рендерингу сторінки представляє як модель, так і контролер. Тобто, даний механізм реалізує забехппечує підтримку парадигми об’єктно-орієнтованого програмування: один об'єкт інкапсулює методи та дані для роботи з ним.

Отже, механізм реалізації рендерингу сторінки являє собою звичайний клас. Таким чином, ми можемо зробити висновок, що таким класом може бути

частина паттерну MVC (Model — View — Controller), яка реалізовує Controller.

Як результат, технологія Razor Pages, яку використовує ASP.NET Core, забезпечує краще рішення для реалізації клієнт-серверних затосувань, які використовують у своїй структурі шаблон програмування MVC (Model — View — Controller). Однією з переваг технології Razor Pages є те, що вона використовує у своїй структурі інтуїтивно-зрозуміле та традиційне поняття «сторінки», а не класу, який використовується у патерні MVC (Model — View — Controller), що у свою чергу ускладнює тестування та відладку прооєкту. Оскільки, реалізації Model та Controller MVC можуть бути розкидані по всьому проєкту, бо вони можуть бути розкидані по всьому проєкту.

Однією з характерних рис технології Razor Pages та ASP.NET Core MVC є те, що методи Controller, які рендерять сторінку повертають об'єкт, який по факту являє собою сторінку користувацького інтерфейсу з розширенням .cshtml. Також обов'язковою умовою є те, що метод, який рендерить дане відображення та саме відображення повинні мати однакову назву, тобто:

```
public IActionResult index ()
{
return View ();
}
```

буде повертати сторінку відображення index.cshtml. Оскільки, ASP.NET Core MVC використовує строгу типізацію, то частина патерну MVC, а саме View (представлення) майже завжди буде строго типізуватися моделлю, яку приймає або повертає метод Controller. Тобто, метод index контролера буде повертати модель:

```
public IActionResult index ()
{
//some code
return View (Model);
}
```

якою типізується View (представлення), тобто у файлі index.cshtml буде такий

рядок:

@Model Model

Також однією з рис технології Razor Pages є можливість написання коду об'єктно-орієнтованою мовою безпосередньо у розмітці представлення.

```
<div>
    @foreach(var el in Model)
   {<h2>@el.Name</h2>}
</div>
```

Для фронтенд-частини, яка реалізовується у паттерні MVC частиною View було використано безкоштовний фреймоврк з відкритим кодом Bootstrap. Сам фреймворк являж собою вже написані, готові до використання шаблони html-розмітки зі стилями CSS та підтримки скриптів, написаних мовою JavaScript. Фактично фреймворк Bootstrap являє собою готові класи, які можуть бути використані для окремих компонентів таких, як кнопки, форми, посилання, картки та багато іншого. Завдяки підходу розділення розмітки на окремі компоненти фреймворк Bootstrap пришвидшує розробку фронтенд-частини застосунку в кілька разів. Оскільки підхід розділення представлення на окремі компоненти являє собою цілком логічну та інтуїтивну форму написання сторінок розмітки.

# 3. Реалізація практичної частини

## 3.1. Опис технічного завдання на розробку

Уважно вивчивши опис необхідних функцій для веб-застосунку, можна перейти до складання технічного завдання. Веб-застосунок з купівлі реклами у подкастах розробляється для двох груп користувачів: покупців реклами та адміністратора платформи. Таким чином, веб-застосунок має містити три інтерфейси для різних груп користувачів:

- адміністратор онлайн-платформи;
- зареєстрований користувач (рекламодавець);
- користувач онлайн-платформи без реєстрації

Кожна група користувачів має окремий набір функціональностей. Належність до групи користувачів визначається за допомогою логіна та пароля.

Менеджер онлайн-платформи може:

- додавати нові подкасти на онлайн-платформу;
- редагувати інформацію про подкасти;
- видаляти неактуальні подкасти;
- бачити усі доступні подкасти для реклами;
- переглядати усю необхідну інформацію про подкасти;
- шукати подкасти на сайті за назвою, темою чи ціною;
- переглядати усі замовлення реклами на подкасти.

Рекламодавець, який зареєструвався на платформі, може:

- бачити усі доступні подкасти для реклами;
- переглядати усю необхідну інформацію про подкасти;
- шукати подкасти на сайті за назвою, темою чи ціною;
- оформлювати замовлення на рекламу у подкасті;
- прописувати рекламне повідомлення, яке має прозвучати у подкасті.

Користувач онлайн-платформи без реєстрації може:

- бачити усі доступні подкасти для реклами
- переглядати усю необхідну інформацію про подкасти
- шукати подкасти на сайті за назвою, темою чи ціною.

Функція оформлення замовлення реклами доступна тільки рекламодавцям, які зареєструвалися на онлайн-платформі. Щоб зареєструватися, користувачу потрібно ввести email та пароль (реалізована валідація, якщо email існує у базі даних, то користувачу виведеться відповідне повідомлення).

Вкладка з інформацією про подкасти повинна мати такий перелік даних:

1. Обкладинка подкасту
2. Назва подкасту
3. Автор подкасту
4. Тема подкасту
5. Опис подкасту
6. Тривалість подкасту
7. Дні та терміни виходу подкасту
8. Кількість прослуховувань та завантажувань
9. Аудиторія подкасту
10. Клієнти, які замовляли подкаст раніше
11. Посилання на прослуховування подкасту
12. Вартість

Онлайн-платформа має працювати у будь-якій операційній системі користувача на будь-якому браузері. Щоб мати можливість користуватися веб-застосунком, користувачу потрібно мати доступ до Інтернету.

## 3.2. Структура проєкту

Зазвичай, проєкти, створені за допомогою фреймворку ASP.NET Core MVC складається з:

1. **Два основні файли під назвою Program.cs та Startup.cs:**

- Program.cs — клас, який містить метод Main та є точкою входу у

програму, яка написана на ASP.NET Core MVC. Цей клас створює інстанс IWebHost, на якому розміщується веб-застосунок;

- Startup.cs — клас, який викликається на рівні застосунку після виконання файлу Program.cs. Startup.cs обробляє конвеєр запитів та підкключає сервіси до застосунку, які необхідні для коректного функціонування застосунку та виконання коректної логіки застосунку.

