

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ПІЗНАВАЛЬНОГО
ПІДКАСТИНГУ
EDUCATIONAL PODCAST SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-117
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Зозуля В.В.

(прізвище та ініціали)

Керівник Скрупський С.Ю.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Зозулі Вячеслава Віталійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки пізнавального підкастингу
Educational Podcast Software

керівник проєкту (роботи) Скрупський Степан Юрійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) травень 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “9” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис) Зозуля В.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Скрупський С.Ю.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 89 с., 32 рис., 1 табл., 3 дод., 16 джерел.

ПІДКАСТИ, ВЕБЗАСТОСУНОК, ВЕБФРЕЙМВОРК, MODEL-VIEW-CONTROLLER, PYTHON, DJANGO.

Об'єкт роботи – процес відтворення підкастів. Предмет роботи – програмне забезпечення підтримки пізнавального підкастингу.

Мета роботи – розроблення програмного забезпечення підтримки пізнавального підкастингу для забезпечення персоналізованого пізнавального потоку.

Матеріали, методи та технічні засоби: мова програмування Python, вебфреймворк Django.

Результати. Виконане порівняння мов програмування Python і Java, визначені функціональні вимоги, діаграми прецедентів неавторизованого користувача, автора підкастів та слухача, схема бази даних, структура вебзастосунку, розроблений вебзастосунок.

Висновки. Реалізоване програмне забезпечення дозволяє працювати з ним різним користувачам, у тому числі неавторизованим, за рахунок віддаленого доступу, дозволяючи авторам створювати власні підкасти, завантажувати епізоди, а слухачам відтворювати епізоди, створювати плейлист для відтворення епізодів, підписуватися на підкасти, прослуховувати нові епізоди підписаних підкастів, виконувати пошук підкастів, вподобати окремі епізоди, залишати власні коментарі, окрім того можна повертатися до епізодів, які користувач прослухав раніше.

Галузь використання – платформи для прослуховування підкастів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 89 pages, 32 figures, 1 table, 3 appendixes, 16 sources.

PODCASTS, WEB APPLICATION, WEBFRAMEWORK, MODEL-VIEW-CONTROLLER, PYTHON, DJANGO.

The object of the work is the process of playing podcasts. The subject of the work is software to support educational podcasting.

The purpose of the work is to develop software to support educational podcasting to provide a personalized educational flow.

Materials, methods and technical means: Python programming language, Django web framework.

Results. Comparison of Python and Java programming languages is performed, functional requirements are defined, use case diagrams of unauthorized user, podcast author and listener are created, database scheme, structure of web application, the developed web application were presented.

Conclusions. Implemented software allows various users to work with it, including unauthorized, through remote access, allowing authors to create their own podcasts, download episodes and listeners to play episodes, to create a playlist to play episodes, to subscribe to podcasts, to listen to new episodes, to search for podcasts, to like individual episodes, to leave you're their own comments and to return to episodes that the user has listened to before.

Field of usage is platforms for listening to podcasts.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Концепція підкастів	10
1.2 Огляд програмних засобів підтримки підкастингу	14
1.3 Висновки за розділом 1	18
2 Розробка архітектури програми.....	20
2.1 Вибір мови програмування	20
2.2 Вимоги до програмного забезпечення	21
2.3 Проектування бази даних	26
2.4 Визначення архітектури програмного забезпечення.....	30
2.5 Висновки за розділом 2	31
3 Розробка програми	32
3.1 Розроблені модулі	32
3.2 Опис прийнятих рішень під час реалізації	35
3.3 Висновки за розділом 3	42
4 Експлуатація та тестування програми	43
4.1 Призначення програми	43
4.2 Умови виконання програми	43
4.3 Робота з програмою	43
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	59
Додаток Б Текст програми	64
Додаток В Слайди презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

HTML	– HyperText Markup Language;
URL	– Uniform Resource Locator;
СУБД	– система управління базами даних.

ВСТУП

У порівнянні з 2018 роком, кількість людей у США, які прослухали щонайменше один підкаст у своєму житті, протягом року зросла на 20 мільйонів, а ще 14 мільйонів людей описують себе щотижневими слухачами, підкасти популярні серед усіх вікових груп, слухачі можуть підписатися на різні канали, завантажувати певні епізоди та слухати, де завгодно, в режимі офлайн, крім того, вони безкоштовні, хоча може використовуватися реклама або на розвиток проєкту автор може і просити якісь кошти, але це не обов'язково, до того ж на даний момент видніється тільки потенційний ріст галузі:

- це ідеальна платформа для необмеженого споживання вмісту на вимогу – до чого звикли більшість споживачів;

- індивідуальна курація вмісту для споживачів завжди відповідатиме тому, що люди слухають, що вони хочуть, коли вони хочуть;

- дохід від реклами від підкастів постійно зростає: все більше і більше брендів беруть участь у світі підкастингу, і це відображатиме рентабельність інвестицій, надаючи брендам більше впевненості в інвестуванні в платформу;

- оскільки подкасти доступні на таких платформах, як Spotify, вони так само легко доступні, як і прослуховування музики: в автомобілях, які їдуть на роботу, на кухні, через навушники під час тренувань у тренажерному залі;

- зростання популярності пристроїв розпізнавання голосу, таких як Alexa, Google та Siri, полегшить доступ до підкастів;

- високовизнані, провідні на ринку бренди, що визначають тенденції, такі як Google, Microsoft, eBay та McDonalds, збільшили використання формату реклами, а також створили власні канали: це додало більшості брендів впевненості та надало можливості розширення їх маркетингових горизонти [1].

Пізнавальний підкастинг, зважаючи на популярність загалом підкастингів у молоді, є потенційним інструментом з соціальним впливом, який окрім отримання додаткового доходу має потенційний довготривалий дохід – соціальний ефект.

Тому метою роботи було визначено розроблення програмного забезпечення підтримки пізнавального підкастингу для забезпечення персоналізованого пізнавального потоку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Концепція підкастів

Підкаст (скорочення від iPod та Broadcast) – це цифровий аудіофайл, доступний в мережі Інтернет для завантаження на комп'ютер або мобільний пристрій, зазвичай доступний у вигляді епізоду, нові появи яких можуть отримувати абоненти автоматично [2].

Підкастинг розпочався як здебільшого незалежний спосіб для людей створити своє повідомлення та створити спільноту людей зі схожими інтересами, на сьогодні є подкасти від:

- фізичних осіб;
- компаній (великих та малих);
- радіомереж, засоби яких частково і перейняв на себе підкастинг;
- телевізійних мереж;
- нових мереж лише для підкастів (Gimlet – чудовий приклад);
- коміків;
- казкарів;
- церков [2].

Сучасна популярність підкастів обумовлена наступним:

– підкасти дозволяють реалізовувати багатозадачність: основна причина, чому підкасти настільки популярні, полягає в тому, що вони дозволяють аудиторії економити час, в Edison Research було досліджено місця прослуховування підкастів і з'ясовано, що найчастіше люди слухають підкасти вдома, в машині, під час прогулянки або тренування, це означає, що ті, хто слухає підкасти, виконують принаймні два завдання одночасно, тренуючись і слухаючи, наприклад, загалом було також визначено, що під час прослуховування підкастів найпоширеніша діяльність – це робота по дому, їзда, готування їжі;

– прослуховування підкастів – це простий спосіб дізнатись про останні новини та події: щоденні новинні підкасти – це переповнений ринок,

на якому вже зайняті такі медіа-гіганти, як The Washington Post, Vox, The Guardian, The Economist та The New York Times (щодня має понад 2 мільйони слухачів);

– розважальні підкасти: окрім вивчення нових речей та з'ясування новин, люди також вибирають підкасти, коли хочуть розважитись: музика, їжа, спорт, трилери та хобі – одні з найпопулярніших категорій подкастів, крім того, подкасти також розважають через різноманітність форматів, які вони мають, є інтерв'ю, бесіди, пізні вечірні шоу, фантастичні та нехудожні розповіді, а також відео та прямі підкасти;

– сила вибору: пропонуючи набагато інтимніший досвід, ніж радіо, підкасти дозволяють вибрати розповідь, яка відповідає смакам користувача, вибір епізоду на основі настрою, симпатій, антипатій та того, що користувач хоче отримати, означає, що можна адаптувати весь досвід відповідно до своїх бажань та потреб;

– позитивні ефекти підкастів: пошук людей, які слухають та люблять ту саму музику, мають ті самі інтереси, що і ви, створює величезне відчуття спільноти, що особливо чітко видно, коли шанувальники збираються на живих концертах, багато дружніх стосунків на все життя складаються навколо спільної любові до певного виконавця, групи чи музичного жанру, пошук прихильників підкастів-однодумців може створити подібний зв'язок, тоді як публікація про свої улюблені підкасти в соціальних мережах – це чудовий спосіб поділитися розвагами, які вам подобаються [3]-[4].

Щодо підкастів можна виділити наступні характеристики:

– не існує заздалегідь визначеної довжини, формату, стилю, рівня виробництва чи чогось іншого;

– підкасти можуть бути розбиті не лише на епізоди, але і на сезони, як телевізійне шоу, або бути епізодичними та постійними, при цьому щотижневий випуск нових епізодів є загальним явищем, але є щоденні подкасти, щотижневі підкасти;

– підкаст – це серія епізодів, ці епізоди є аудіофайлами, які, ймовірно, зберігаються у компанії, що розміщує підкасти, до того ж зазвичай підкастами можна підписатися, що полегшує отримання сповіщень, коли з’являються нові серії;

– створити власне шоу відносно легко і дешево, при цьому популярність подкастів зростає з кожним днем, все це привертає увагу авторів до цього інструменту: не потрібно покладатися на радіостанції для запису та мовлення, можна говорити про все, що цікавить самого автора – саме він визначає формат, якщо такий формат взагалі існує [2].

У даній роботі акцент буде зроблено на пізнавальні підкасти, які з одного боку дозволяють людині пізнавати нове, а з іншого боку за рахунок різноманіття форматів, як було зазначено вище, мають розважальний формат, тобто подають пізнавальне у тому вигляді, який може зацікавити сучасну аудиторію. Пізнавальні підкасти окрім всього виконують також соціальну функцію, адже дозволяють розширювати рівень знань, обізнаності в звичайному житті звичайних людей. Усе це вказує на важливість саме даного жанру.

Багато прогнозів вказують на те, що галузь підкастів буде тільки розвиватися в майбутньому адже з привертанням першої значимої уваги далі великі гравці будуть займатися цією галуззю, тобто вкладати додаткові кошти в галузь. Відповідно все це вимагає підтримки відповідного програмного забезпечення, що вказує на актуальність розробки. Дана робота дозволяє скерувати розробку на досягнення соціального ефекту в майбутньому. Тобто в результаті створення цієї платформи може заснувати певну спільноту, учасники якої будуть об’єднані спільними інтересами. Створення інших програмних засобів для цієї спільноти може призвести до отримання додаткових засобів на розвиток нових проєктів або отримання ресурсів для підтримки роботи кампанії загалом.

Під час створення підкастів і відповідних засобів роботи з ними потрібно враховувати наступні принципи організації даного жанру:

– управління фокусом і увагою слухача: підкасти налаштовують на те, що слухач довіряє контроль над власним досвідом протягом усього періоду відтворення епізоду автору, доки автор продовжує надавати задовольняючий досвід, слухачі з задоволенням поступаються контролем та довірою на постійній основі, тому найкращий спосіб для професійного надавача послуг підкастингу – продемонструвати свою прихильність цій угоді про довіру (забезпечувати лише чудовий вміст для своїх слухачів), відповідно в епізодах не має бути нічого зайвого, нічого, що відволікає чи буквально не додає вартості, створений продукт потрібно тримати щільним та актуальним, цей принцип має представлення і у вебзастосунку теж, при цьому потрібно переконатися, що використовуються чіткі та очевидні заголовки та резюме, коли епізод публікується, переконатися, що цей підкаст помітний у застосунку і що він привертає увагу, окрім того хороша розшифровка може допомогти;

– направлення слухачів до кінцевої мети: важко направити власних слухачів, якщо підкаст не має загальну мету або окремі епізоди підкастів не мають такої цілі, мета повинна бути чітко сформульована в одному реченні, при цьому якою мета б не була мета, її потрібно знати, незважаючи на те, що загалом може бути декілька цілей у підкасту, які автор хоче досягти за допомогою свого підкасту, при цьому будь-який з епізодів повинен мати одну мету, як тільки мета для окремого епізоду встановлена, потрібно зробити її надзвичайно зрозумілою та очевидною для слухача;

– послідовність є дуже важливою для підкастів: слухачі підписуються на підкасти і очікують, що вміст надходитиме регулярно, якщо нового епізоду немає в стрічці підкастів, вони знайдуть щось інше, щоб замінити його, тоді буде важче повернути їх, коли з'явиться наступний епізод, але при цьому потрібно враховувати і те, що не слід поспішати з епізодами, які мають низьку якість, лише щоб дотримуватися послідовного графіка, адже тоді також можна втратити слухачів, потрібно спочатку спланувати графік запису та публікації нових епізодів, залежно від тематики, можливо логічним

буде працювати наперед, а епізоди випускати пізніше, або можливо застосувати серіалізований підхід і випустити групу епізодів за короткий проміжок часу, а потім зробити перерву і розпочати знову наступного сезону, у будь-якому випадку потрібно вибрати графік і дотримуватися його – і дати слухачам знати, коли очікувати наступного епізоду;

– особистісний підхід також має значення: ключовий фактор, який визначає підкаст, – це особистість людини чи людей за мікрофоном, багато людей слухають підкасти, роблячи інші справи – водіння, вигулювання собаки, прибирання будинку тощо – і вміст повинен бути достатньо залученим, щоб тримати їх зосередженими, щоб зберегти інформацію, незважаючи ні на що інше, що вони можуть робити, наявність гостей у шоу може допомогти в цьому, це допомагає зокрема і забезпечувати формат, якого можна дотримуватися щотижня, при цьому чим більше людей додано в підкаст, тим складніше стає послідовний запис, адже потрібно знайти моменти, коли всі доступні, потрібно отримати більше обладнання, щоб зробити запис, і редагування стає більш складним, коли задіяні декілька голосів [5]-[6].

1.2 Огляд програмних засобів підтримки підкастингу

У даному підрозділі потрібно виконати аналіз принципів організації основних сучасних платформ, які використовуються для підтримки підкастів.

Podbean [7] (рис. 1.1) – один з основних постачальників послуг хостингу підкастів на ринку, що пропонує необмежену кількість хостингових підкастів з необмеженою пропускнуою здатністю, при цьому надаються різноманітні засоби відображення з налаштованими темами, можна вбудувати програвач підкастів Podbean безпосередньо у свої публікації та сторінки на WordPress, також надаються інструменти просування підкастів, які автоматично надсилають епізоди підкасту до всіх каталогів найкращих підкастів, окрім того Podbean також пропонує рекламний ринок, який дає

можливість підкастерам монетизувати свій вміст, вони мають величезну спільноту підкастерів, які вже використовують Podbean як платформу для підкастингу, але при цьому потрібно враховувати, що безкоштовний тариф має обмежені обсяги пам'яті, а платні тарифи починаються з 9 доларів на місяць з необмеженим сховищем [8].

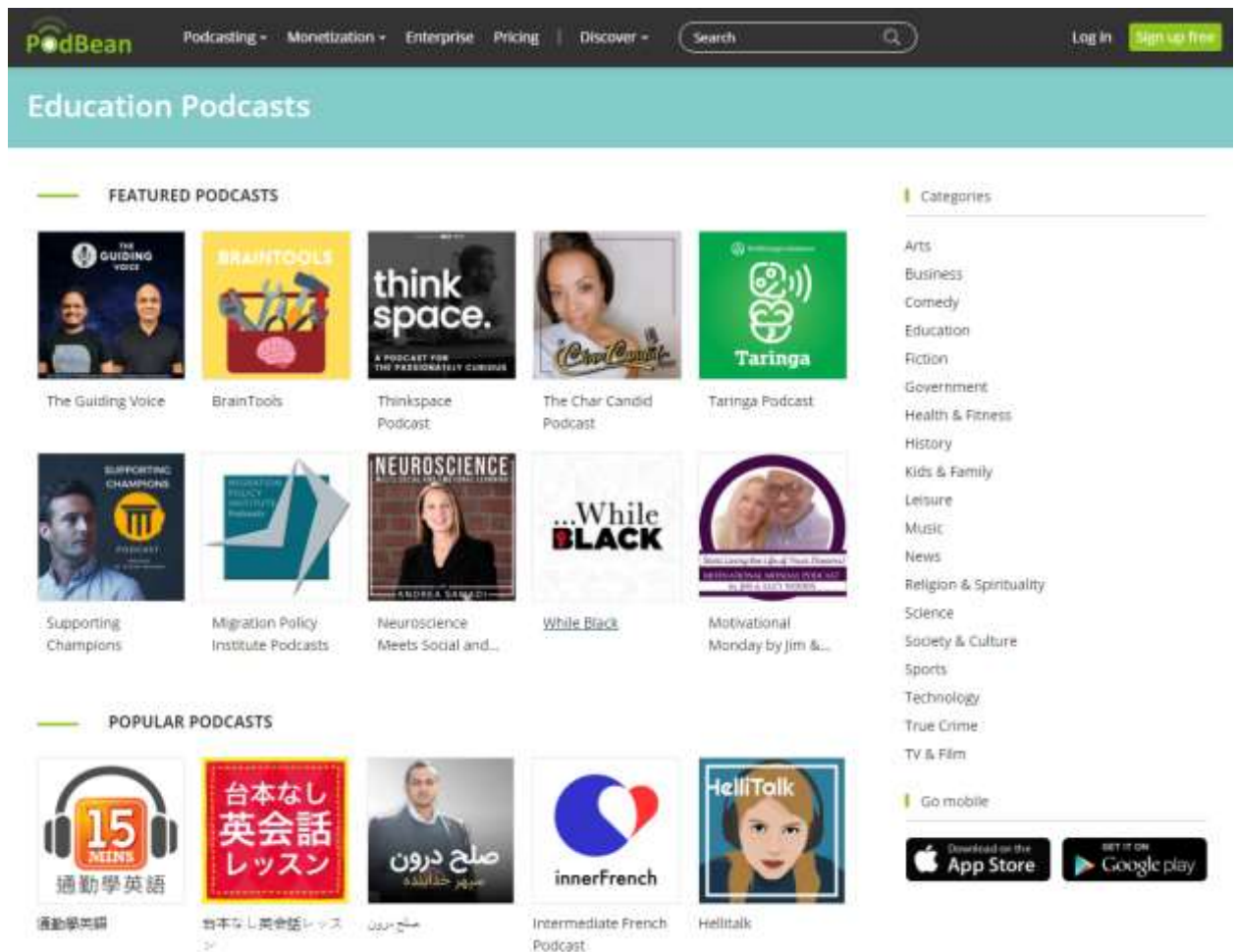


Рисунок 1.1 – Застосунок Podbean

Підкасти поділені на різноманітні категорії, що включають освіту, музику, мистецтво, історію, новини тощо.