2. **Чотири стандартні папки:**

- **Models** — папка, у якій знаходяться моделі;
- **Controllers** — папка, у якій знаходяться усі контролери проєкту. Тобто .cs класи, які своїми методами рендерять відображення (файл .cshtml);
- **Views** — папка, у якій зберігаються представлення. Так, як патерн MVC використовує у своїй основі контролери та представлення, то папка Views, крім загальних представлень, зберігає у своїй основі папки, які зберігають представлення для кожного окремого контролеру. Тобто, HomeController.cs з папки Controllers буде використовувати представлення, які знаходять у папці Views/Home. Папка Shared зазвичай зберігає основний шаблон _Layout.cshtml, який використовується як каркас для всіх представлень на цій сторінці. Тобто, в даний шаблон рендериться представлення з інших контролерів. Сам по собі цей шаблон не має контролерів. Представлення _ViewImports.cshtml слугує як представлення для загальних імпортів шляхів, залежностей та сервісів, що підключаються. Представлення _ViewStart.cshtml визначає, який шаблон буде основним.
- **Wwwroot** — папка, яка використовується для підключення та зберігання статичних файлів (стилі, скріпти, зображення);

Здавалося, що цього достатньо для створення застосунку на ASP.NET Core MVC, проте це не так. Застосування додаткової логіки, робота з даними та обмеження доступу до окремих частин проєкту примушує нас створювати додаткові розбиття (папки) задля структуризації застосунку та простішого його розуміння і подальшого розвитку.

Таким чином, нагромадження додаткових папок (розбиття папок) є необхідністю під час створення проєкту. Розглянемо детальніше наші папки:

- **Areas** — папка, у якій розміщуються області. У даному випадку для створення захищеної області адміністратора AdminArea використовується папка Areas. Вона включає в себе папку Admin, яка є областю адміністратора. У свою чергу папка Admin реалізовує патерн MVC та включає окремі Models та Views, що використовуються адміністратором та реалізовуються в папках Controllers та Views;
- **Domain** — папка, яка слугує для зберігання сутностей, класів для CRUD-операцій (create, read, update, delete), а також класів для роботи з базою даних. Розбиття папки Domain складається з папок:

  - **Entities**: сутності (.cs класи) з полями, які представляють таблиці та рядки у реляційній базі даних відповідно;

  - **Repository**: папка виконує зберігання інтерфейсів та реалізації цих інтерфейсів по роботі з CRUD-операціями з базою даних.

    - **Abstract**: папка слугає для зберігання інтерфейсів по роботі з CRUD-операціями. Кожний інтерфейс відповідає оголошенню CRUD-операцій для однієї єдиної моделі

    - **EntityFramework**: папка, яка зберігає інтерфейси з папки Abstract

  Підхід до розділення логіки на інтерфейс та реалізацію є одним з головних принципів об'єктно-орієнтованого програмування, а саме інкапсуляції. Тобто, навіть при написанні веб-застосунку зберігається об'єктно-орієнтований підхід. Окрім папок Entities та Repository папка Domain включає два класи:

  - **ApplicationDbContext.cs**: один з основних класів у нашому застосунку на рівні з Program.cs та Startup.cs. Слугує для створення бази даних, взаємодії з нею та створення таблиць цієї бази на основі моделей.

  - **DataManager.cs:** клас, який є обгорткою до реалізації

інтерфейсів з папки Repositiry/Abstract. Використовується для CRUD-операцій

- **Migrations** — папка, яка слугує для зберігання міграцій. Тобто, кожен раз під час внесення змін до бази даних відбувається міграція. Щоб зробити міграцію, необхідно відкрити консоль та ввести команду:
  >>Add-Migration "назва міграції"
  >>Update Database
  Таким чином, коли ми робимо першу міграцію, у каталозі проєкту автоматично створюється папка Migration. Отже, при кожній наступній міграції в папку Migrations буде додаватися дана міграція, що дозволяє переглядати історію міграцій (зміну в базу даних) та за необхідності повернути попередній стан бази даних (зробити відкат)
- **Service** — необхідність створення логіки, яка реалізовує захищену область адміністратора вимагає створення домовленості, яка робить область адміністратора захищеною. Саме ця логіка реалізована в папці Service, а саме в класі AdminAreaAutorization.cs.

## 3.3. Розробка бази даних

База даних веб-застосунку з купівлі реклами у подкастах складається з шести таблиць.

### 3.3.1. Реалізація бази даних

Таблиця **Podcast** містить детальну інформацію про подкаст, яка може бути корисна для рекламодавця.

| Поле | Тип | Опис |
|---|---|---|
| id | uniqueidentifier | Унікальний ідентифікатор в базі даних |

| | | |
|---|---|---|
| PathToPhoto | nvarchar(MAX) | Титульна картинка |
| PodcastName | nvarchar(MAX) | Назва подкасту |
| PodcastAuthor | nvarchar(MAX) | Автор подкасту |
| PodcastDuration | nvarchar(MAX) | Тривалість подкасту |
| PodcastPremierDate | nvarchar(MAX) | Дні виходу подкасту |
| PodcastTopic | nvarchar(MAX) | Тема подкасту |
| PodcastDescription | nvarchar(MAX) | Опис подкасту |
| PodcasrListenedAmount | nvarchar(MAX) | Кількість слухачів подкасту |
| PodcastClients | nvarchar(MAX) | Замовники подкасту |
| PodcastAudience | nvarchar(MAX) | Цільова аудиторія |
| PodcastLink | nvarchar(MAX) | Посилання на подкаст |
| PodcastPrice | float | Ціна подкасту |

Таблиця **Orders** містить інформацію про замовлення реклами.

| Поле | Тип | Опис |
|---|---|---|
| id | uniqueidentifier | Унікальний ідентифікатор в базі даних |
| Name | varchar(MAX) | Ім'я замовника |
| Surname | varchar(MAX) | Прізвище замовника |
| Phone | varchar(MAX) | Номер телефону |
| Email | varchar(MAX) | Електронна пошта |
| OrderTime | datetime2(7) | Час замовлення |
| OrderNumber | uniqueidentifier | Номер замовлення |

Таблиця **ShopCartItems** містить інформацію про одинарне замовлення реклами.

| Поле | Тип | Опис |
|---|---|---|
| id | uniqueidentifier | Unique identifier in database |
| PodcastItem | uniqueidentifier | Унікальний ідентифікатор подкасту |
| Price | float | Вартість реклами |

| | | |
|---|---|---|
| AdDuration | varchar(MAX) | Тривалість конкретної реклами, яку замовляє рекламодавець |
| AdMessage | varchar(MAX) | Текст реклами |
| AdMonth | varchar(MAX) | Місяць, у кому має вийти реклама |
| ShopCartId | varchar(MAX) | Номер замовлення |

Таблиця **OrderDetails** містить інформацію про деталі замовлення.

| Поле | Тип | Опис |
|---|---|---|
| id | uniqueidentifier | Unique identifier in database |
| OrderId | uniqueidentifier | Унікальний ідентифікатор замовлення |
| ShopCartId | | |
| PodcastItemId | uniqueidentifier | Унікальний ідентифікатор подкасту |
| AdDuration | varchar(MAX) | Тривалість конкретної реклами, яку замовляє рекламодавець |
| AdMessage | varchar(MAX) | Текст реклами |
| AdMonth | varchar(MAX) | Місяць, у кому має вийти реклама |
| Price | float | Вартість реклами |

Таблиця **AspNetUsers** містить інформацію про користувача.

| Поле | Тип | Опис |
|---|---|---|
| Id | nvarchar(450) | Unique identifier in database |
| UserName | nvarchar(450) | Ім’я користувача |
| NormalizedUserName | nvarchar(450) | Upper Case user name |
| Email | nvarchar(450) | Пошта користувача |
| NormalizedEmail | nvarchar(450) | Upper Case user name |
| EmailConfirmed | bit | Чи підтверджено пошту |
| PasswordHash | varchar(MAX) | Кешований пароль користувача |
| PhoneNumber | varchar(MAX) | Номер телефону користувача |
| PhoneNumberConfirmed | bit | Чи підтверджено номер |

| | | |
|---|---|---|
| | | телефону |
| TwoFactorEnabled | bit | Чи є двофакторна авторизація |
| LockoutEnd | datetimeoffset(7) | Коли закінчується блокування акаунту |
| LockoutEnabled | bit | Чи заблоковано |
| AccessFailedCount | bit | Заборона доступу до акаунту |

Таблиця **AspNetRoles** містить інформацію про права доступу користувачів.

| Поле | Тип | Опис |
|---|---|---|
| Id | nvarchar(450) | Unique identifier in database |
| Name | nvarchar(256) | Ім'я ролі |
| NormalizedName | nvarchar(256) | Upper Case user name |

Таблиця **AspNetUserRoles** містить інформацію про те, яку роль займає користувач.

| Поле | Тип | Опис |
|---|---|---|
| UserId | nvarchar(450) | ID користувача |
| RoleId | nvarchar(450) | ID ролі |

## 3.3. Функціональність сайту

### 3.3.1. Сторінка «Подкасти»

Сторінка «Подкасти» реалізовується методом Podcasts контролера Home. Метод Podcasts реалізований для двох видів запиту: get-запитів та post-запитів. Get-варіація методу Post повертає список усіх подкастів з бази даних. Добування подкастів з бази даних відбувається завдяки **DataManager**.

```
private readonly ILogger<HomeController> _logger;
    private readonly DataManager dataManager;
    public HomeController(ILogger<HomeController> logger, DataManager dataManager)
    {
```

```
        _logger = logger;
        this.dataManager = dataManager;
    }
```

Post-варіація методу Podcasts отримує **SearchFiltersViewModel** і її повертає. Тобто, вона повертає список усіх подкастів з урахуванням фільтрів.

```
public class SearchFiltersViewModel
  {
    public List<PodcastItem> Podcasts { get; set; }
    public string SearchString { get; set; }
    public string PodcastCategory { get; set; }
    public string PodcastPrice { get; set; }
  }
```

При натисканні кнопки «Детальніше» метод Details контролеру Home отримує на вхід модель PodcastItem і повертає у форму дану модель, отримаючи її з бази даних за допомогою Id моделі PodcastItem.

```
public class PodcastItem
  {
    [Required]
    public Guid Id { get; set; }
    [Display(Name = "Шлях до титульної картинки")]
    public string PathToPhoto { get; set; }
    [Display(Name = "Назва подкасту")]
    public string PodcastName { get; set; }
    [Display(Name = "Автор подкасту")]
    public string PodcastAuthor { get; set; }
    [Display(Name = "Тривалість подкасту")]
    public string PodcastDuration { get; set; }
    [Display(Name = "Дата виходу подкасту")]
    public string PodcastPremierDate { get; set; }
    [Display(Name = "Тема подкасту")]
    public string PodcastTopic { get; set; }
    [Display(Name = "Опис подкасту")]
```

```
    public string PodcastDescription { get; set; }
    [Display(Name = "Кількість слухачів подкасту")]
    public string PodcasrListenedAmount { get; set; }
    [Display(Name = "Замовники подкасту")]
    public string PodcastClients { get; set; }
    [Display(Name = "Цільова аудиторія")]
    public string PodcastAudience { get; set; }
    [Display(Name = "Посилання на подкаст")]
    public string PodcastLink { get; set; }
    [Display(Name ="Ціна подкасту")]
    public double PodcastPrice { get; set; }
}
```

### 3.3.2. Головна сторінка та контакти

Головна сторінка сайту реалізовується методом **Index** контролера Home.

```
public IActionResult Index()
    {
        return View();
    }
```

Цей метод повертає представлення Index.

Сторінка «Контакти» реалізовується методом **Contacts** контролера Home.

```
public IActionResult Contacts()
    {
        return View();
    }
```

Цей метод повертає представлення Contacts.

### 3.3.3. Реєстрація та авторизація

Функції «реєстрація» та «авторизація» реалізовані методом Register та

Login відповідно, які у свою чергу мають реалізацію для get-запитів та post-запитів.

Дана get-варіація методу Register повертає представлення (форму для реєстрації).

```
public IActionResult Register()
    {
       return View();
    }
```

Розмітка форми **RegisterViewModel** представлена нижче

```
@model RegisterViewModel
<div class="card m-auto col-md-7 border-0">
  <div class="card-body">
    <h2>Реєстрація нового користувача</h2>
    <form method="post" asp-controller="Account" asp-action="Register">
      <div asp-validation-summary="ModelOnly"></div>
      <div class="form-floating">
        <label asp-for="Email"></label><br />
        <input class="form-control" asp-for="Email" />
        <span asp-validation-for="Email"></span>
      </div>
      <div class="form-floating mt-2">
        <label asp-for="Password"></label><br />
        <input class="form-control" asp-for="Password" />
        <span asp-validation-for="Password"></span>
      </div>
      <div class="form-floating mt-2">
        <label asp-for="PasswordConfirm"></label><br />
        <input class="form-control" asp-for="PasswordConfirm" />
        <span asp-validation-for="PasswordConfirm"></span>
      </div>
      <div class=" mt-1">
        <input class="btn btn-primary" type="submit" value="Реєстрація" />
      </div>
```

```
      </form>
   </div>
</div>
```

Це post-запит для методу Register.

```
public async Task<IActionResult> Register(RegisterViewModel model)
      {
         if (ModelState.IsValid)
         {
            IdentityUser user = new IdentityUser { Email = model.Email, UserName =
model.Email};
            // додаємо користувача
            var result = await userManager.CreateAsync(user, model.Password);
            if (result.Succeeded)
            {
               // встановлюєму кукі
               await signInManager.SignInAsync(user, false);
               return RedirectToAction("Index", "Home");
            }
            else
            {
               foreach (var error in result.Errors)
               {
                  ModelState.AddModelError(string.Empty, error.Description);
               }
            }
         }
         return View(model);
      }
```