Кожен підкаст як і окремі епізоди підкасту можуть мати певний опис, який визначається самим автором, певне оформлення сторінки, окрім того надається статистика щодо завантаження підкасту, кількості епізодів. Можна переглядати перелік всіх епізодів і відтворювати їх з цієї загальної сторінки, можна переходити на окрему сторінку, загалом набір засобів однаковий,

окрім того, що на окремій сторінці перелічуються коментарі користувачів. Статистика завантаження, можливість поділитися епізодом, а також вподобати епізод надається і з загальної сторінки.

Окрім того надається перелік користувачів, які підписалися на підкаст.

На основі подібних механізмів, включаючи засоби підписування на підкасти, є і в іншій платформі – Castbox [9] (рис. 1.2).

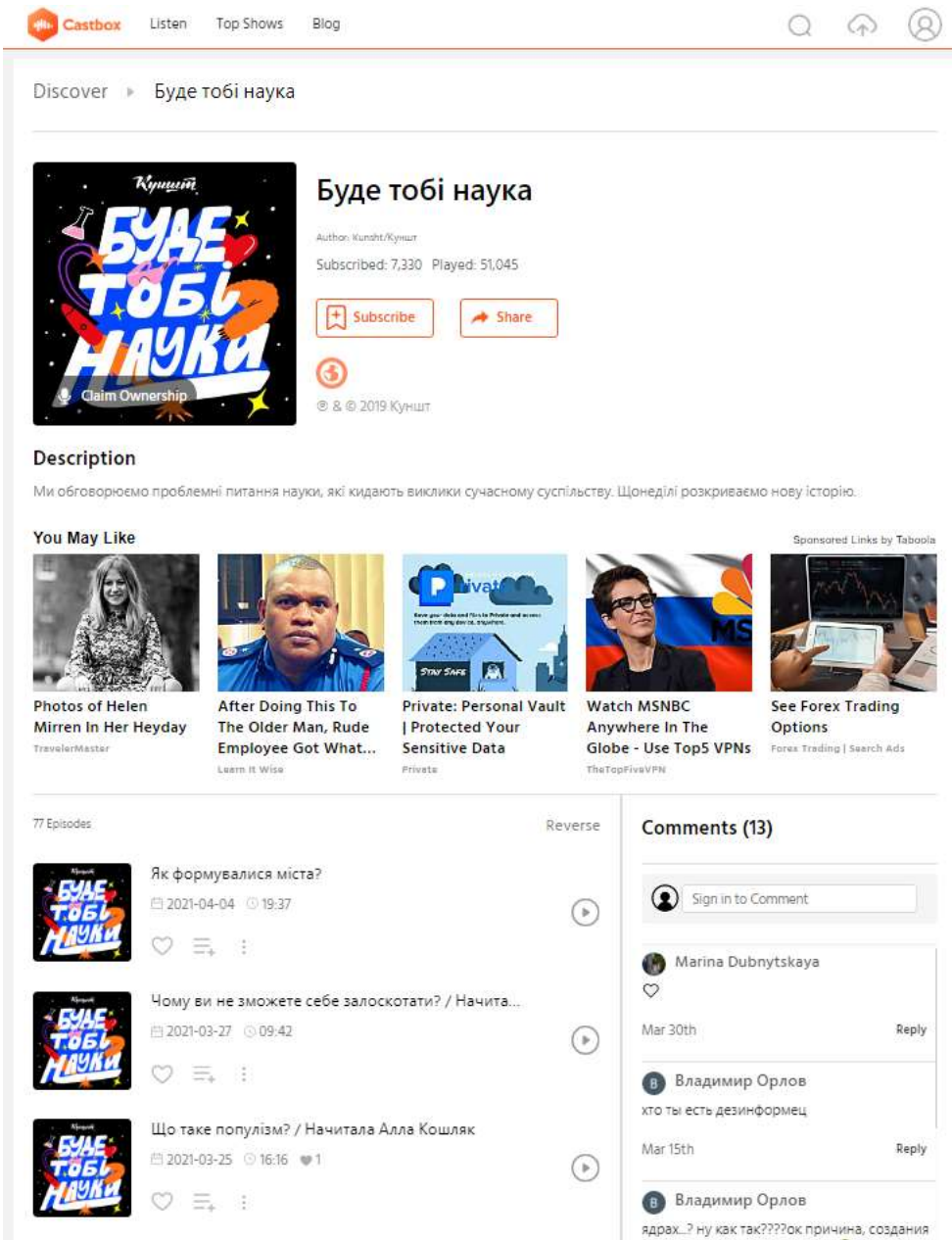


Рисунок 1.2 – Застосунок Castbox

Окрім того даний застосунок має також засіб створення власних плейлистів для відтворення без проміжків. Для цього потрібно додати відповідний епізод до плейлиста.

Mixcloud [10] (рис. 1.3) відомий швидше як засіб хостингу власних підкастів, які потім розташовуються на власних сайтах підкастів. Пошук надається за жанром або тегом. Тут також є коментарі, набір тегів, що може бути достатньо розширеним.

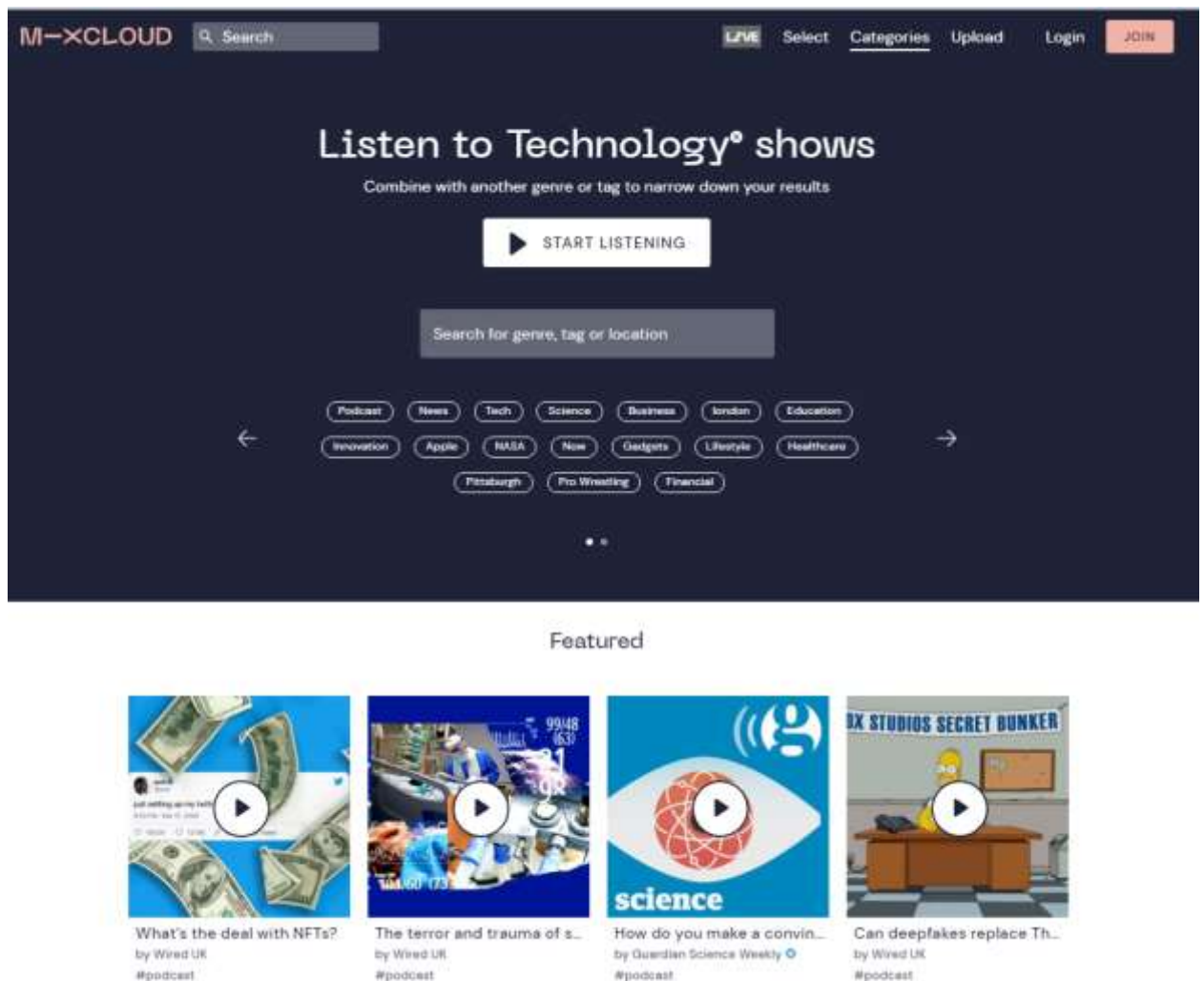


Рисунок 1.3 – Застосунок Mixcloud

Окрім того підкасти можуть мати власні сайти, де об'єднуються автори різних підкастів для роботи під одним брендом. Одним з таких сайтів є Радіо Поділ [11] (рис. 1.4), яке надає доступ до достатньо великої кількості підкастів. На цьому сайті також надаються інструменти для фінансування

проектів. З сайту надається також можливість перейти до місць розташування підкастів на сайтах-агрегаторах. При цьому щодо кожного підкасту надається тільки набір епізодів та інформація про автора.

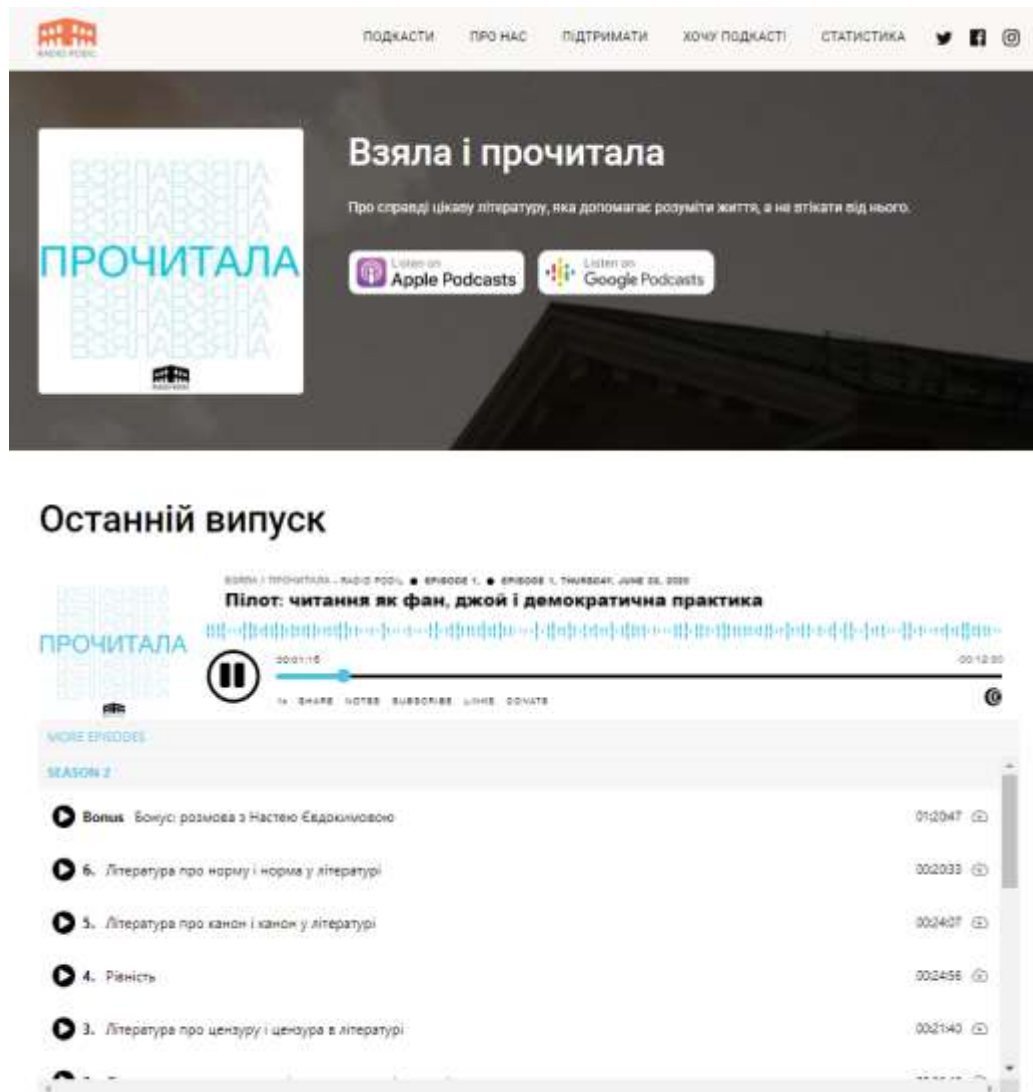


Рисунок 1.4 – Підкаст Радіо Поділ

1.3 Висновки за розділом 1

Проаналізовано концепцію підкастингу, виділено основні особливості підкастів, причини сучасної популярності підкастів та принципи їх організації. Визначено пізнавальні підкасти в якості основного призначення розроблюваного програмного забезпечення.

Розглянуто основні програмні засоби хостингу підкастів, що включають Podbean, Castbox, Mixcloud, та проаналізовано їх функціональні можливості та принципи організації.

Завдання дипломної кваліфікаційної роботи бакалавра:

- виділення вимог до програмного забезпечення підтримки пізнавального підкастингу;
- вибір мови і засобів проєктування і розробки;
- проєктування програмного забезпечення;
- реалізація і налагодження програмного забезпечення.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір мови програмування

На першому етапі потрібно визначити, якою саме мовою буде вестися розробка, адже від цього залежать і проєктні рішення: зокрема вибір мови пов'язаний з визначенням архітектури в подальшому, а також вибором системи управління базами даних (СУБД), яка в свою чергу впливає на вибір типів полів під час роботи з проєктування бази даних зокрема.

За багатьма рейтингами найпопулярнішою в даний момент мовою програмування є мова Java. Оскільки для підтримки доступу з різноманітних пристроїв, у тому числі персональних комп'ютерів, мобільних пристроїв, до підкастів потрібно створити вебзастосунок, а Java дозволяє їх створювати на основі фреймворків, то її було визначено в якості першого можливого вибору мови програмування, якою може бути створено проєкт.

Другим варіантом було визначено мову Python [12]-[13], оскільки вона також дозволяє створювати вебзастосунки, а також відома своєю легкістю у використанні.

За результатами порівняння (табл. 2.1) було виявлено, що мова програмування Python дещо поступається мові Java щодо швидкості виконання застосунків, але при цьому переважає її за критеріями, які пов'язані зі швидкістю запуску проєкту, швидкістю самої розробки та подальшої підтримки. У даному випадку проєкт розробляється одним учасником, а відповідно критерії, які були вище визначені в якості переваг мови Python, є критичними.

Загалом мова Python дозволяє створювати проєкти швидше, створювати універсальніший код. Тому було вирішено використовувати саме мову Python для розробки вебзастосунку підтримки пізнавального підкастингу.