Даний метод приймає **RegisterViewModel**.

```
public class RegisterViewModel
   {
      [Required]
      [Display(Name = "Логін")]
```

```
    public string Email { get; set; }

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

    [Required]
    [Compare("Password", ErrorMessage = "Паролі не збігаються")]
    [DataType(DataType.Password)]
    [Display(Name = "Підтвердіть пароль")]
    public string PasswordConfirm { get; set; }
  }
```

І за допомогою засобів бібліотеки **Identity** додає нового користувача.

```
private readonly UserManager<IdentityUser> userManager;
    private readonly SignInManager<IdentityUser> signInManager;
    public AccountController(UserManager<IdentityUser> userMgr,
SignInManager<IdentityUser> signinMgr)
    {
      userManager = userMgr;
      signInManager = signinMgr;
    }
```

Авторизація відбувається схожим чином, але замість RegisterViewModel використовує **LoginViewModel**.

```
public class LoginViewModel
  {
    [Required(ErrorMessage ="Поле логін необхідне")]
    [Display(Name = "Логін")]
    public string UserName { get; set; }

    [Required(ErrorMessage = "Поле пароль необхідне")]
    [UIHint("password")]
```

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

    [Display(Name = "Запам'ятати мене?")]
    public bool RememberMe { get; set; }
  }
```

### 3.3.4. Корзина

При натисканні кнопки «В корзину» викликається метод **ShopCartDetails** контролера ShopCart. Даний метод отримує модель типу PodcastItem і повертає модель типу ShopCartViewModel

```
public IActionResult ShopCartDetails(PodcastItem pd)
    {
      return View(new ShopCartViewModel { Podcast =
dataManeger.podcasts.GetPodcastItem(pd.Id) });
    }
```

При натисканні кнопки «Додати в корзину» викликається контролер ShopCart, а саме метод **AddToCart**, який отримує ShopCartViewModel та Id.

```
public RedirectToActionResult AddToCart(ShopCartViewModel pd,Guid Id)
    {
      pd.Podcast = dataManeger.podcasts.GetPodcastItem(Id);
      if (pd != null)
      {
        shopCart.AddToShopCart(pd);
      }

      return RedirectToAction("Index");
    }
```

За допомогою DataManager додає дані в корзину та виконує переадресацію

на метод Index контролеру ShopCart. Тобто, ми автоматично переходимо у корзину, у якій метод **Index** контролеру ShopCart повертає усі товари у корзині.

```
public ViewResult Index()
    {
      var items = shopCart.ShopCartItemsList();
      shopCart.shopCartItems = items;
      return View(shopCart);
    }
```

При натисканні кнопки «Замовити» викликається метод **CheckOut** контролера Order. Він має get-реалізацію та post-реалізацію.

```
public IActionResult CheckOut()
    {
      return View();
    }
```

Дана get-реалізація повертає представлення CheckOut.

```
public IActionResult CheckOut(Order order)
    {
      shopCart.shopCartItems = shopCart.ShopCartItemsList();
      if(shopCart.shopCartItems.Count == 0)
      {
        ModelState.AddModelError("", "У вас повинні бути товари");
      }
      if(ModelState.IsValid)
      {
        dataManager.allOrders.CreateOrder(order);
        return RedirectToAction("Complete");
      }
      return View(order);
    }
```

Post-реалізація приймає модель Order, перевіряє цю модель на валідність і

за допомогою DataManager додає її у базу даних.

```
public class Order
   {
      [Required]
      [BindNever]
      public Guid Id { get; set; }
      [Display(Name ="Ім'я")]
      [Required(ErrorMessage ="Поле Ім'я необхідне")]
      public string Name { get; set; }
      [Display(Name="Прізвище")]
      [Required(ErrorMessage = "Поле Прізвище необхідне")]
      public string Surname { get; set; }
      [Display(Name ="Email")]
      [DataType(DataType.EmailAddress)]
      [Required(ErrorMessage = "Поле Email необхідне")]
      public string Email { get; set; }
      [Required]
      [BindNever]
      [ScaffoldColumn(false)]
      public DateTime OrderTime { get; set; }
      public List<OrderDetails> AllOrderDetails { get; set; }
   }
```

### 3.3.5. Адміністратор

Щоб зайти у захищену область адміністратора, потрібно ввести /admin на головній сторінці сайту. Таким чином, викличеться метод Index контролера Home області Admin. Дана функціональність дає можливість додавати, редагувати та видаляти подкаст.

При натисканні на кнопку «Додати подкаст» викликається метод **Edit** контролера PodcastItem. Даний метод має get-реалізацію та post-реалізацію.

```
public IActionResult Edit(Guid id)
      {
         var entity = id == default ? new PodcastItem() :
```

```
dataManager.podcasts.GetPodcastItem(id);
            return View(entity);
        }
```

Get-варіація методу Edit отримує Id PodcastItem і за допомогою DataManager повертає даний PodcastItem з бази даних. Post-реалізація отримує на вхід PodcastItem і IFormFile. Даний метод створює новий PodcastItem за допомогою DataManager.