Таблиця 2.1 – Обґрунтування вибору мови програмування

Критерій порівняння	Python	Java
Лаконічність коду	Вища	Нижча
Швидкість розробки	Вища	Нижча
Складність підтримки	Нижча	Вища
Швидкість запуску	Вища	Нижча
Швидкість виконання	Нижча	Вища
Типізація	Динамічна	Статична
Запуск під різними операційними системами	Так	Так
Веброзробка	Так	Так
Доступність фреймворків розробки	Так	Так

2.2 Вимоги до програмного забезпечення

Грунтуючись на основі виконаного в розділі 1 огляду існуючих програмних засобів для вирішення цієї проблеми, було визначено набір функціональних вимог до програми:

- створення підкастів;
- перегляд власних підкастів;
- перегляд переліку епізодів власного підкасту;
- редагування підкастів;
- архівування підкастів;
- внесення епізодів підкастів з можливістю розділення на сезони;
- створення розшифровувань епізодів;

- редагування епізодів підкасту;
- підписування на підкаст;
- відписування від підкасту;
- перегляд підписаних на підкаст користувачів;
- вподобання епізодів підкастів;
- коментування епізодів підкастів;
- пошук підкастів;
- визначення тегів епізоду;
- визначення жанру підкасту;
- перегляд підкастів за жанрами;
- отримання переліку найпопулярніших підкастів за жанром;
- додавання епізоду до власного плейлиста;
- вилучення епізоду з власного плейлиста;
- перегляд переліку епізодів підкасту;
- прослуховування епізоду підкасту;
- авторизація та можливість неавторизованого доступу;
- визначення епізодів, доступних тільки для підписаних користувачів;
- перегляд історії прослуховувань епізодів користувачем;
- перегляд автором підкасту кількості прослуховувань підкасту загалом та кожного епізоду зокрема;
- перегляд переліку підкастів, на які підписався користувач;
- отримання переліку нових епізодів підписаних підкастів;
- реєстрація акаунтів.

Таким чином, в результаті було створено розгалуджену систему функціональних можливостей програмного забезпечення, що надає велику кількість засобів для слухача щодо доступу до тих підкастів, які його зацікавили або можуть зацікавити, а автору щодо роботи з власними підкастами.

Окрім того кожен епізод отримує не тільки звукове представлення та опис, але також автор може скористатися розшифровуванням епізоду, тобто

сформувати текстове представлення. Це достатньо важливо для пізнавальних підкастів, адже текстові описи потім зручно накопичувати або використовувати в соціальних мережах, що додатково привертає увагу до підкастів у результаті, тобто виконує функцію популяризації як знань, так і самих підкастів.

За результатами моделювання сценаріїв використання програми було створено діаграми прецедентів неавторизованого користувача (рис. 2.1), автора підкасту (рис. 2.2) та слухача підкасту.

Фактично автор і слухач мають виражатися одним і тим же акаунтом, але в залежності від того, виконується доступ до власних підкастів чи ні, розмежовується фактична поточна роль користувача в системі: автор або слухач.

Неавторизований користувач не має повних прав доступу звичайного авторизованого користувача, але може ознайомитися з підкастами.



Рисунок 2.1 – Діаграма прецедентів неавторизованого користувача

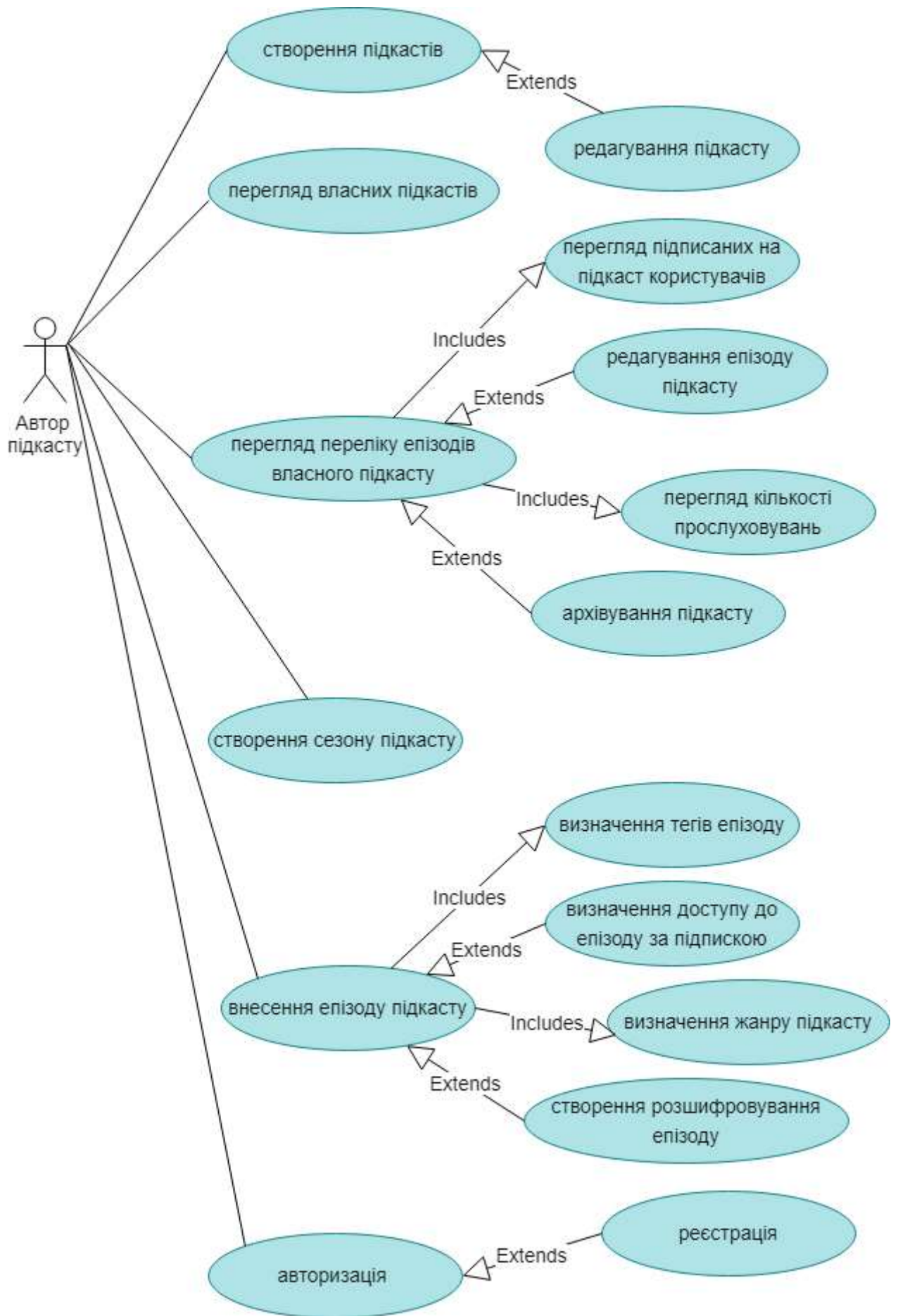


Рисунок 2.2 – Діаграма прецедентів автора підкасту

Слухач (рис. 2.3) має розширені права прослуховування підкастів.

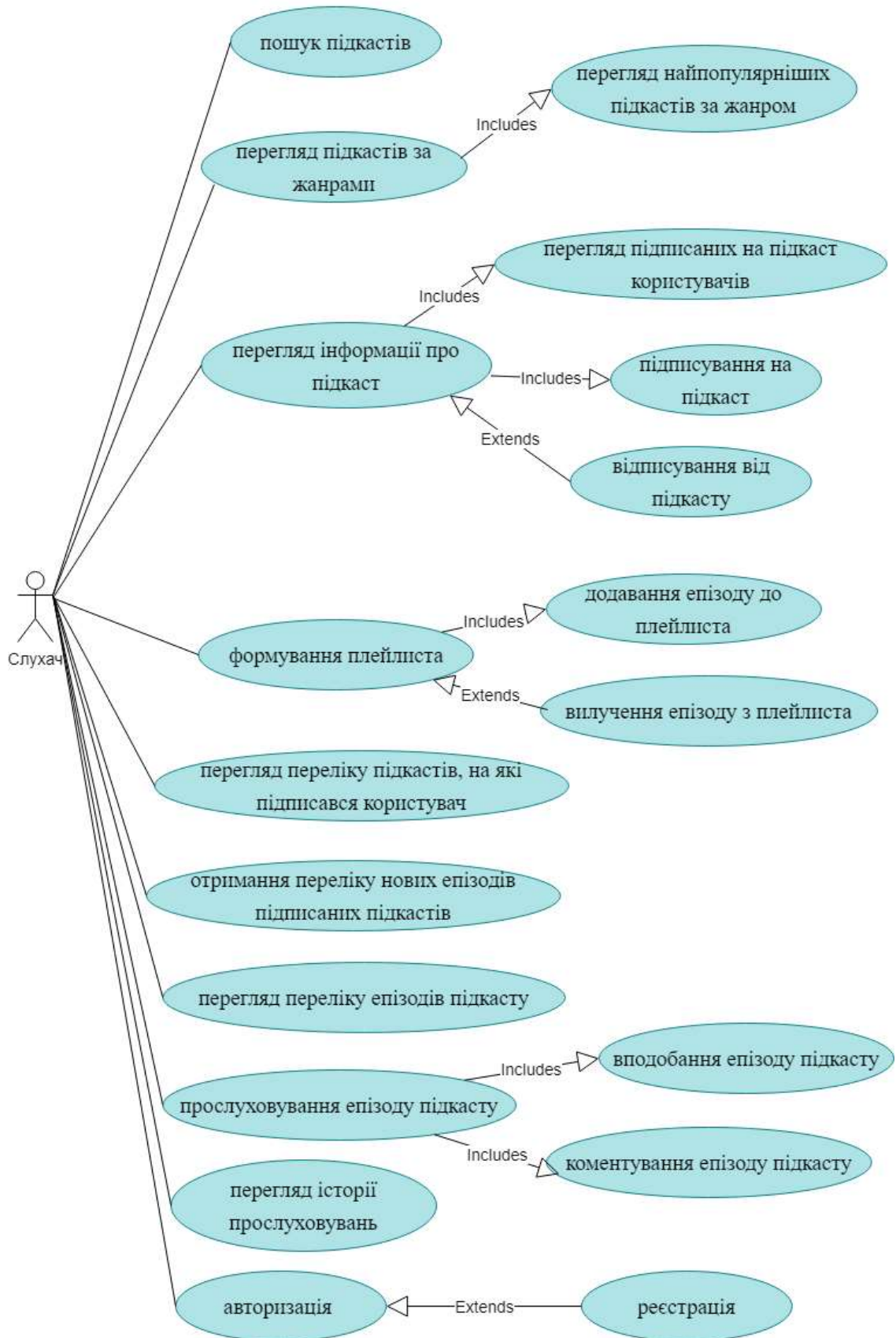


Рисунок 2.3 – Діаграма прецедентів слухача

2.3 Проєктування бази даних

Для забезпечення доступу слухачів до підкастів, а авторів підкастів до наповнення вебзастосунку даними потрібно спочатку визначити схему роботи з даними. Оскільки до даних належать підкасти, акаунти користувачів, дані про прослуховування, підписки тощо, тобто дані є достатньо розгалуженими, то для їх збереження та забезпечення структурованого доступу до них необхідно спроєктувати базу даних.

Проєктування бази даних виконувалось на основі використання СУБД MySQL, адже вона надає повноцінні засоби роботи з реляційними базами даних, якісно підтримує корпоративні програмні продукти різного масштабу та до того ж є безкоштовною.

Розглянемо основні таблиці, які були створені в результаті проєктування реляційної бази даних.

Таблиця Podcast представляє дані відповідного підкасту і містить такі поля:

- id – ідентифікаційний номер підкасту;
- name – назва;
- photo – фотографія;
- description – опис;
- userid – автор підкасту;
- followercount – кількість підписників;
- date1 – дата публікації;
- mode – стан (архівований чи ні).

Таблиця Episode представляє дані конкретного епізоду підкасту і містить такі поля:

- id – ідентифікаційний номер епізоду;
- name – назва епізоду;
- description – опис;
- photo – фотографія;

– `textpresent` – текстове розшифрування епізоду (може бути пропущене, а якщо наявне, то представляється іменем файлу, в якому розташовано текст, оскільки може бути надто довгим);

– `seasonid` – сезон підкасту, до якого належить епізод (за замовчуванням одразу для кожного підкасту створюється перший сезон);

– `audio` – аудіофайл епізоду підкасту;

– `likecount` – кількість уподобань;

– `listencount` – кількість прослуховувань;

– `mode` – стан (доступний без підписки для відтворення чи ні);

– `date1` – дата і час публікації.

Таблиця `User` представляє дані акаунтів користувачів і містить такі поля:

– `id` – ідентифікаційний номер акаунту;

– `login` – логін акаунту;

– `password` – пароль акаунту;

– `lastlisten` – дата і час останнього прослуховування підкастів користувачем.

Таблиця `Likes` представляє дані вподобань користувачами епізодів підкастів і містить такі поля:

– `id` – ідентифікаційний номер уподобання;

– `date1` – дата і час вподобання;

– `userid` – акаунт користувача;

– `episodeid` – епізод.

Таблиця `Comments` представляє дані коментарів користувачів щодо епізодів підкастів і містить такі поля:

– `id` – ідентифікаційний номер коментаря;

– `date1` – дата і час коментаря;

– `userid` – користувач, який написав коментар;

– `episodeid` – епізод;

– `commtext` – текст коментаря.

Таблиця Follow представляє дані підписок користувачів на підкасти і містить такі поля:

- id – ідентифікаційний номер підписки;
- date1 – дата і час підписки на підкаст;
- userid – користувач, який підписався;
- podcastid – підкаст, на який оформлено підписку.

Таблиця Playlist представляє дані плейлиста користувача епізодів підкастів і містить такі поля:

- id – ідентифікаційний номер плейлиста;
- userid – акаунт користувача;
- episodetid – епізод.

Таблиця Listen представляє дані прослуховувань епізодів підкастів і містить такі поля:

- id – ідентифікаційний номер прослуховування;
- date1 – дата і час прослуховування;
- userid – акаунт користувача;
- episodeid – епізод, який було прослухано.

Таблиця Tag представляє дані тегів епізодів підкастів і містить такі поля:

- id – ідентифікаційний номер тегу;
- name – тег.

Таблиця Genre представляє дані жанрів підкастів і містить такі поля:

- id – ідентифікаційний номер жанру;
- name – жанр.

Таблиця EpisodeTag представляє дані тегів епізодів підкастів, реалізує зв'язок багато-до-багатьох між таблицями Episode і Tag, в структурі містить такі поля:

- id – ідентифікаційний номер запису;
- tagid – тег;
- episodeid – епізод.

Схема бази даних приведена на рис. 2.4.

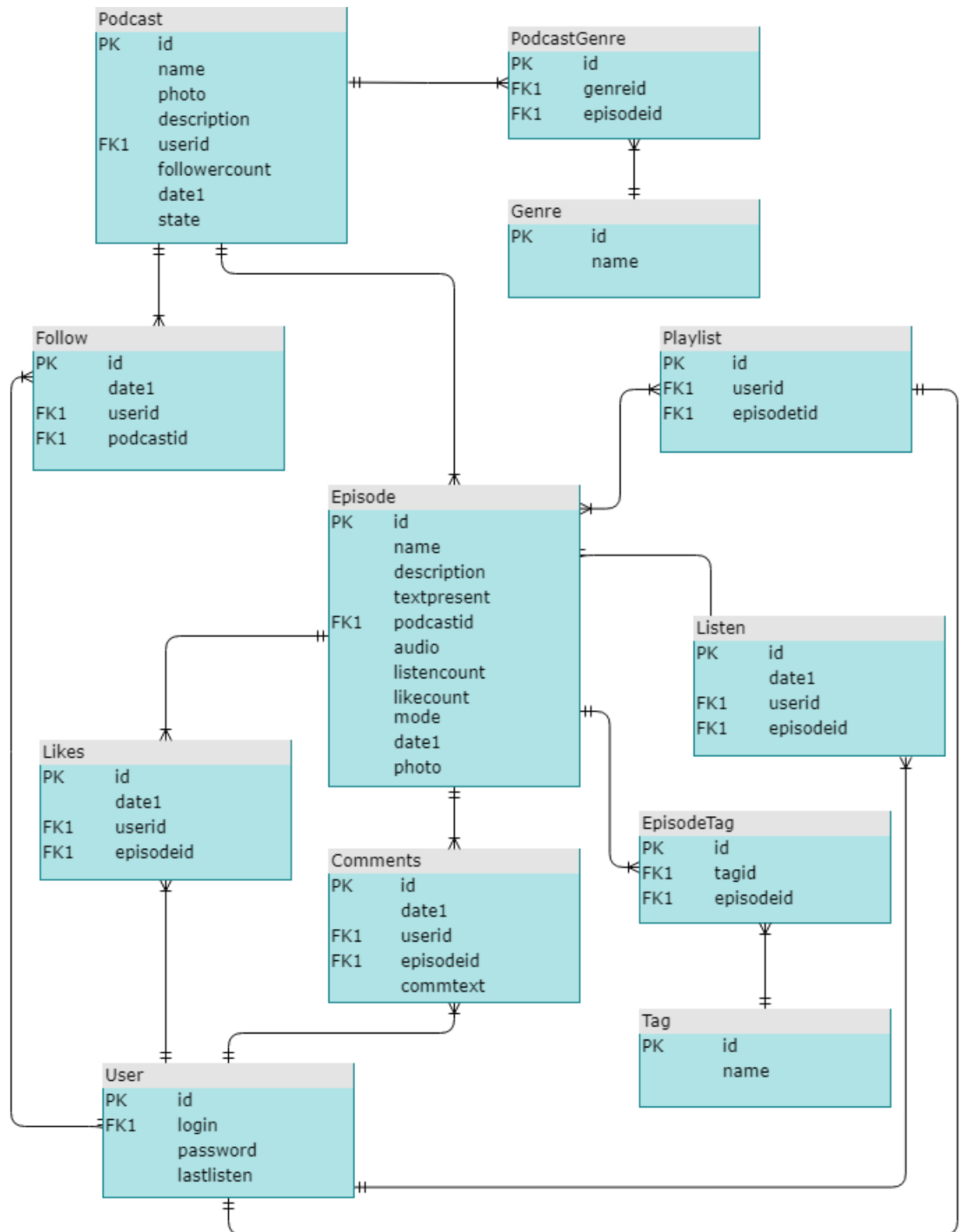


Рисунок 2.4 – Схема бази даних

Таблиця PodcastGenre представляє дані жанрів підкастів, реалізує

зв'язок багато-до-багатьох між таблицями Podcast і Genre, в структурі містить такі поля:

- id – ідентифікаційний номер запису;
- episodeid – епізод;
- genreid – жанр.