```
        public IActionResult Edit(PodcastItem model, IFormFile titleImageFile)
        {
            if (ModelState.IsValid)
            {
                if (titleImageFile != null)
                {
                    model.PathToPhoto = titleImageFile.FileName;
                    using (var stream = new
FileStream(Path.Combine(hostingEnvironment.WebRootPath, "img/", titleImageFile.FileName),
FileMode.Create))
                    {
                        Console.WriteLine(titleImageFile);
                        titleImageFile.CopyTo(stream);
                    }
                }
                dataManager.podcasts.SavePodcastItem(model);
                return RedirectToAction(nameof(HomeController.Index),
nameof(HomeController).Replace("Controller", ""));
            }
            return View(model);
        }
```

Завдяки get-реалізаціям та post-реалізаціям методу Edit відбувається додавання та редагування подкасті відповідно. Тобто, для редагування та створення нового подкасту використовується один і той самий метод представлення і контролер.

## 3.4. Тестування програми

Головна сторінка веб-застосунку — це сторінка із переліком подкастів. На ній відображені коротка інформація про подкаст: назва подкасту, категорія подкасту, автор подкасту та ціна подкасту. Також можливий пошук за назвою та фільтрація за категоріями.

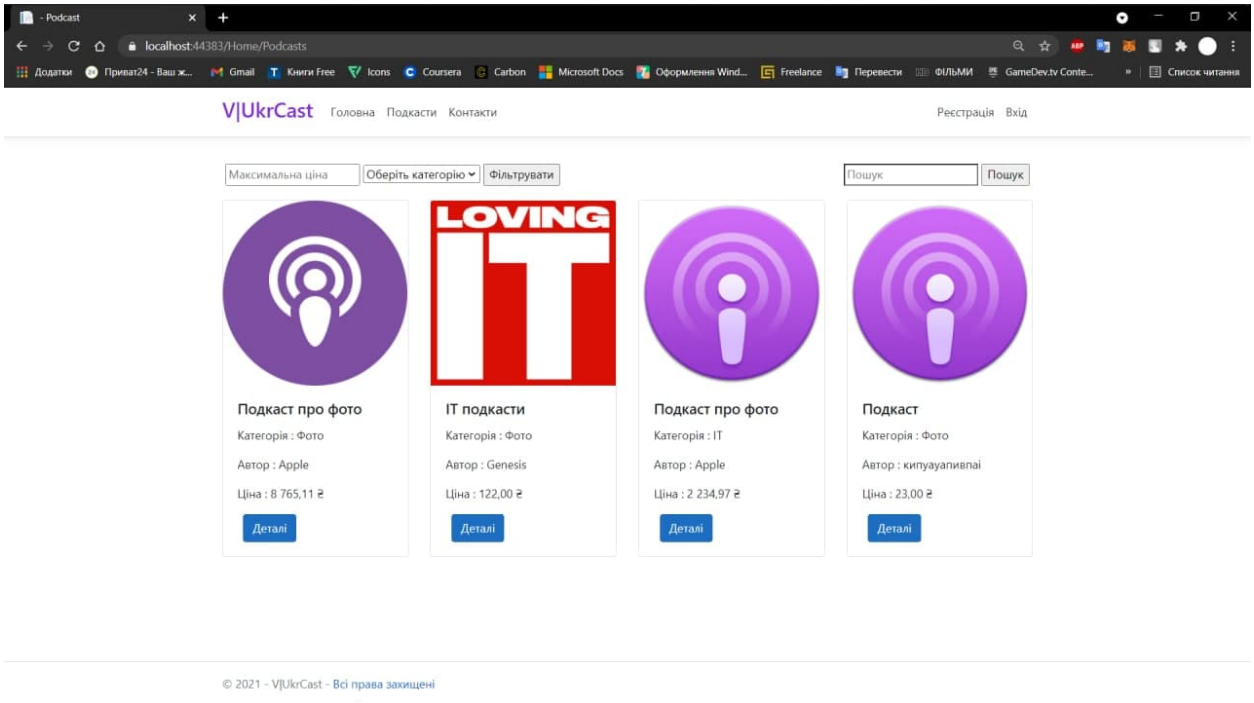