2.4 Визначення архітектури програмного забезпечення

Кожного разу, коли користувач вводить Uniform Resource Locator (URL)-адресу, браузер робить запит на сервер (рис. 2.5), який надсилає файли – це, як правило, вебсторінки, і браузер обробляє отриманий HyperText Markup Language (HTML)-код та відображає його, зрештою, незалежно від того, наскільки складним є вебзастосунок, все, що він робить, – це надсилання файлів за запитом, у свою чергу вебфреймворк надає набір компонентів для багаторазового використання, що дозволяє розробникам зосередитися на швидкому створенні свого застосунку: такі проблеми як, наприклад, як прив'язати певну URL-адресу до набору коду або як динамічно створювати HTML-сторінку відповідно до деяких даних з бази даних, виникають у кожному вебзастосунку, вебфреймворк дозволяє виконати ці дії швидко, щоб можна було зосередитися на написанні коду для вказаної URL-адреси або на вмісті певної сторінки, ці дві основні проблеми називаються маршрутизацією та шаблонізацією відповідно і є основними компонентами багатьох вебфреймворків [14]-[15].

У даній роботі в якості вебфреймворку, враховуючи зокрема вибір мови програмування Python, було використано найпопулярніший відповідний фреймворк – Django.

Дане рішення безпосередньо визначає те, яким чином буде сформовано структуру застосунку.

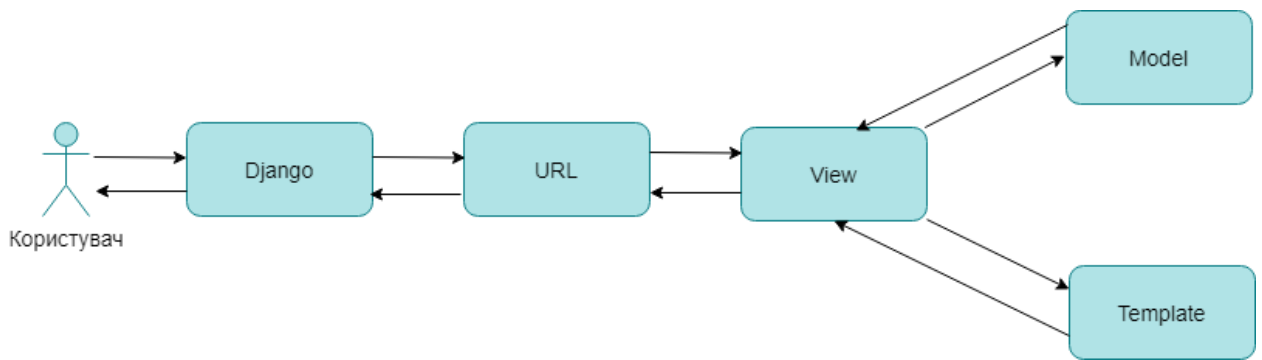


Рисунок 2.5 – Взаємодія користувача з вебзастосунком, побудованим на основі використання вебфреймворку Django

2.5 Висновки за розділом 2

За результатами виконання другого розділу було отримано наступні результати:

- вирішено розробляти програмне забезпечення у вигляді вебзастосунку;
- обрано мову програмування Python для реалізації програмного забезпечення;
- визначено функціональні вимоги до програми;
- побудовано діаграми прецедентів неавторизованого користувача, автора підкасту та слухача;
- визначено структуру бази даних;
- обрано MySQL в якості СУБД;
- обрано вебфреймворк Django для створення програмного забезпечення.

3 РОЗРОБКА ПРОГРАМИ

3.1 Розроблені модулі

Структуру модулів програми було обрано у відповідності зі схемою взаємодії, представленою на рис. 2.5. Відповідно основними одиницями було визначено:

- модуль `manage.py`, який відповідає за запуск застосунку в ролі сервера;
- модуль `forms.py`, який відповідає за створення форм;
- модуль `urls.py`, який відповідає за перенаправлення запитів користувача на відповідні представлення;
- модуль `views.py`, який містить набір представлень;
- модуль `settings.py`, який містить налаштування застосунку;
- модуль `models.py`, який містить моделі застосунку;
- папка `static`, яка містить статичні елементи застосунку, включаючи зображення, таблиці стилів, скрипти;
- папка `templates`, яка містить набір шаблонів застосунку.

До переліку шаблонів застосунку у відповідності з визначеними функціональними вимогами було включено:

- `index.html` – шаблон базової сторінки, визначає перегляд переліку підкастів;
- `history.html` – шаблон сторінки доступу до історії прослуховування користувачем епізодів підкастів;
- `genre_podcasts.html` – шаблон сторінки перегляду підкастів за жанрами (фактично має два варіанти відображення: для авторизованого та неавторизованого користувача);
- `followed_podcasts.html` – шаблон сторінки перегляду підкастів, на які підписаний користувач;
- `user_podcasts.html` – шаблон сторінки власних підкастів користувача;

– `podcast_episodes.html` – шаблон сторінки перегляду епізодів підкасту (фактично має два варіанти відображення: для авторизованого та неавторизованого користувача);

– `followed_episodes.html` – шаблон сторінки перегляду епізодів підкастів, на які підписався користувач;

– `search_results.html` – шаблон сторінки пошуку і перегляду результатів пошуку (фактично має два варіанти відображення: для авторизованого та неавторизованого користувача);

– `tag_episodes.html` – шаблон сторінки перегляду епізодів, що відповідають заданому тегу;

– `episode.html` – шаблон сторінки перегляду конкретного епізоду підкасту (фактично має два варіанти відображення: для авторизованого та неавторизованого користувача);

– `add_tag.html` – шаблон сторінки створення тегу;

– `episode_create.html` – шаблон сторінки створення нового епізоду підкасту;

– `episode_edit.html` – шаблон сторінки редагування даних існуючого епізоду підкасту;

– `podcast_edit.html` – шаблон сторінки редагування даних підкасту;

– `podcast_create.html` – шаблон сторінки створення нового підкасту;

– `playlist.html` – шаблон сторінки прослуховування власного плейлиста епізодів;

– `user_podcast_episodes.html` – шаблон сторінки роботи з епізодами власного підкасту;

– `login.html` – шаблон сторінки авторизації;

– `register.html` – шаблон сторінки реєстрації.

Відповідну описану структуру розробленої програми приведено на рис. 3.1.

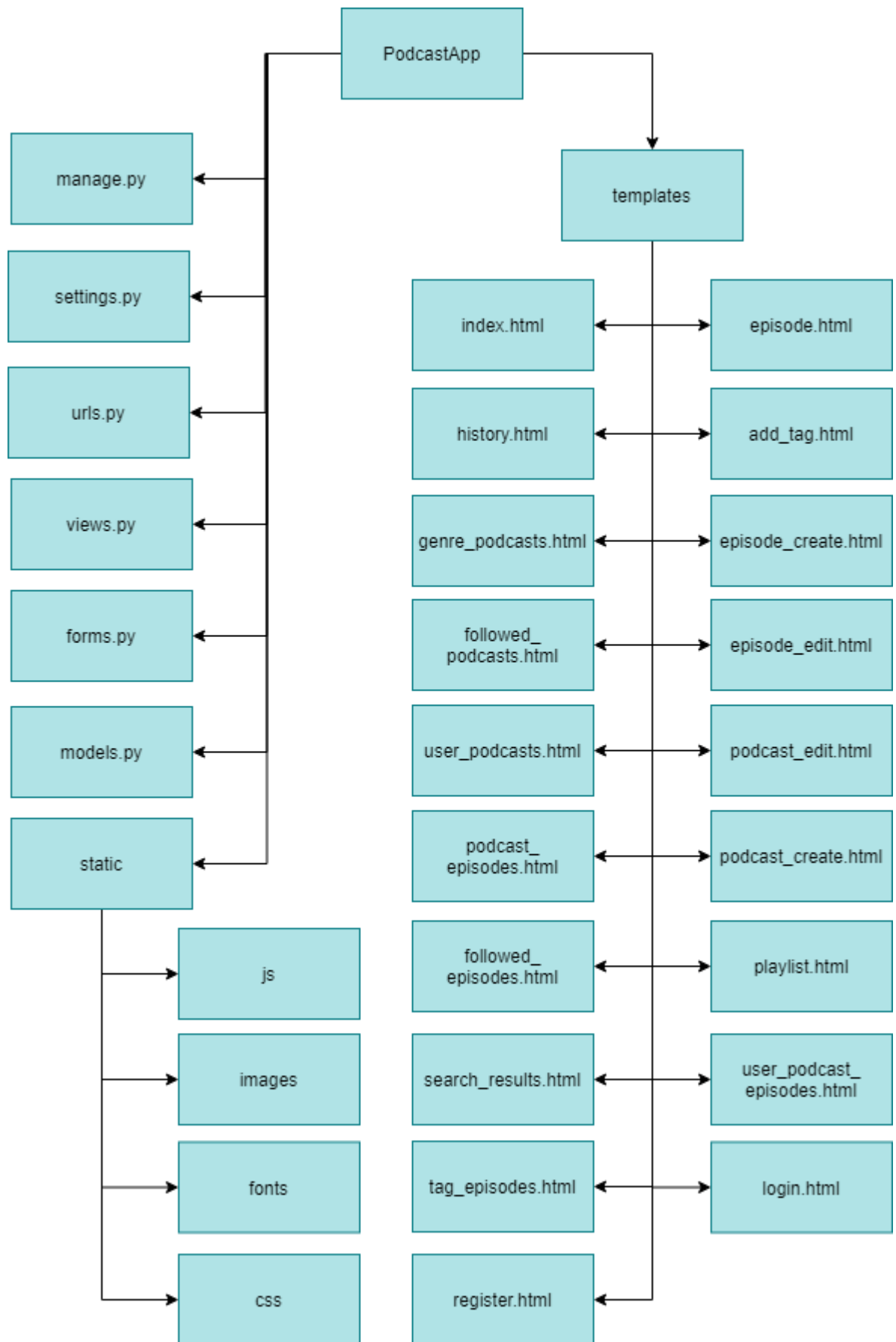


Рисунок 3.1 – Структура розробленої програми

3.2 Опис прийнятих рішень під час реалізації

У складі вебфреймворку Django представлено об'єктно-реляційне відображення, за допомогою якого можна визначати структуру даних на основі класів, а відображення цих класів в таблиці бази даних виконується автоматично.

Для цього було визначено набір класів моделей, поміщених у модуль `models.py`.

Клас `Podcast` створено шляхом наслідування класу `models.Model` для визначення моделі підкасту. Модель має у своєму складі наступні поля:

- `name` – поле `CharField`, що визначає назву підкасту;
- `photo` – поле `CharField`, що визначає шлях до файлу, в якому міститься зображення, встановлене у відповідність підкасту;
- `description` – поле `TextField`, яке визначає опис підкасту;
- `user` – поле `ForeignKey(User, on_delete=models.CASCADE)`, що визначає зовнішній ключ до моделі користувача і формує дані автора підкасту;
- `followercount` – поле `IntegerField()`, що визначає загальну поточну кількість підписників на підкаст, має перераховуватися кожен раз під час оформлення підписки або відписки від підкасту;
- `genres` – поле `ManyToManyField(Genre)`, що визначає зв'язок багато-до-багатьох до моделі `Genre`, тобто проміжна таблиця не створюється вручну, і встановлює набір жанрів, яким відповідає підкаст;
- `date1` – поле `DateTimeField()`, що визначає дату створення підкасту;
- `slug` – поле `SlugField(max_length=200, blank=True, unique=True)` представляє унікальний рядок, що характеризує підкаст і створюється програмно на основі виклику функції `slugify` з пакету засобів `django.utils.text`.

Аналогічно побудовано інші класи моделей. Моделі тегу `Tag` та жанру `Genre` складаються з одного поля `name`.

Клас моделі користувача `User` містить поля:

- login – поле CharField, що визначає логін користувача;
- password – поле CharField, що визначає пароль користувача;
- lastlisten – поле DateTimeField, що визначає дату і час останнього прослуховування епізодів користувачем, необхідно для того, що сформувати перелік нових епізодів підкастів, на які підписався користувач.

Клас моделі вподобання епізоду Likes містить поля:

- date1 – поле DateTimeField(), що визначає дату і час вподобання;
- userid – поле ForeignKey (User, on_delete=models.CASCADE) визначає зовнішній ключ до моделі користувача і формує дані про користувача, який вподобав епізод;
- episodeid – поле ForeignKey (Episode, on_delete=models.CASCADE) визначає зовнішній ключ до моделі епізоду і формує дані про епізод, який був уподобаний.

Клас моделі коментаря до епізоду Comments містить поля:

- date1 – поле DateTimeField(), що визначає дату і час створення коментаря;
- userid – поле ForeignKey (User, on_delete=models.CASCADE), що представляє зовнішнє поле і визначає дані автора коментаря;
- episodeid – поле ForeignKey (Episode, on_delete=models.CASCADE), що представляє зовнішнє поле і визначає дані епізоду;
- commtext – поле TextField(), що визначає текст коментаря.

Клас моделі прослуховування епізоду Listen містить поля:

- date1 – поле DateTimeField, що визначає дату і час прослуховування;
- userid – поле ForeignKey (User, on_delete=models.CASCADE), що представляє зовнішнє поле моделі користувача і формує дані слухача;
- episodeid – поле ForeignKey (Episode, on_delete=models.CASCADE) , що представляє зовнішнє поле моделі епізоду і формує дані епізоду, який було прослухано.

Клас моделі епізоду Episode містить поля:

- name – поле CharField, що визначає назву епізоду;

- description – поле TextField, що визначає опис епізоду;
- textpresent – поле CharField, що визначає шлях до текстового файлу з розшифруванням епізоду;
- photo – поле CharField, що визначає шлях до файлу, в якому міститься зображення, встановлене у відповідність епізоду;
- podcastid – поле ForeignKey (Podcast, on_delete=models.CASCADE), що визначає зовнішнє поле з моделі підкасту і формує дані підкасту, до якого належить цей епізод;
- audio – поле CharField визначає шлях до аудіофайлу, в якому збережено епізод підкасту;
- tags – поле ManyToManyField(Genre), що формує відношення багато-до-багатьох з моделлю тегів і представляє перелік тегів, якими відзначено епізод;
- slug – поле SlugField(max_length=200, blank=True, unique=True) представляє унікальний рядок, що характеризує епізод і створюється програмно на основі виклику функції slugify з пакету засобів django.utils.text.
- likecount – поле IntegerField, що визначає поточну кількість уподобань, поставлених епізоду і змінюється за кожного уподобання;
- listencount – поле IntegerField, що визначає поточну кількість прослуховувань, поставлених епізоду і змінюється за кожного прослуховування;
- mode – поле IntegerField(), що визначає режим взаємодії з епізодом непідписаних на підкаст користувачів;
- date1 – поле DateTimeField, що визначає дату і час додавання епізоду.

Клас моделі підписки на підкаст Follow містить поля:

- date1 – поле DateField(), що визначає дату оформлення підписки;
- userid – поле ForeignKey (User, on_delete=models.CASCADE), що визначає зовнішнє поле з моделі користувача і формує дані слухача епізоду;

– `podcastid` – поле `ForeignKey` (`Podcast`, `on_delete=models.CASCADE`), що визначає зовнішнє поле з моделі підкасту і формує дані підкасту, до якого належить цей епізод.

Клас моделі плейлиста `Playlist` містить поля:

– `userid` – поле `ForeignKey` (`User`, `on_delete=models.CASCADE`), що представляє зовнішнє поле моделі користувача і формує дані слухача;

– `episodeid` – поле `ForeignKey` (`Episode`, `on_delete=models.CASCADE`), що представляє зовнішнє поле моделі епізоду і формує дані епізоду, який внесено в перелік для прослуховування.

Для формування даних для роботи описаних у попередньому підрозділі шаблонів було сформовано набір відповідних представлень. Частина з них сформована у вигляді окремих класів, які створені шляхом наслідування відповідних базових класів моделей створення, редагування, а частина створені як окремі функції, які повертають набір потрібних даних.

Клас `PodcastsView` створює представлення, яке повертає перелік всіх доступних, тобто активних (неархівованих), підкастів, сортуючи його за кількістю підписників:

```
podcasts = super(PodcastsView, self).get_queryset()
podcasts = podcasts.filter()
for p in podcasts:
    if p.mode == 1:
        podcasts.remove(p)
podcasts = podcasts.order_by("followercount")
```

Клас `HistoryView` визначає представлення, що формує перелік прослуханих авторизованим користувачем епізодів, сортуючи його за датою і часом прослуховування.

Функція `genre_podcasts(request)` формує перелік підкастів, які відповідають жанру, назва якого передана через параметр запиту `request`:

```

genrename = request.GET.get('genre', '')
podcasts = Podcast.objects.filter(Q(genres__name__contains=genrename))

```

Окрім того з даного переліку функція відділяє 9 перших за кількістю підписників підкастів і повертає їх окремо, а з переліком усіх інших виконує керування відображенням по сторінках, на кожній з яких відображаючи по 9 підкастів і формуючи відповідні посилання `before` і `after` для переходу на сторінку назад та на сторінку вперед:

```

pin = Paginator(podcasts, 9)
cp = request.GET.get('page', 1)
res = pin.get_page(cp)
paginatorflag = res.has_other_pages()
if res.has_previous():
    before = '?page={}'.format(res.previous_page_number())
else:
    before = ""
if res.has_next():
    after = '?page={}'.format(res.next_page_number())
else:
    after = ""

```

Функція `followed_podcasts(request)` повертає перелік підкастів, на які підписався авторизований користувач з розподіленням об'єктів за сторінками по 9.

Клас `OwnPodcastsView` створює представлення для перегляд власних підкастів, повертаючи за запитом відповідний перелік, відсортований за ідентифікаційними номерами за зменшенням.

Клас `PodcastEpisodesView` формує представлення для роботи з епізодами підкасту. Повертається перелік епізодів, виключаючи з них ті, на які авторизований користувач має бути підписаний для прослуховування.

Клас `OwnPodcastEpisodesView` створює представлення для наповнення даними епізодів власного підкасту.

Функція `followed_episodes(request)` повертає перелік епізодів, які належать підкастам, на які підписався користувач, при цьому до переліку включаються тільки ті епізоди, які були опубліковані не пізніше ніж у момент останнього прослуховування будь-якого іншого епізоду авторизованим користувачем:

```

auser = request.user
follows = Follow.objects.filter(Q(userid=auser))
episodes = []
for f in follows:
    sa = Episode.objects.filter(Q(podcastid=f.podcastid))
    sa.filter(Q(date1__gte=auser.lastlisten))
    for seas in sa:
        episodes.append(seas)
episodes.order_by('-date1')
```

Епізоди повертаються з посторінковим розбиттям по 12 на сторінці.

Функція `search_podcasts(request)` формує представлення для реалізації пошуку за частиною назви підкасту.

Функція `tag_episodes(request)` формує представлення, яке повертає перелік епізодів, по 5 на сторінці, що містять заданий тег, отриманий з параметрів запиту користувача.

Функція `episode_detail(request, slug)` повертає детальні дані епізоду, а окрім того вибирає набір коментарів, які відповідають даному епізоду, та надає обробник додавання нового коментаря через відповідну форму.

Функція `like_episode(request)` реалізує представлення для обробки уподобань користувачів шляхом додавання відповідного уподобання до уподобань епізоду, якщо воно не було ще раніше створено даним авторизованим користувачем, або вилученням його з переліку, якщо таке уподобання вже було створено. При цьому одночасно змінюється загальна кількість уподобань епізоду у відповідній моделі.

Функція `listen_episode(request)` реалізує представлення для обробки прослуховувань користувачем епізодів шляхом додавання відповідного прослуховування до прослуховувань епізоду, якщо воно не було ще раніше створено даним авторизованим користувачем, або вилученням його з переліку, якщо таке прослуховування вже було внесено. При цьому одночасно змінюється загальна кількість прослуховувань епізоду у відповідній моделі.

Функція `follow_podcast(request)` створює підписку на підкаст для користувача або вилучає її, якщо вона вже була створена раніше, відповідним чином також змінюючи кількість підписників підкасту.

Клас `CreatePodcastView` реалізує представлення створення нового підкасту, клас `EditPodcastView` – його редагування, клас `CreateEpisodeView` – створення епізоду підкасту, `EditEpisodeView` – редагування епізоду.

Клас `AddTagView` забезпечує представлення для створення нового тегу.

Функція `to_playlist` виконує додавання поточного епізоду, визначеного в параметрах запита, в плейлист користувача, якщо він ще не був доданий.

Функція `archive_podcast(request)` забезпечує представлення для архівування підкасту.

Клас `PlaylistView` реалізує представлення плейлиста епізодів користувача.

3.3 Висновки за розділом 3

Представлено структуру розробленої програми з приведенням структурної схеми програми. Описано створені шаблони застосунку, моделі та представлення, які виконують формування даних моделей для виведення в шаблонах.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програмне забезпечення надає доступ до пізнавальних підкастів щодо їх зручного прослуховування з привертанням уваги до самоосвіти протягом всього життя людини. Програма призначена для роботи з підкастами шляхом внесення їх епізодів та прослуховування їх слухачами. Слухачі можуть виражати власні інтереси шляхом підписування на підкасти, створення власних плейлистів, встановлення вподобань.

4.2 Умови виконання програми

Мінімальні системні вимоги до вебсервера включають:

- процесор: 4 ядра, тактова частота – 2,9 ГГц;
- оперативна пам'ять – 10 Гб;
- платформа – 32-розрядна або 64-розрядна;
- жорсткий диск – 1 Гб вільного простору.

4.3 Робота з програмою

Після звернення до вебсервера відкривається початкова сторінка вебзастосунку (рис. 4.1).

Основна сторінка містить:

- перелік найпопулярніших підкастів, представлений у вигляді каруселі, що дозволяє переключатися між підкастами, які мають найбільшу кількість підписників, за допомогою стрілок вліво та вправо: перейти до обраного підкасту можна натисканням на посилання у назві підкасту та зображенні;

- перелік всіх інших підкастів у вигляді зображень з переключенням між сторінками.

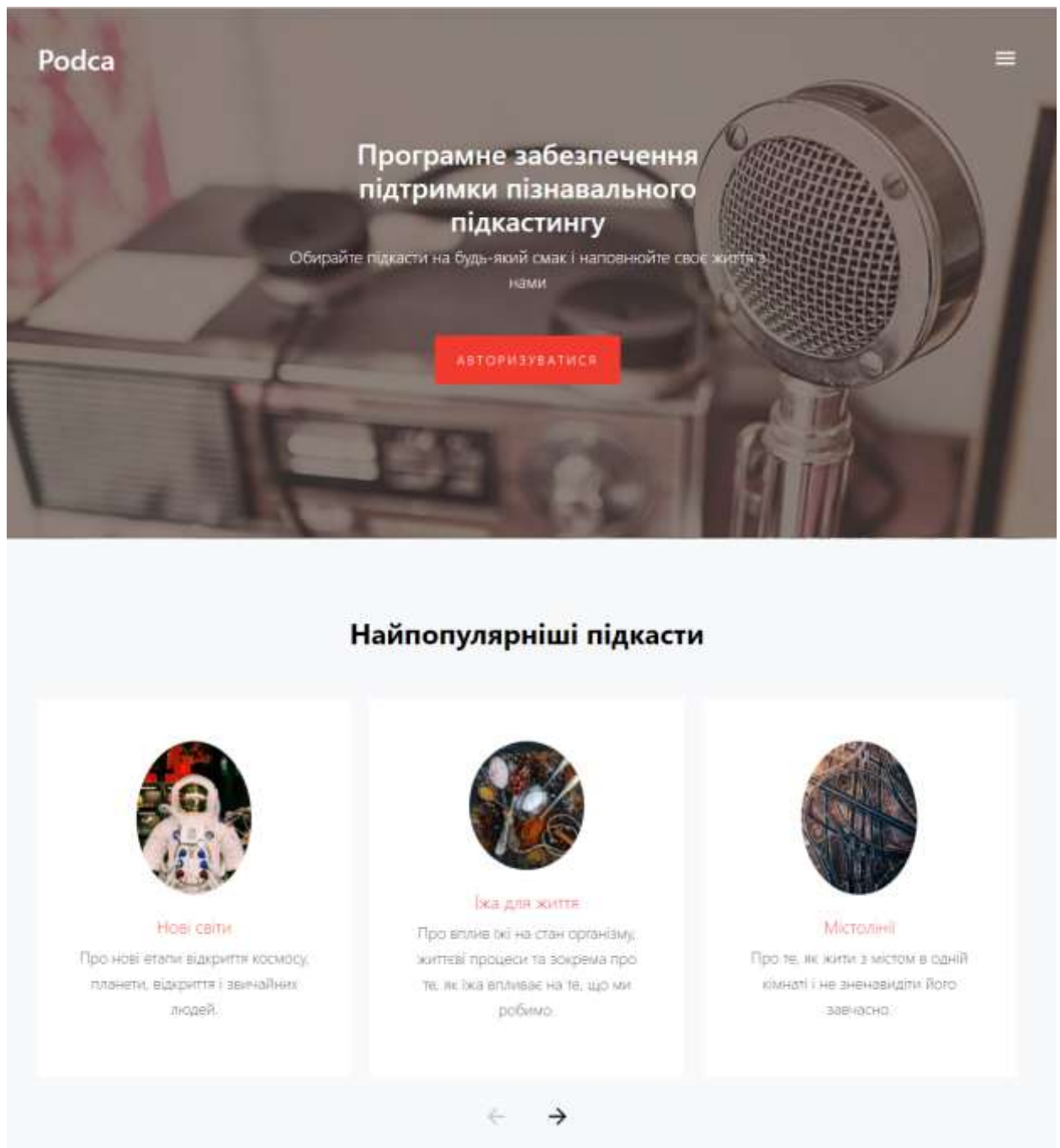


Рисунок 4.1 – Початкова сторінка

Основна сторінка дозволяє одразу перейти до авторизації за допомогою відповідної кнопки «Авторизуватися».

Повний перелік пунктів меню застосунку включає пункти:

- «Підкасти», що дозволяє перейти до перегляду початкової сторінки;
- «Жанри», що дозволяє перейти до вибору конкретного жанру і перегляду підкастів за обраним жанром;

– «Власні підкасти» (доступний тільки авторизованому користувачу), що включає такі підпункти: «Перелік» (перелік всіх підкастів, звідки можна перейти до редагування підкасту), «Новий підкаст» (забезпечує створення нового підкасту), «Новий епізод» (забезпечує створення нового епізоду в межах підкасту);

– «Плейлист» (доступно тільки авторизованому користувачу), що дозволяє прослуховувати обрані користувачем епізоди, додані до переліку користувачем;

– «Пошук», що дозволяє виконувати пошук підкастів за назвою або її частиною;

– «Підписки» (доступно тільки авторизованому користувачу) включає пункти «Перелік», що відображає перелік виконаних підписок користувачем, «Нові епізоди», що дозволяє переглянути нові епізоди підкастів, на які підписаний користувач, що з'явилися з моменту останнього прослуховування користувачем будь-якого епізоду;

– «Історія» (доступно тільки авторизованому користувачу), що дозволяє переглянути перелік епізодів, які раніше прослухав користувач;

– «Увійти»/«Вийти» – дозволяє авторизуватися або вийти з облікового запису.

Сторінка підкасту (рис. 4.2) містить зображення підкасту, його назву, логін автора, дату публікації, опис підкасту, загальну кількість підписників та загальну кількість епізодів. Якщо користувач не підписаний на підкаст, то для нього відображається кнопка «Підписатися». Якщо користувач підписаний, то відображається кнопка «Відписатися».

Нижче на сторінці відображається перелік епізодів підкасту. Епізоди розташовані в порядку від останнього до самого раннього.

Кожен епізод містить відповідне зображення (або стандартне зображення, якщо автор не визначив окреме), опис епізоду, його назву, загальну кількість уподобань, а також посилання для уподобання епізоду,

якщо користувач ще не вподобав епізод, і посилання «забрати лайк», якщо користувач вже уподобав епізод, але можливо хоче його прибрати.

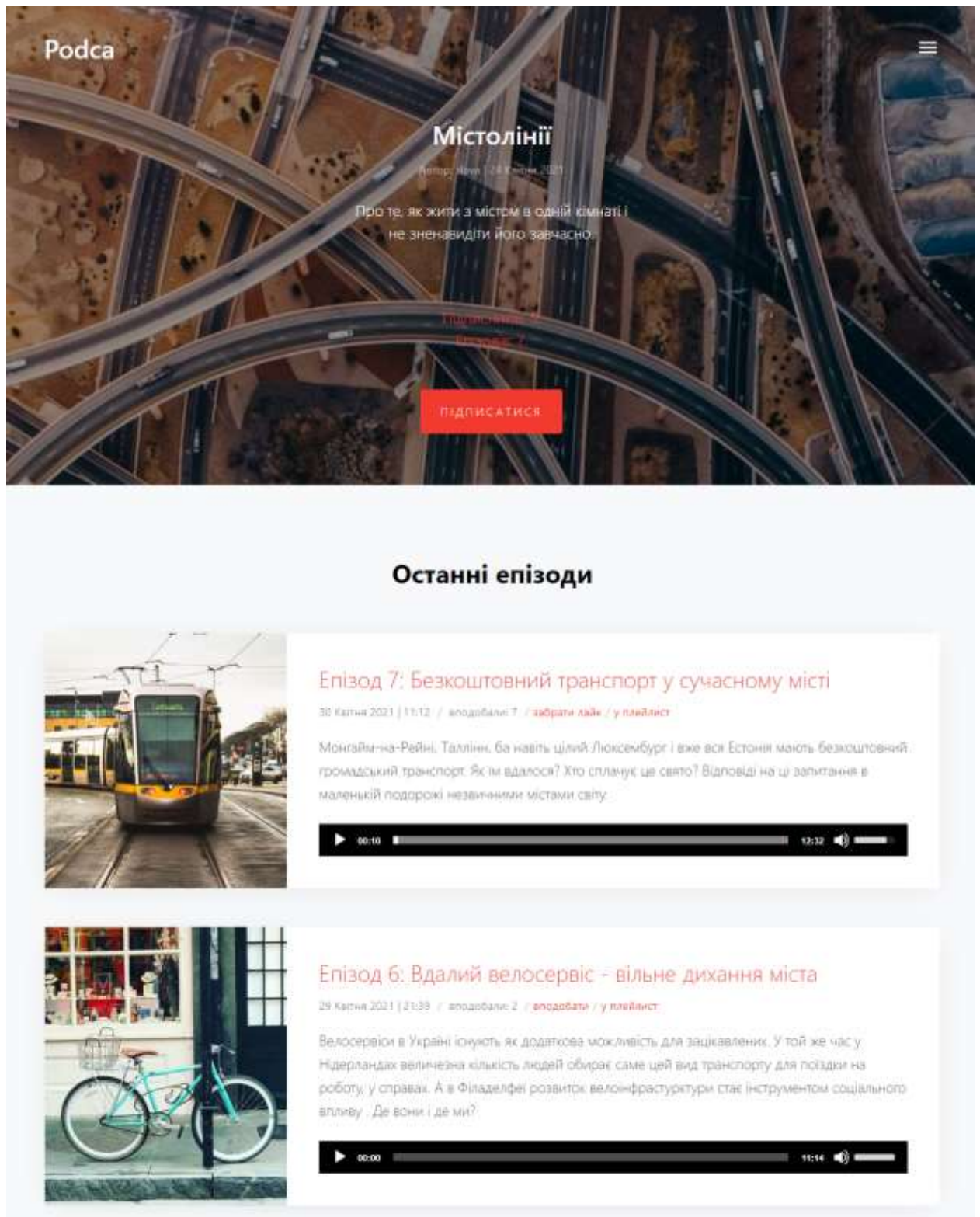


Рисунок 4.2 – Сторінка підкасту

Для кожного епізоду відображається програвач для відтворення епізоду.

На одній сторінці відображається 5 епізодів підкасту. Для переключення між сторінками створені відповідні кнопки для переходу між сторінками (рис. 4.3). Також внизу сторінку перелічено усіх користувачів, які підписалися на цей підкаст, та усі жанри, до яких автор відніс даний підкаст.