Рисунок 3.1 — Сторінка «Подкасти» для незареєстрованого користувача

Незареєстрований користувач може подивитися деталі про кожен подкаст, а саме дізнатися: попередні клієнти, аудиторія, кількість прослуховувань, тривалість, дата виходу та ціна.

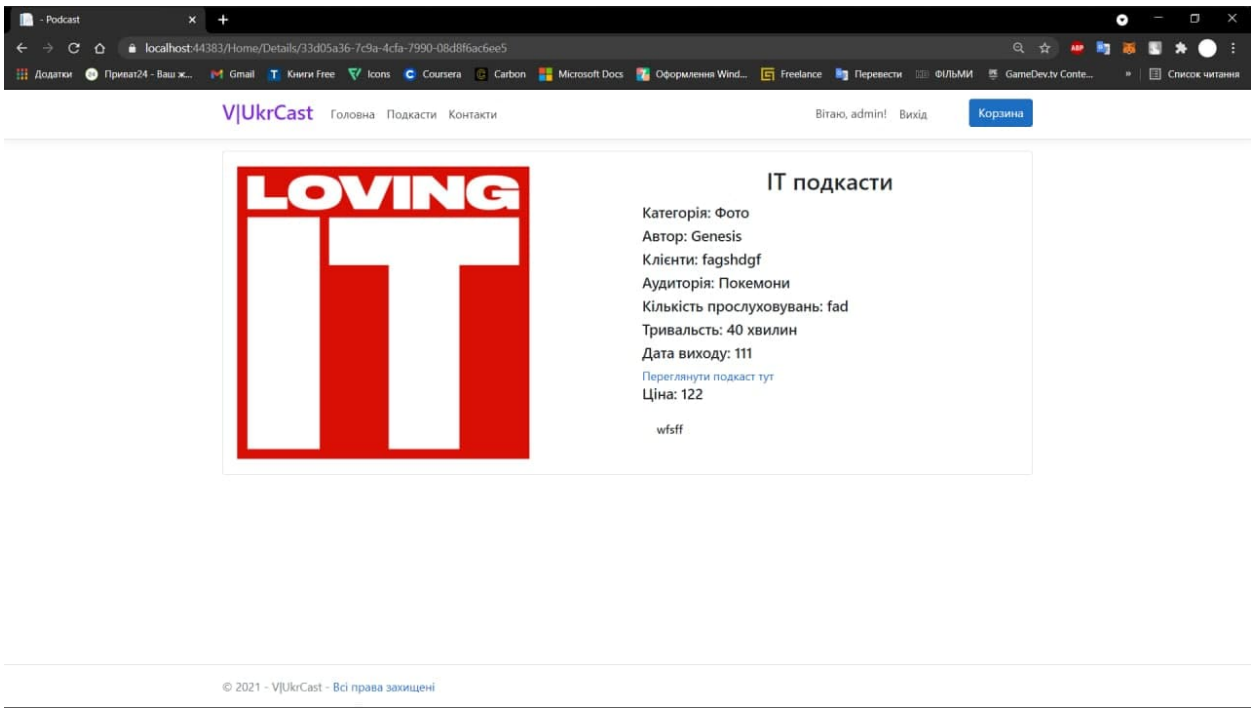


Рисунок 3.2 — Сторінка «Деталі подкасту»

Основна функція зареєстрованого користувача — можливість купівлі реклами у подкастах.

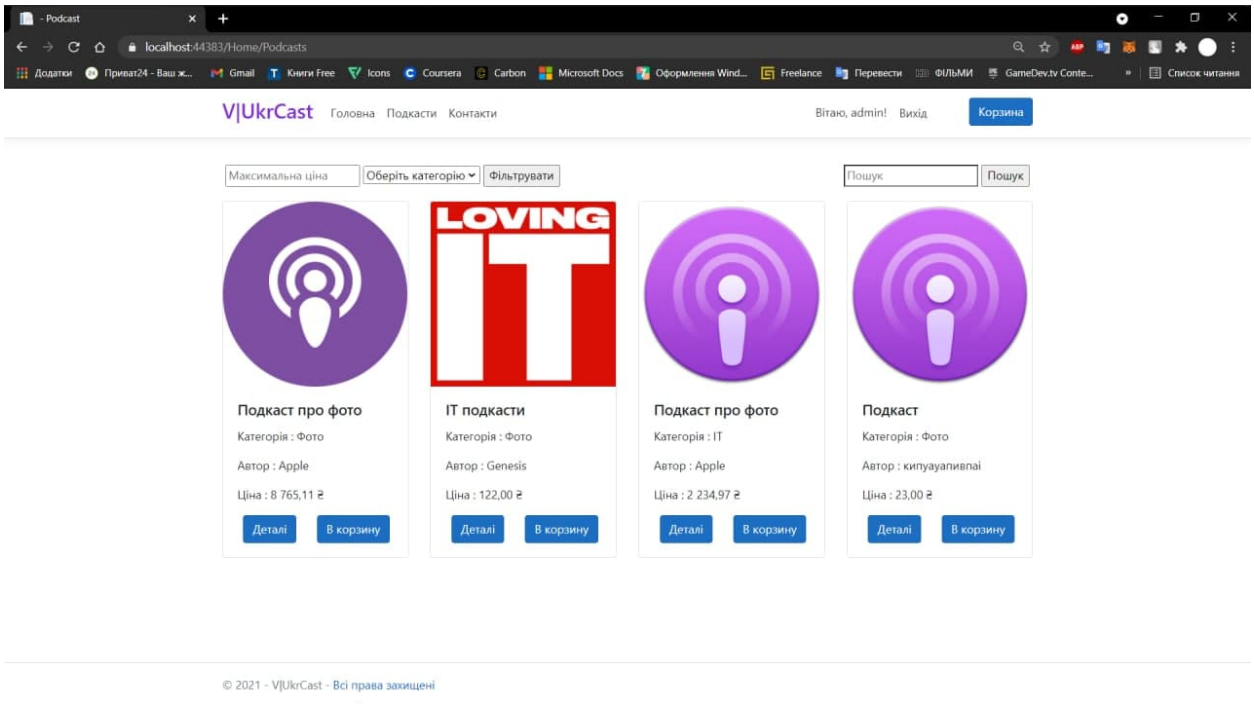


Рисунок 3.3 — Сторінка «Подкасти» для зареєстрованого користувача

Під час додавання у корзину, користувач може обрати тривалість рекламного повідомлення та зазначити саме рекламне повідомлення.

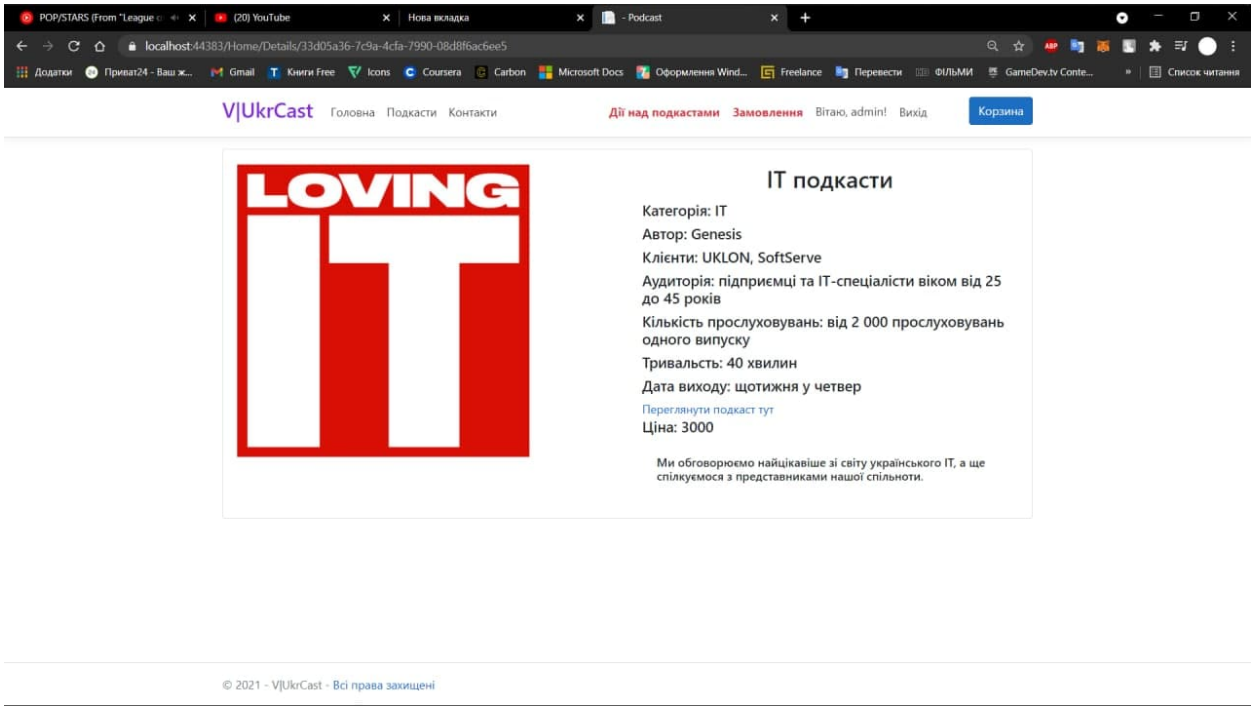


Рисунок 3.4 — Сторінка «Деталі замовлення» для зареєстрованого користувача

Адміністратор може перейти на сторінку (/admin), де він зможе керувати подкастами: додавати, редагувати та видаляти.

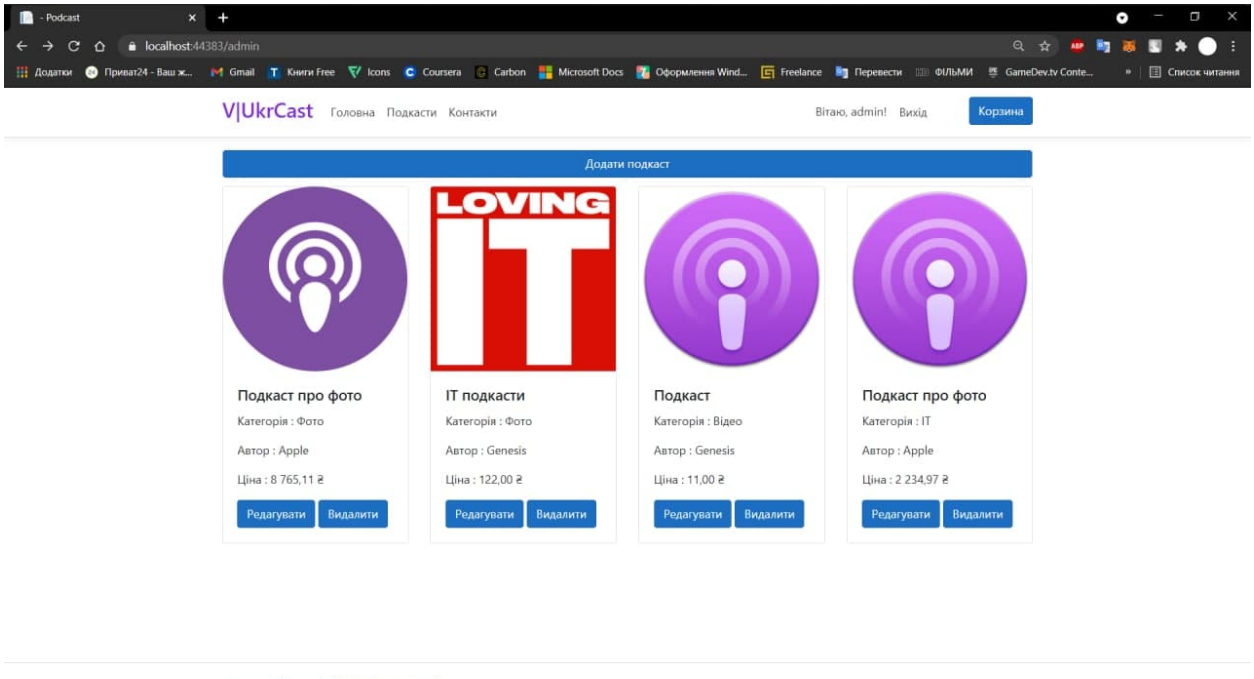


Рисунок 3.5 — Сторінка «Подкасти» для адміністратора

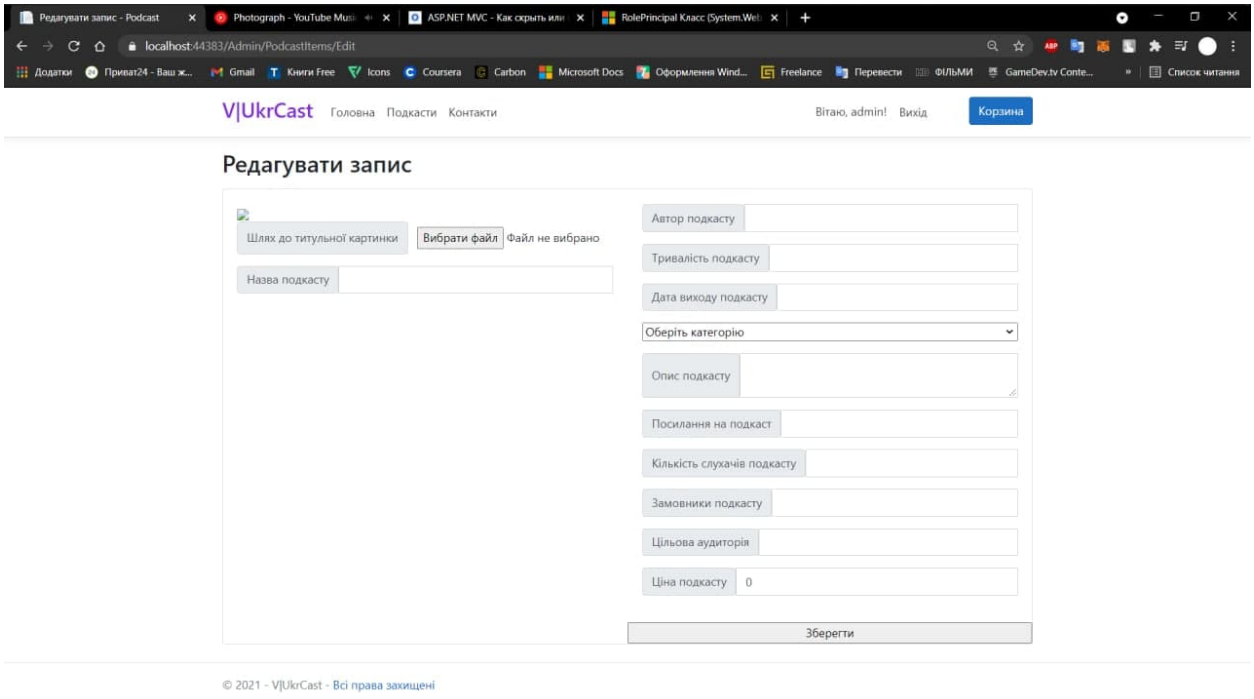


Рисунок 3.6 — Сторінка «Додати подкаст» для адміністратора

Адміністратор може переглядати історію замовлення та деталі замовлення.

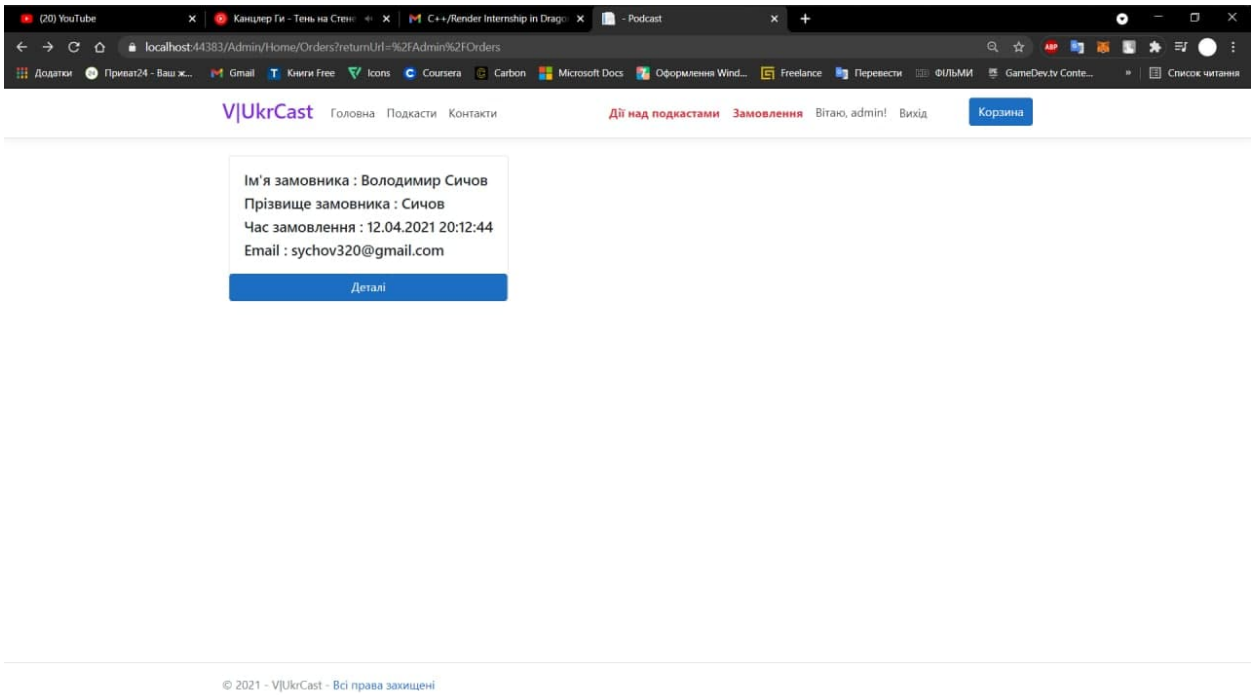


Рисунок 3.7 — Сторінка «Історія замовлення»

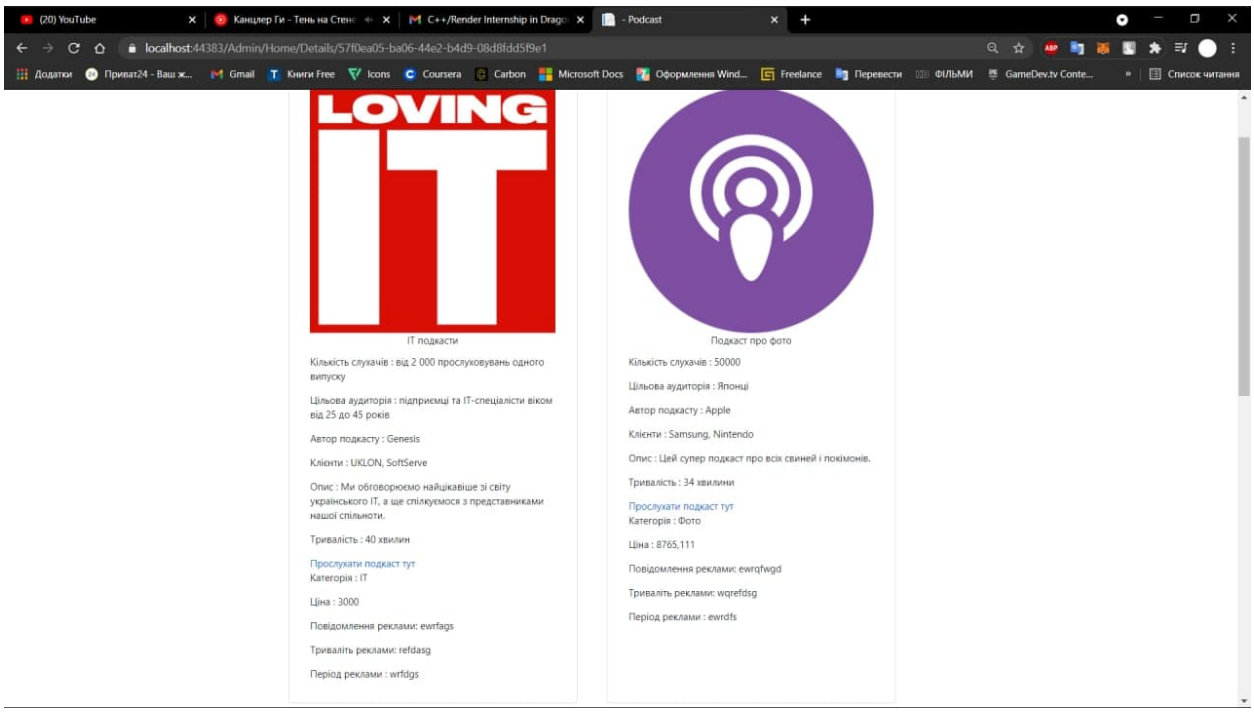


Рисунок 3.8 — Сторінка «Деталі замовлення»