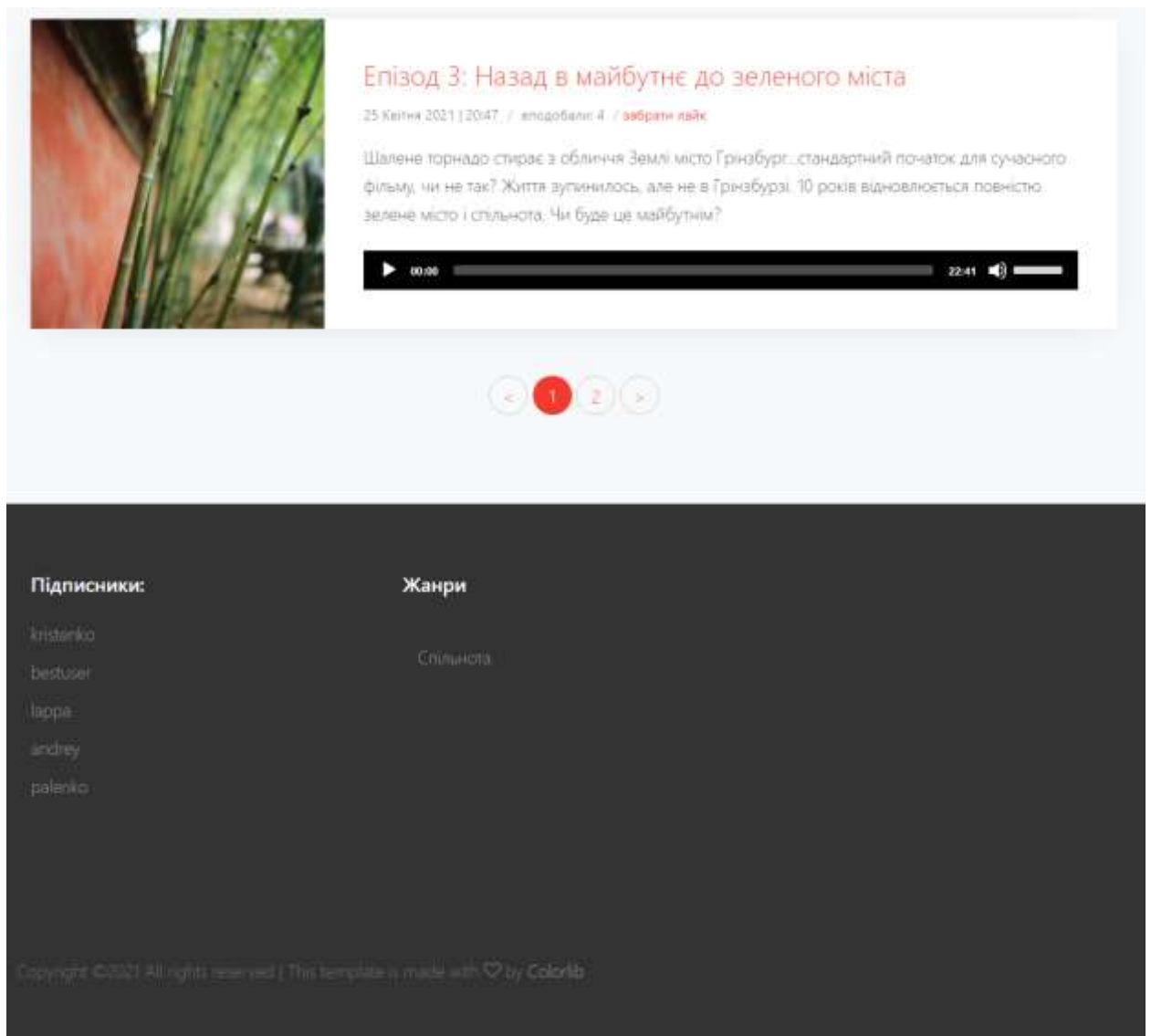


Рисунок 4.3 – Нижня частина сторінки підкасту

На сторінці перегляду і прослуховування епізоду (рис. 4.4) користувач може отримати повну інформацію про конкретний епізод.

Якщо сторінку відкрив автор підкасту, то йому у верхній частині демонструється кнопка «Редагувати». Якщо сторінку відкрив звичайний слухач, то йому відображається кнопка «Вподобати».

Нижче розташовано програвач для відтворення епізоду, загальна інформація про епізод, включаючи кількість уподобань, прослуховувань (відображається також тільки автору підкасту, якому належить даний епізод), набір тегів епізоду, опис епізоду (виділений жирним) та текстове розшифрування епізоду (якщо воно було додано).

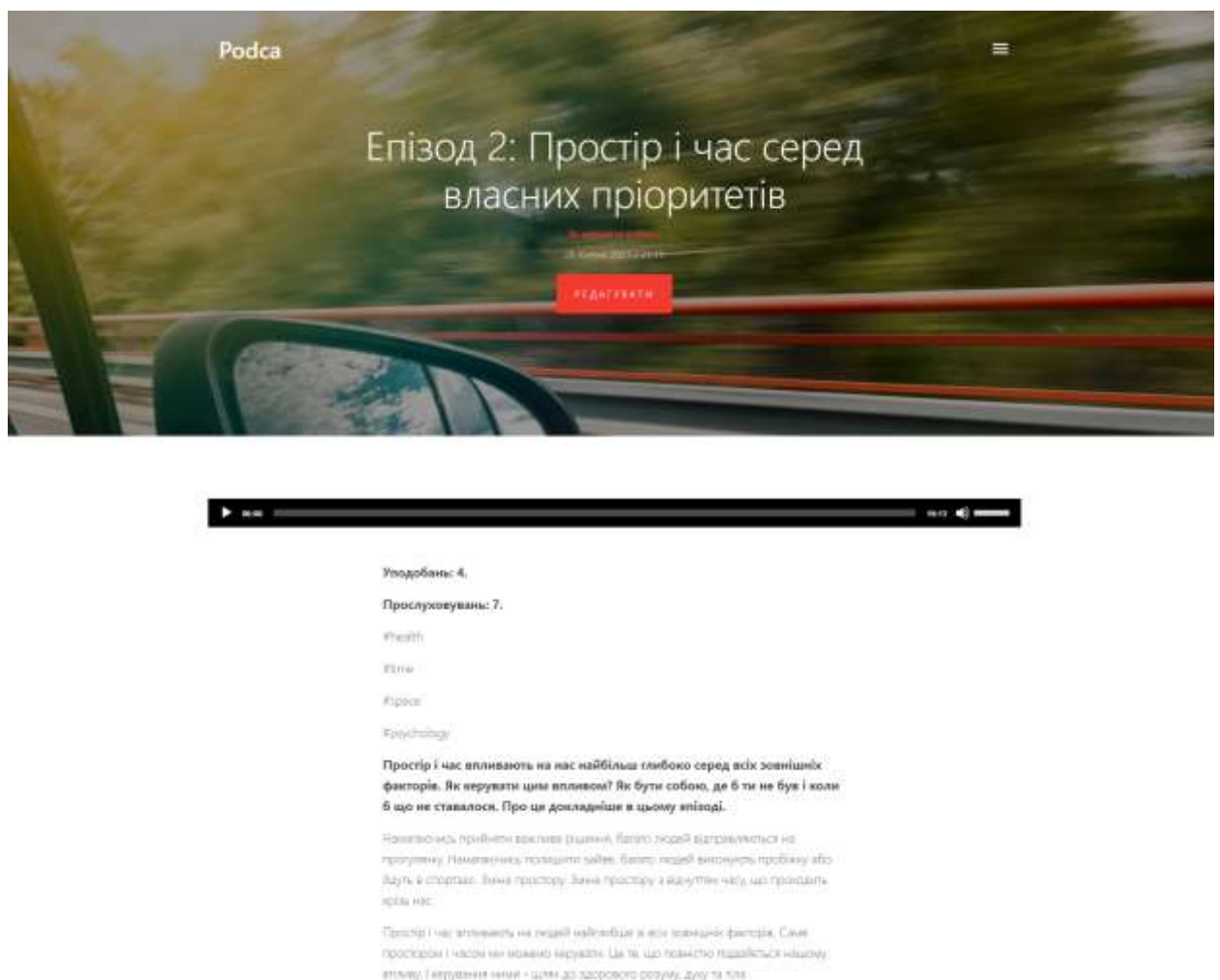


Рисунок 4.4 – Сторінка епізоду підкасту

В нижній частині сторінки (після розшифрування) відображається кнопка для додавання епізоду в плейлист, набір коментарів, які залишили

інші користувачі, та поле для введення власного нового коментаря. За допомогою кнопки «Додати» можна відправити введений коментар (рис. 4.5).

На цьому наш сьогоднішній випуск завершено. Дякую і далі буде далі.

У ПЛЕЙЛИСТ

Коментарі:

kristenko, 29.04.2021 22:51

Знайшов багато цікавого для себе, дякую за цей епізод

slava, 30.04.2021 01:11

Знайшов майже випадково підкаст, але цей епізод надіюсь підписатися

Ваш коментар

ДОДАТИ

Інші епізоди



Епізод 3: Як їсти правильно в неправильний час?

1 Травня 2021 / 20:37

Майже всі знають, коли потрібно їсти протягом дня, але не всі так роблять. Що ж до того, якщо все-таки посночно поїсти або перекусити у відведений час не вдалося, то як зробити це з найбільшою користю або найменшими втратами для свого організму?



Епізод 1: Чому нам стає боляче?

26 Квітня 2021 / 14:12

Ми відчуваємо біль, зайомлячись зі світом. Біль фізичний, психологічний. Він може бути тимчасовим, переслідувати нас протягом життя. Ми можемо утікати від нього, а хтось навпаки прагне його. Звідки виникає біль? Як його перемогти? І як уникнути болюних відчуттів? Про це в першому епізоді.



Рисунок 4.5 – Нижня частина сторінки епізоду підкасту

Нижче коментарів на сторінці розташовано попередній та наступний епізоди підкасту, до яких одразу може перейти користувач з метою подальшого прослуховування, наприклад, коли знайомиться з підкастом. Для кожного епізоду демонструється його зображення, назва, дата і час додавання та короткий опис.

Сторінка перегляду підкастів за жанром відкривається при виборі пункту меню «Жанри» (рис. 4.6). На даній сторінці відображається у верхній частині перелік всіх жанрів підкастів, які було створено у вебзастосунку. Після вибору конкретного жанру шляхом натискання на відповідне посилання відкривається перелік підкастів, які відповідають цьому жанру.

Спочатку відображається перелік найпопулярніших підкастів за цим жанром, представлений каруселлю, що дозволяє переходити між підкастами.

Нижче на сторінці розташовано перелік підкастів у вигляді плиток з посторінковим переходом. На кожній сторінці відображається 9 підкастів. Підкаст представлений зображенням підкасту. Якщо навести курсор на зображення, то відобразиться його назва, автор та опис підкасту. Для відображення повної інформації за підкастом та всіх його епізодів потрібно натиснути на дане зображення.

Сторінка перегляду нових епізодів відкривається при виборі підпункту меню «Нові епізоди» пункту меню «Підписки». Користувачу відображається перелік епізодів, які належать підкастам, на які він в даний момент підписаний, і які було опубліковано пізніше ніж момент його останнього прослуховування. Тобто кожен раз, коли користувач натискає на кнопку відтворення програвача, дата і час останнього його прослуховування оновлюється, а кількість прослуховувань відповідного епізоду збільшується. Фіксується кількість унікальних прослуховувань, тобто один користувач може прослухати епізод тільки один раз. Наступні відтворення йому даного епізоду фіксуватися не будуть. Якщо почати прослуховувати один з епізодів, то після його завершення автоматично буде відтворено наступний епізод на сторінці. Аналогічно побудовано відтворення плейлиста для користувача.

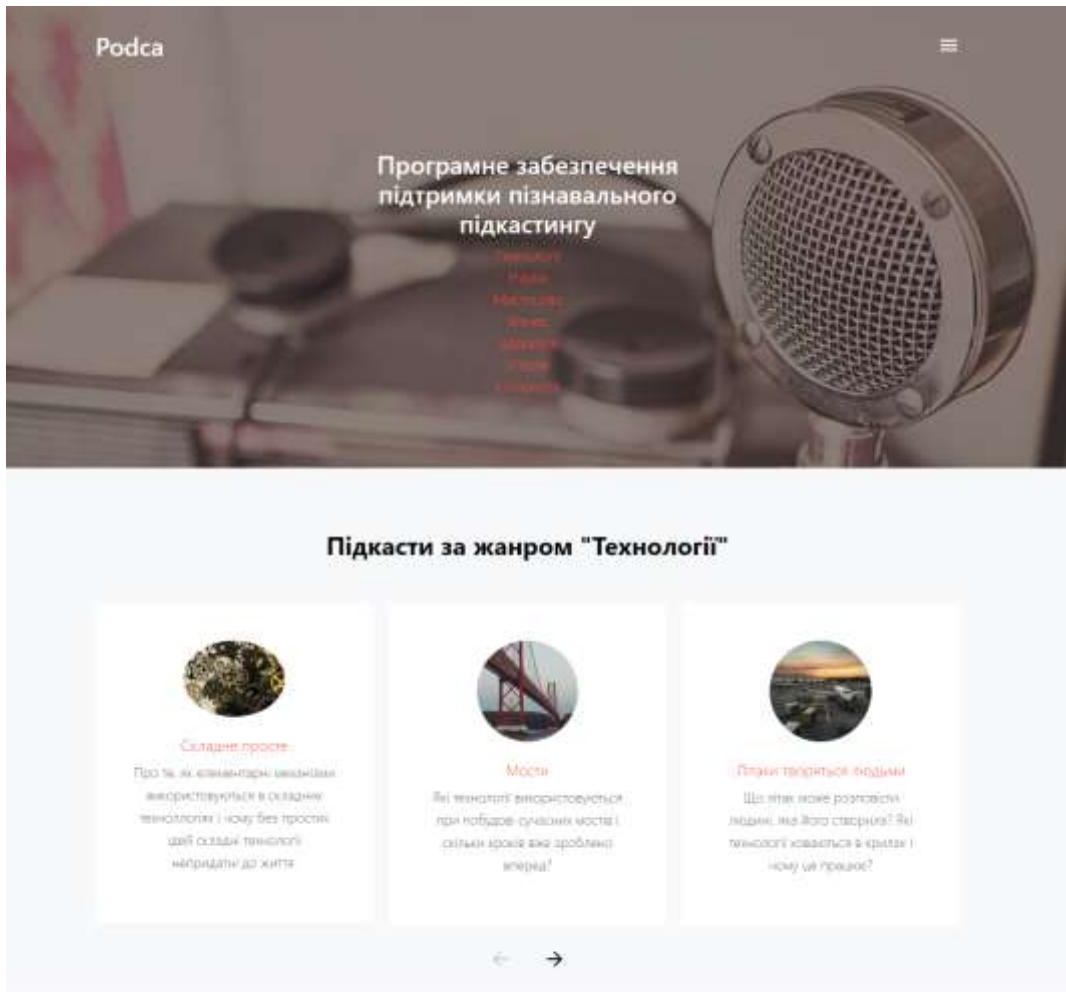


Рисунок 4.6 – Сторінка перегляду підкастів за жанром

На рис. 4.7 представлено сторінку перегляду нових епізодів.

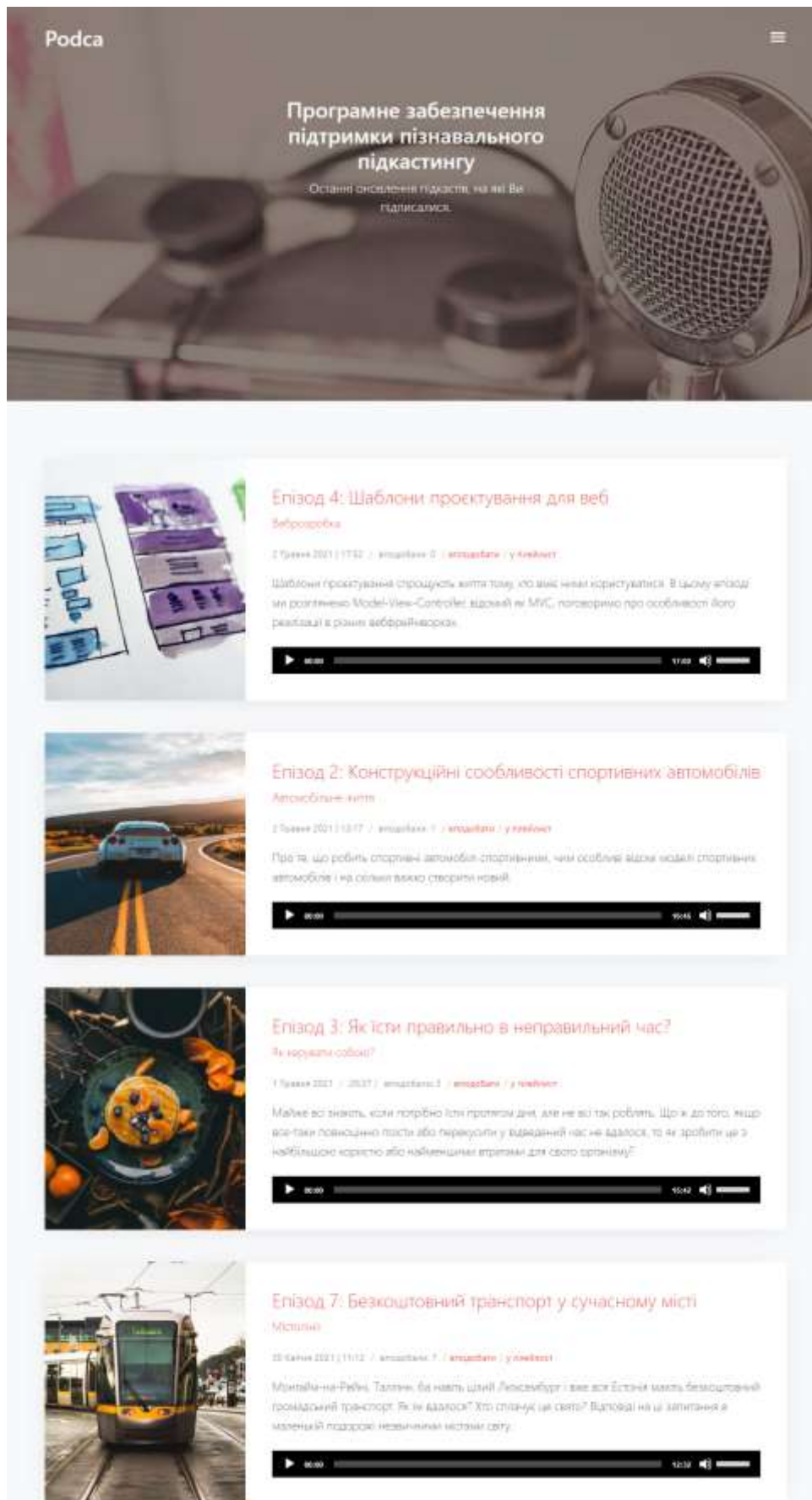
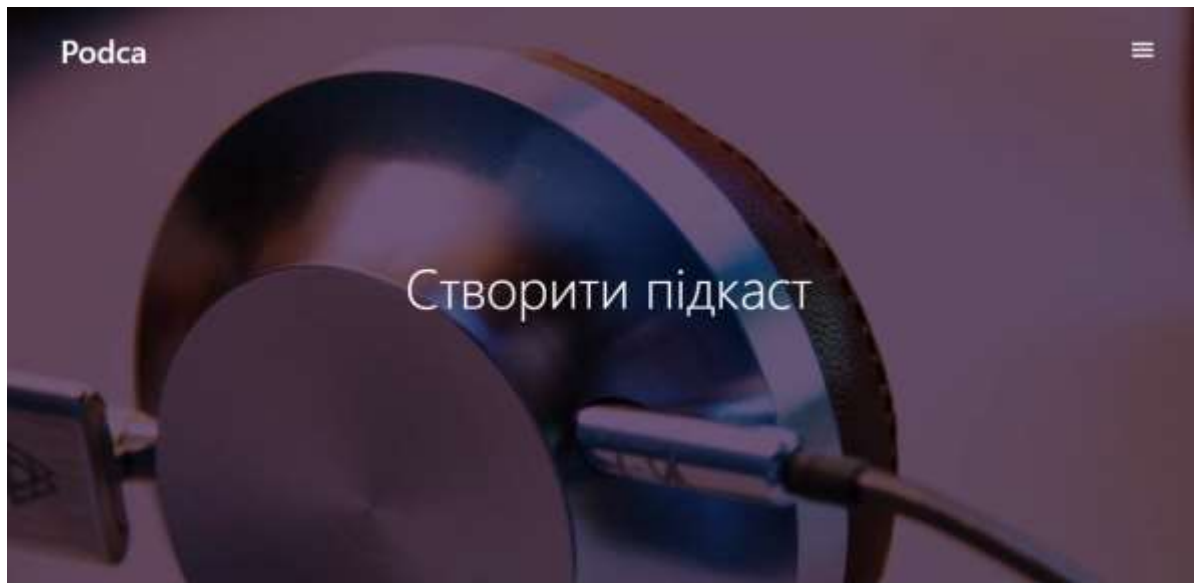


Рисунок 4.7 – Сторінка перегляду нових епізодів підкастів, на які підписаний користувач

Під час створення власного підкасту користувач повинен визначити його назву, обрати один або декілька з доступних жанрів за допомогою кнопок «Додати» і «Вилучити», обрати файл зображення, визначити опис. Завантажене зображення одразу відображається правіше (рис. 4.8).



Назва*

Жанр*

Технології

▼

ДОДАТИ

ВИЛУЧИТИ

Додано:

Технології

Зображення

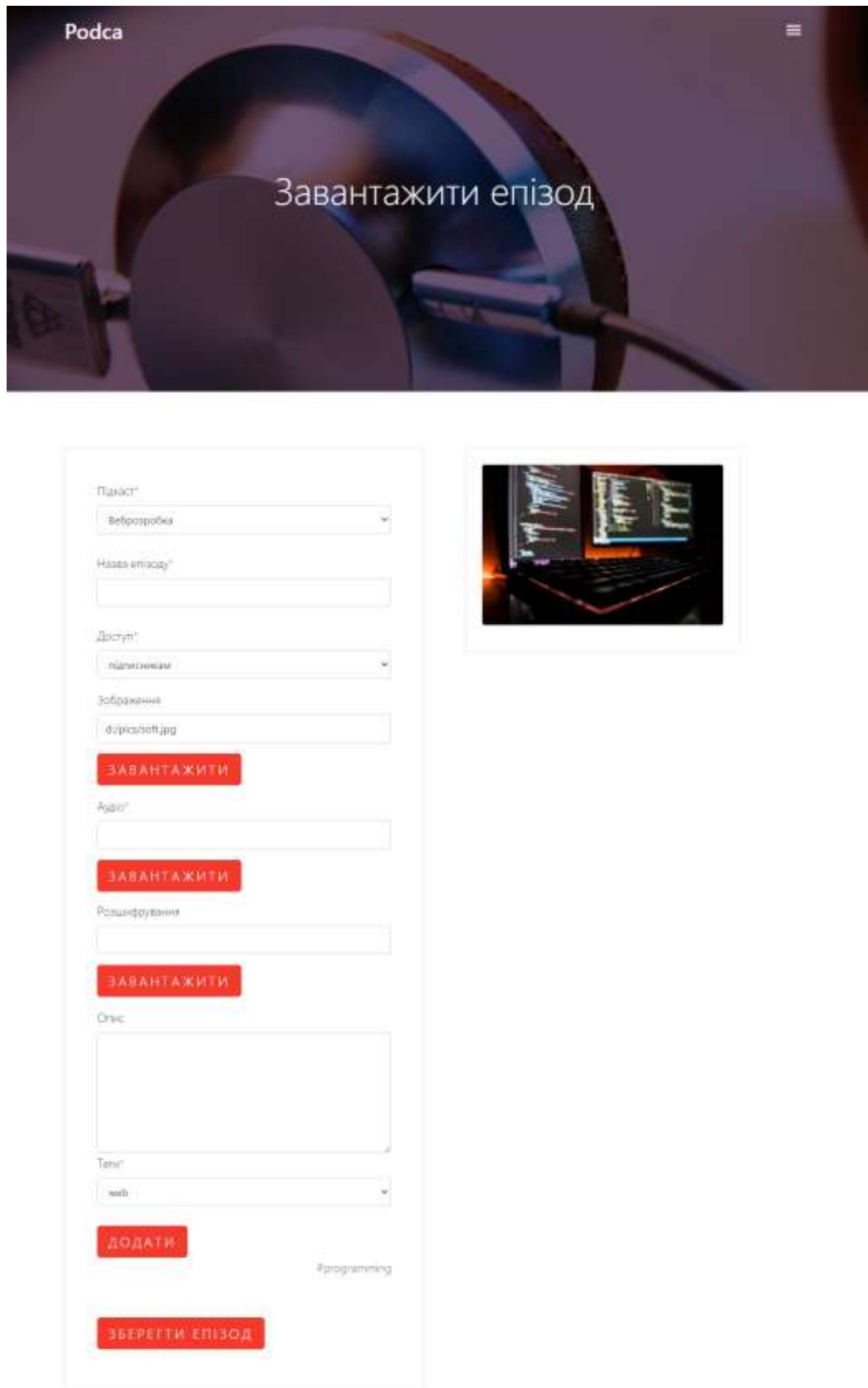
ЗАВАНТАЖИТИ

Опис:

СТВОРИТИ

Рисунок 4.8 – Сторінка створення нового підкасту

Після створення підкасту можна завантажувати епізоди (рис. 4.9).



The image shows a web interface for a podcasting application. At the top, there's a header with the word 'Podca' and a hamburger menu icon. Below the header, a large banner features a close-up of a microphone with the text 'Завантажити епізод' (Upload episode) overlaid. Below the banner, there's a form for adding a new episode. The form includes several input fields and dropdown menus, each followed by a red 'ЗАВАНТАЖИТИ' (Upload) button. The fields are: 'Підкаст:' (Podcast) with a dropdown menu showing 'Веброзробка'; 'Назва епізоду:' (Episode name); 'Доступ:' (Access) with a dropdown menu showing 'підписникам' (subscribers); 'Зображення:' (Image) with a text input field containing 'd3riscu50tt.jpg'; 'Аудіо:' (Audio) with a text input field; 'Розшифрування:' (Encryption) with a text input field; 'Опис:' (Description) with a large text area; and 'Тег:' (Tag) with a dropdown menu showing 'web'. At the bottom of the form, there's a red 'ДОДАТИ' (Add) button and a hashtag input field containing '#programming'. A final red button at the bottom is labeled 'ЗБЕРЕГТИ ЕПІЗОД' (Save episode). To the right of the form, there's a small thumbnail image of a laptop screen displaying code.

Рисунок 4.9 – Сторінка додавання нового епізоду до підкасту

Завантаження епізодів доступне через відповідний пункт меню, а також при відкритті сторінки підкасту його автором. Автору доступна кнопка «Створити епізод».

На сторінці створення епізоду потрібно задати підкаст спочатку, до якого завантажуватиметься епізод. Якщо перехід було виконано зі сторінки підкасту, то вибір буде виконано автоматично. Далі потрібно визначити назву епізоду, метод доступу (всім або тільки підписникам), файл зображення, файл аудіо, файл розшифрування (потрібно задати шлях до файлів і натиснути на кнопку «Завантажити»), сформулювати опис епізоду, набір тегів (кнопка «Вилучити» доступна тоді, коли обрати з переліку тег, який вже було додано, додані теги відображаються нижче з вирівнюванням по правій стороні). У кінці потрібно натиснути на кнопку «Зберегти епізод» для збереження внесених даних.

Усі обов'язкові для заповнення пункти відзначені червоною зірочкою.

ВИСНОВКИ

У роботі проаналізовано концепцію підкастингу, виділено основні особливості підкастів, причини сучасної популярності підкастів та принципи їх організації. Визначено пізнавальні підкасти в якості основного призначення розроблюваного програмного забезпечення. Розглянуто основні програмні засоби хостингу підкастів та проаналізовано їх функціональні можливості та принципи організації.

Програмне забезпечення спроектовано та реалізовано у вигляді вебзастосунку. Обрано мову програмування Python для реалізації програмного забезпечення. Визначено функціональні вимоги до програми, побудовано діаграми прецедентів неавторизованого користувача, автора підкасту та слухача. Визначено структуру бази даних, обрано MySQL в якості СУБД, обрано вебфреймворк Django для створення програмного забезпечення.

Реалізоване програмне забезпечення надає доступ до пізнавальних підкастів щодо їх зручного прослуховування з привертанням уваги до самоосвіти протягом всього життя людини. Використання даного програмного забезпечення за належного наповнення підкастами має призвести до створення цілої спільноти. Програмне забезпечення призначене для роботи з підкастами шляхом внесення їх епізодів та прослуховування їх слухачами. Слухачі можуть виражати власні інтереси шляхом підписування на підкасти, створення власних плейлистів, встановлення вподобань, взаємодіючи з підкастами як шляхом пошуку, так і шляхом перегляду нових епізодів з переліку підписаних, керування епізодами, які йому відтворюються.

Усі завдання, які було визначено на дипломну кваліфікаційну роботу, повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Watts, M. The Rise in Podcast Popularity: How Can Retailers Utilise This Trend? [Electronic resource] / M. Watts. – Access mode : <https://www.silverdisc.co.uk/blog/2019/08/02/rise-podcast-popularity-how-can-retailers-utilise-trend>
2. What Is A Podcast And How Do They Work? [Electronic resource]. – Access mode : <https://www.podcastinsights.com/what-is-a-podcast/>
3. Sibirtseva, M. Why Are Podcasts So Popular? A Look at Their Past, Present, and Future [Electronic resource] / M. Sibirtseva. – Access mode : <https://blog.depositphotos.com/why-are-podcasts-so-popular.html>
4. Why are Podcasts becoming more popular | Benefit of Podcasts [Electronic resource]. – Access mode : <https://pplprs.co.uk/popularity-of-podcasts/>
5. Terra, E. 5 Universal Listener Experience (LX) Principles for Podcasting [Electronic resource] / E. Terra. – Access mode : <https://medium.com/the-firm-voice/what-makes-a-good-podcast-listener-experience-28d1772b3398>
6. Spinelle, J. Podcasting Advice for Professors [Electronic resource] / J. Spinelle. – Access mode : <https://www.insidehighered.com/advice/2020/03/26/how-create-effective-podcasts-spread-word-about-your-work-and-make-new-connections>
7. Free Podcast Hosting – Starting a Podcast in 5 Minutes | Podbean [Electronic resource]. – Access mode : <https://www.podbean.com/>
8. 7 Best Podcast Hosting for 2021 Compared (Most are Free) [Electronic resource]. – Access mode : <https://www.wpbeginner.com/showcase/best-podcast-hosting-compared-most-are-free/>
9. Best free podcast app for Apple iOS and Android | Let words move you [Electronic resource]. – Access mode : <https://castbox.fm/>
10. Mixcloud – This is Audio Culture | Mixcloud [Electronic resource]. –

Access mode : <https://www.mixcloud.com/>

11. Головна – «Радіо Поділ» [Електрон. ресурс]. – Режим доступу : <https://radiopodil.org/>
12. Lubanovic, B. Introducing Python [Text] : Modern Computing in Simple Packages / B. Lubanovic. – Sebastopol, CA : O'Reilly Media, 2019. – 630 p.
13. Lutz, M. Programming Python [Text] : Powerful Object-Oriented Programming / M. Lutz. – Sebastopol, CA : O'Reilly Media, 2011. – 1632 p.
14. Ko, T. A Quick Glance of Django [Electronic resource] / T. Ko. – Access mode : <https://medium.com/@timmykko/a-quick-glance-of-django-for-beginners-688bc6630fab>
15. The web framework for perfectionists with deadlines [Electronic resource]. – Access mode : <https://www.djangoproject.com/>
16. Podca – Free Bootstrap 4 HTML5 podcast website template [Electronic resource]. – Access mode : <https://themewagon.com/themes/free-bootstrap-4-html5-podcast-website-template-podca/>

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення має надавати доступ до пізнавальних підкастів щодо їх зручного прослуховування з привертанням уваги до самоосвіти протягом всього життя людини.

A.1 Підстави для розробки

Підставою для виконуваної розробки в межах дипломної кваліфікаційної роботи бакалавра є завдання, що було визначене за темою «Програмне забезпечення підтримки пізнавального підкастингу» і затверджене у відповідності з наказом № 103 від 30 березня 2021 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програмне забезпечення призначене для роботи з підкастами шляхом внесення їх епізодів та прослуховування їх слухачами. Слухачі можуть виражати власні інтереси шляхом підписування на підкасти, створення власних плейлистів, встановлення вподобань.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

За результатами виконання дипломної кваліфікаційної роботи бакалавра програмне забезпечення, що розробляється, повинно мати наступні функціональні характеристики:

- створення підкастів;
- перегляд власних підкастів;
- перегляд переліку епізодів власного підкасту;

- редагування підкастів;
- архівування підкастів;
- внесення епізодів підкастів з можливістю розділення на сезони;
- створення розшифровувань епізодів;
- редагування епізодів підкасту;
- підписування на підкаст;
- відписування від підкасту;
- перегляд підписаних на підкаст користувачів;
- вподобання епізодів підкастів;
- коментування епізодів підкастів;
- пошук підкастів;
- визначення тегів епізоду;
- визначення жанру підкасту;
- перегляд підкастів за жанрами;
- отримання переліку найпопулярніших підкастів за жанром;
- додавання епізоду до власного плейлиста;
- вилучення епізоду з власного плейлиста;
- перегляд переліку епізодів підкасту;
- прослуховування епізоду підкасту;
- авторизація та можливість неавторизованого доступу;
- визначення епізодів, доступних тільки для підписаних користувачів;
- перегляд історії прослуховувань епізодів користувачем;
- перегляд автором підкасту кількості прослуховувань підкасту загалом та кожного епізоду зокрема;
- перегляд переліку підкастів, на які підписався користувач;
- отримання переліку нових епізодів підписаних підкастів;
- реєстрація акаунтів.

A.3.2 Вимоги до надійності

Для забезпечення надійності має виконуватися валідація даних, що було введено користувачем в поля форм.

A.3.3 Умови експлуатації

Експлуатація програмного забезпечення має виконуватися на основі клієнт-серверної моделі. Для цього потрібно виділити окремий комп'ютер або скористатися виділеним сервером, який буде забезпечувати підтримку реалізації вебсервера. Вебсервер має бути доступним для запитів користувачів постійно.

A.3.4 Вимоги до складу та параметрів технічних засобів

Склад та параметри технічних засобів вебсервера було визначено для реалізації можливості доступу 300 користувачів одночасно. Відповідно вимоги можуть бути знижені, якщо потрібне забезпечення меншої кількості запитів, а для більшої кількості одночасних запитів вимоги навпаки мають бути збільшені.

Мінімальні системні вимоги до вебсервера включають:

- процесор: 4 ядра, тактова частота – 2,9 ГГц;
- оперативна пам'ять – 10 Гб;
- платформа – 32-розрядна або 64-розрядна;
- жорсткий диск – 1 Гб вільного простору.

A.3.5 Вимоги до транспортування та збереження

Безпосередньо програмне забезпечення має транспортуватися та зберігатися через репозиторій вебсервера. Збереження на окремих фізичних

носіях не передбачається.

А.3.6 Вимоги до програмної документації

Основою розробки даного програмного проєкту є технічне завдання на нього.

Опис прийнятих рішень виконується в пояснювальній записці. У розділі «Експлуатація та тестування програми» має бути представлено опис процесу роботи з програмою. Інших програмних документів не передбачається.