## Висновок

Під час написання курсової роботи було реалізовано усі поставлені задачі щодо створення веб-застосунку з купівлі реклами у подкастах. Перед початком роботи були проведені опитування із представниками медіасфери, які підтвердили свою зацікавленість у розробці такої платформи, адже вона спрощить процес закупівлі реклами у подкастах для компаній та брендів. Крім цього, було враховано всі недоліки наявних способів та рішень купівлі реклами у подкастах, аби не допустити їх під час розробки цього веб-застосунку.

З технічного боку були використані сучасні технології розробки веб-застосунків, що надало можливість створити робочий застосунок, який у подальшому можна використовувати не лише у навчальних цілях, а також у комерційних.

Завдяки гнучкості патерну MVC, який використовувався під час розробки, у подальшому можна розширювати функціональність веб-застосунку для покращення його бізнес-моделі. Зокрема, у майбутньому планується додати відгуки на кожен з подкастів на сайт та створити рейтинг подкастів на основі оцінок користувачів за результатами рекламних кампаній.

Отже, у курсовій роботі було проаналізовано технічне завдання та мета створення веб-застосунку, проведено дослідження усіх наявних сервісів та способів з метою виявлення переваг та недоліків, а також обрано найбільш підходящі технології для створення веб-застосунку. Фінальним результатом курсової роботи стало створення веб-застосунку з купівлі реклами у подкастах.

**Список використаних джерел**

1. Дослідження про подкасти [Електронний ресурс] Автор: Edison Research https://www.edisonresearch.com/infinite-dial-2019/2. Дослідження про рекламу у подкастах [Електронний ресурс] Автор: Nielsen http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/2018-reports/marketeres-guide-to-podcasting-march-2018.pdf3. Перелік послуг, які надає піар-агентство [Електронний ресурс] https://www.newsfront.com.ua/

4. Платформа Midroll для купівлі реклами у подкастах [Електронний ресурс] https://www.midroll.com/

5. Платформа AdvertiseCast для купівлі реклами у подкастах [Електронний ресурс] https://www.advertisecast.com/

6. Класифікація сайтів за версією веб-розробників [Електронний ресурс] https://webmaestro.com.ua/ua/blog/vydy-saitiv/

7. Опис клієнт-серверної архітектури [Електронний ресурс] https://teachcomputerscience.com/client-server-architecture/

8. Документація .NET [Електронний ресурс] https://docs.microsoft.com/en-us/dotnet/fundamentals/

9. Документація ASP.NET Core [Електронний ресурс] https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0

10. MVC tutorial [Електронний ресурс] https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

11. Документація по MS SQL Server [Електронний ресурс] https://www.microsoft.com/ru-ru/sql-server/sql-server-2019

12. Pro Entity Framework Core 2 for ASP.NET Core MVC. Автор А. Фрімен. Рік видання – 2018.

13. ASP.NET Core 2.0 MVC & Razor Pages for Beginners: How to Build a Website. Автор Д. Фагерберг. Рік видання — 2017