А.4 Порядок контролю та приймання

Календарний план завдання на дипломну кваліфікаційну роботу визначає набір етапів, за якими має забезпечуватися виконання, а також результати, які мають бути отримані за виконання відповідного етапу. Саме основі даних результатів має забезпечуватися приймання роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла models.py

```

from django.db import models
from django.shortcuts import reverse
from django.utils.text import slugify
from time import time

def gen_slug(s):
    res = slugify(s, allow_unicode=True)
    return res + '-' + str(int(time()))

class Podcast(models.Model):
    name = models.CharField(max_length=255, unique=True)
    photo = models.CharField(max_length=255, blank=True)
    description = models.TextField()
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    followercount = models.IntegerField()
    genres = models.ManyToManyField(Genre)
    date1 = models.DateTimeField()
    slug = models.SlugField(max_length=200, blank=True,
unique=True)

    def get_absolute_url(self):
        return reverse('podcast_present_url', kwargs={'slug':
self.slug})

    def get_update_url(self):
        return reverse('podcast_update_url', kwargs={'slug':
self.slug})

    def get_delete_url(self):
        return reverse('podcast_archive_url', kwargs={'slug':
self.slug})

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = gen_slug(self.name)
        super().save(*args, **kwargs)

    def __str__(self):
        return self.name

class Meta:
    ordering = ['-id']

```

```

class Tag(models.Model):
    name = models.SlugField(max_length=50, unique=True)

    def get_absolute_url(self):
        return reverse('tag_present_url', kwargs={'slug':
self.slug})

    def __str__(self):
        return '{}'.format(self.name)

    class Meta:
        ordering = ['-id']

class Genre(models.Model):
    name = models.SlugField(max_length=50, unique=True)

    def get_absolute_url(self):
        return reverse('genre_present_url', kwargs={'slug':
self.slug})

    def __str__(self):
        return '{}'.format(self.name)

    class Meta:
        ordering = ['-id']

class User(models.Model):
    login = models.CharField(max_length=50, unique=True)
    password = models.CharField(max_length=50)
    lastlisten = models.DateTimeField

    def __str__(self):
        return self.login

class Likes(models.Model):
    date1 = models.DateTimeField()
    userid = models.ForeignKey (User, on_delete=models.CASCADE)
    episodeid = models.ForeignKey (Episode,
on_delete=models.CASCADE)

class Comments(models.Model):
    date1 = models.DateTimeField()

```

```

        userid = models.ForeignKey (User, on_delete=models.CASCADE)
        episodeid = models.ForeignKey (Episode,
on_delete=models.CASCADE)
        commtext = models.TextField()

class Listen(models.Model):
    date1 = models.DateTimeField ()
    userid = models.ForeignKey (User, on_delete=models.CASCADE)
    episodeid = models.ForeignKey (Episode,
on_delete=models.CASCADE)

class Episode(models.Model):
    name = models.CharField(max_length=255, unique=True)
    description = models.TextField()
    textpresent = models.CharField (max_length=255,
unique=True)
    photo = models.CharField(max_length=255, blank=True)
    podcastid = models.ForeignKey (Podcast,
on_delete=models.CASCADE)
    audio = models.CharField(max_length=255, unique=True)
    tags = models.ManyToManyField(Genre)
    slug = models.SlugField(max_length=200, blank=True,
unique=True)
    likecount = models.IntegerField()
    listencount = models.IntegerField()
    mode = models.IntegerField()
    date1 = models.DateTimeField()

    def get_absolute_url(self):
        return reverse('episode_present_url', kwargs={'slug':
self.slug})

    def get_update_url(self):
        return reverse('episode_edit_url', kwargs={'slug':
self.slug})

    def __str__(self):
        return self.name

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = gen_slug(self.name)
        super().save(*args, **kwargs)

```

```

class Meta:
    ordering = ['-id']

class Follow(models.Model):
    date1 = models.DateField()
    userid = models.ForeignKey (User, on_delete=models.CASCADE)
    podcastid = models.ForeignKey (Podcast,
on_delete=models.CASCADE)

class Playlist(models.Model):
    userid = models.ForeignKey (User, on_delete=models.CASCADE)
    episodeid = models.ForeignKey (Episode,
on_delete=models.CASCADE)

```

Б.2 Текст файла views.py

```

from django.shortcuts import render, redirect
from django.shortcuts import get_object_or_404
from django.template import context
from django.urls import reverse
from django.views.generic import View
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic.edit import CreateView, UpdateView,
DeleteView
from django.views.generic.detail import DetailView
from django.views.generic import TemplateView
from django.core.paginator import Paginator
from django.db.models import Q
from datetime import date

from .models import Podcast, Tag, Genre, User, Likes, Comments,
Listen, Episode, Follow, Playlist
from .forms import PodcastForm, TagForm, EpisodeForm, GenreForm

class PodcastsView (ListView):
    model = Podcast
    template_name = 'index.html'

    def get_queryset(self):
        podcasts = super(PodcastsView, self).get_queryset()

```

```

        podcasts = podcasts.filter()
        for p in podcasts:
            if p.mode == 1:
                podcasts.remove(p)
        podcasts = podcasts.order_by("followercount")
        return podcasts

class HistoryView (LoginRequiredMixin, ListView):
    model = Listen
    template_name = 'history.html'

    def get_queryset(self):
        listens = super(HistoryView, self).get_queryset()
        auser = self.request.user
        listens = listens.filter(Q(userid=auser))
        listens = listens.order_by("-date1")
        return listens

def genre_podcasts(request):
    genr = Genre.objects.filter()
    genrename = request.GET.get('genre', '')
    podcasts =
Podcast.objects.filter(Q(genres__name__contains=genrename))

    for p in podcasts:
        if p.mode == 1:
            podcasts.remove(p)

    pop = podcasts.order_by('followercount')
    pop = pop [:9]

    pin = Paginator(podcasts, 9)

    cp = request.GET.get('page', 1)
    res = pin.get_page(cp)

    paginatorflag = res.has_other_pages()
    if res.has_previous():
        before = '?page={}'.format(res.previous_page_number())
    else:
        before = ''

    if res.has_next():

```

```

        after = '?page={}'.format(res.next_page_number())
    else:
        after = ''

```

```

context = {
    'genres': genr,
    'popular': pop,
    'page_object': res,
    'is_paginated': paginatorflag,
    'prev_url': before,
    'next_url': after
}

```

```

return context

```

```

def followed_podcasts(request):
    auser = request.user
    follows = Follow.objects.filter(Q(userid=auser))
    podcasts = []
    for f in follows:
        podcasts.append(Podcast.objects.get(id=f.podcastid.id))

    for p in podcasts:
        if p.mode == 1:
            podcasts.remove(p)

    pin = Paginator(podcasts, 9)

    cp = request.GET.get('page', 1)
    res = pin.get_page(cp)

    paginatorflag = res.has_other_pages()

    if res.has_previous():
        before = '?page={}'.format(res.previous_page_number())
    else:
        before = ''

    if res.has_next():
        after = '?page={}'.format(res.next_page_number())
    else:
        after = ''

    context = {

```

```

        'page_object': res,
        'is_paginated': paginator.flag,
        'prev_url': before,
        'next_url': after
    }

    return render(request, 'followed_podcasts.html',
context=context)

```

```

class OwnPodcastsView (LoginRequiredMixin, ListView):
    model = Podcast
    template_name = 'user_podcasts.html'

    def get_queryset(self):
        podcasts = super(OwnPodcastsView, self).get_queryset()
        podcasts = podcasts.filter(user = self.request.user)
        podcasts = podcasts.order_by("-id")
        return podcasts

class PodcastEpisodesView (ListView):
    model = Episode

    def get_queryset(self):
        episodes = super(PodcastEpisodesView,
self).get_queryset()
        f = Follow.objects.get(userid=self.request.user,
podcastid=self.request.POST['id'])
        if f:
            episodes = episodes.filter(podcastid =
self.request.POST['id'])
        else:
            episodes = episodes.filter(podcastid =
self.request.POST['id'])
        for ep in episodes:
            if ep.mode == 1:
                episodes.remove(ep)
        episodes = episodes.order_by("-id")
        return episodes

class OwnPodcastEpisodesView (LoginRequiredMixin, ListView):
    model = Episode
    template_name = 'user_podcast_episodes.html'

```

```

    def get_queryset(self):
        f = Podcast.objects.get(user=self.request.user,
id=self.request.POST['id'])
        episodes = []
        if f:
            episodes = super(OwnPodcastEpisodesView,
self).get_queryset()
            episodes = episodes.filter(podcastid =
self.request.POST['id'])
        return episodes

```

```

def followed_episodes(request):
    auser = request.user
    follows = Follow.objects.filter(Q(userid=auser))
    episodes = []
    for f in follows:
        sa = Episode.objects.filter(Q(podcastid=f.podcastid))
        sa.filter(Q(date1__gte=auser.lastlisten))
        for seas in sa:
            episodes.append(seas)
    episodes.order_by('-date1')

    pin = Paginator(episodes, 5)

    cp = request.GET.get('page', 1)
    res = pin.get_page(cp)

    paginatorflag = res.has_other_pages()

    if res.has_previous():
        before = '?page={}'.format(res.previous_page_number())
    else:
        before = ''

    if res.has_next():
        after = '?page={}'.format(res.next_page_number())
    else:
        after = ''

    context = {
        'page_object': res,
        'is_paginated': paginatorflag,
        'prev_url': before,
        'next_url': after

```

```

    }

    return render(request, 'followed_episodes.html',
context=context)

def search_podcasts(request):
    podcastsearch = request.GET.get('search', '')
    podcasts =
Follow.objects.filter(Q(name__contains=podcastsearch))

    for p in podcasts:
        if p.mode == 1:
            podcasts.remove(p)

    pin = Paginator(podcasts, 18)

    cp = request.GET.get('page', 1)
    res = pin.get_page(cp)

    paginatorflag = res.has_other_pages()

    if res.has_previous():
        before = '?page={}'.format(res.previous_page_number())
    else:
        before = ''

    if res.has_next():
        after = '?page={}'.format(res.next_page_number())
    else:
        after = ''

    context = {
        'page_object': res,
        'is_paginated': paginatorflag,
        'prev_url': before,
        'next_url': after
    }

    return context

def tag_episodes(request):
    tagname = request.GET.get('tag', '')

```

```

tagpodcasts =
Episode.objects.filter(Q(tags__contains=tagname))

pin = Paginator(tagpodcasts, 18)

cp = request.GET.get('page', 1)
res = pin.get_page(cp)
paginatorflag = res.has_other_pages()

if res.has_previous():
    before = '?page={}'.format(res.previous_page_number())
else:
    before = ''

if res.has_next():
    after = '?page={}'.format(res.next_page_number())
else:
    after = ''

context = {
    'page_object': res,
    'is_paginated': paginatorflag,
    'prev_url': before,
    'next_url': after
}

return render(request, 'tag_episodes.html', context=context)

def episode_detail(request, slug):
    template_name = 'episode.html'
    epis = get_object_or_404(Episode, slug=slug)
    comments = Comments.objects.filter(episodeid=epis.id)
    com = None
    es = Episode.objects.get(podcastid=epis.podcastid)
    if request.method == 'POST':
        cf = CommentForm(data=request.POST)
        if cf.is_valid():
            com = cf.save()
            com.episodeid = epis
            com.userid = request.user
            com.date1 = date.today()
            com.save()
        else:
            cf = CommentForm()

```

```

        return render(request, template_name, {'episode': epis,
                                                'comments': comments,
                                                'add_comment': com,
                                                'comment_form': cf,
                                                'comments': comments,
                                                'episodes': es
                                                })

def like_episode(request):
    episode = get_object_or_404(Episode,
id=request.POST.get('episodeid'))
    likes = Likes.objects.filter(Q(episodeid=episode.id))
    likeflag = False
    for like in likes:
        if like.userid == request.user:
            likeflag = True
            likes.remove(like)
            episode.likecount = episode.likecount - 1
            episode.save()
    if not likeflag:
        like = Likes()
        like.userid = request.user
        like.episodeid = request.POST.get('episodeid')
        like.date1 = date.today()
        likes.append()
        likes.save()
        episode.likecount = episode.likecount + 1
    return HttpResponseRedirect(episode.get_absolute_url())

def listen_episode(request):
    episode = get_object_or_404(Episode,
id=request.POST.get('episodeid'))
    listens = Listen.objects.filter(Q(episodeid=episode.id))
    listenflag = False
    for lis in listens:
        if lis.userid == request.user:
            listenflag = True
    if not listenflag:
        lis = Listen()
        lis.userid = request.user
        lis.episodeid = request.POST.get('episodeid')
        lis.date1 = date.today()

```

```

        lis.append()
        lis.save()
        episode.listencount = episode.listencount + 1
    return HttpResponseRedirect(episode.get_absolute_url())

def follow_podcast(request):
    podcast = get_object_or_404(Podcast,
id=request.POST.get('podcastid'))
    follows = Follow.objects.filter(Q(podcastid=podcast.id))
    follflag = False
    for f in follows:
        if f.userid == request.user:
            follflag = True
            follows.remove(f)
            follows.save()
            podcast.followercount = podcast.followercount - 1
            podcast.save()
    if not follflag:
        f = Follow()
        f.userid = request.user
        f.podcastid = request.POST.get('podcastid')
        f.date1 = date.today()
        follows.append()
        follows.save()
        podcast.followercount = podcast.followercount + 1
    return HttpResponseRedirect(podcast.get_absolute_url())

class RegisterView (CreateView):
    model = User
    fields = ['login', 'password']
    template_name = 'register.html'

    def get_form(self):
        cpf = super(RegisterView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'
        return cpf

    def post(self, request, *args, **kwargs):
        cpf = self.get_form (request.POST)
        if cpf.is_valid():
            res = cpf.save()

```

```

        cpf.lastlisten = date.today()
        res.save()
        user = authenticate(cpf.login, cpf.password)
        if user is not None:
            login(request, user)
            return
    http.HttpResponseRedirect('podcasts_view')
    else:
        return
    http.HttpResponseRedirect('register.html')
    return redirect('podcasts-view', pk=cpf.pk)
return render(request, 'register', {'form': form})

```

```

class CreatePodcastView (LoginRequiredMixin, CreateView):
    model = Podcast
    fields = ['name', 'photo', 'description']
    template_name = 'podcast_create.html'

    def get_form(self):
        cpf = super(CreatePodcastView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'
        return cpf

    def get_context_data(self, **kwargs):
        context = super(CreatePodcastView,
self).get_context_data(**kwargs)
        context['genres'] = Genre.objects.all()
        context['action'] = reverse('create-podcast')
        return context

    def post(self, request, *args, **kwargs):
        cpf = self.get_form (request.POST)
        if cpf.is_valid():
            res = cpf.save()
            res.user = self.request.user
            res.save()
            return redirect('user_podcast_episodes', pk=cpf.pk)
        return render(request, 'podcast_create', {'form': form})

```

```

class PlaylistView (LoginRequiredMixin, DetailView):
    model = Playlist
    template_name = 'playlist.html'

```

```

class EditPodcastView (LoginRequiredMixin, UpdateView):
    model = Podcast
    fields = ['name', 'photo', 'description']
    template_name = 'podcast_edit'

    def get_form(self):
        cpf = super(EditPodcastView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'
        return cpf

    def get_context_data(self, **kwargs):
        context = super(EditPodcastView,
self).get_context_data(**kwargs)
        context['genres'] = Genre.objects.all()
        context['action'] = reverse('podcast_edit')
        return context

    def post(self, request, *args, **kwargs):
        cpf = get_object_or_404(Podcast, pk=request.id)
        if cpf.user != request.user:
            return redirect('error', rule='no-edit-rights')
        if cpf.is_valid():
            res = cpf.save()
            res.save()
            return redirect('user_podcast_episodes', pk=cpf.pk)
        return render(request, 'podcast_edit', {'form': form})

class LoginView(TemplateView):
    template_name = 'login.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        return context

    def post(self, request):
        ulogin = request.POST.get('login')
        upass = request.POST.get('password')
        user = authenticate(ulogin, upass)
        if user is not None:
            login(request, user)
            return http.HttpResponseRedirect('podcasts_view')
        else:
            return http.HttpResponseRedirect('login.html')

```

```

class LogoutView(RedirectView):
    url = '/'

    def get_redirect_url(self):
        logout(self.request)
        return super().get_redirect_url()

class AddTagView (LoginRequiredMixin, CreateView):
    model = Tag
    fields = ['name']
    template_name = 'add_tag.html'

    def get_form(self):
        cpf = super(CreatePodcastView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'
        return cpf

    def post(self, request, *args, **kwargs):
        cpf = self.get_form (request.POST)
        if cpf.is_valid():
            cpf.save()
            return redirect('tag_episodes', pk=cpf.pk)
        return render(request, 'add_tag', {'form': form})

class CreateEpisodeView (LoginRequiredMixin, CreateView):
    model = Episode
    fields = ['name', 'description', 'textpresent', 'audio',
'tags', 'mode']
    template_name = 'episode_create.html'

    def get_form(self):
        cpf = super(CreateEpisodeView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'
        return cpf

    def get_context_data(self, **kwargs):
        context = super(CreateEpisodeView,
self).get_context_data(**kwargs)
        context['tags'] = Tag.objects.all()
        context['action'] = reverse('create-episode')
        return context

```

```

def post(self, request, *args, **kwargs):
    cpf = self.get_form (request.POST)
    if cpf.is_valid():
        res = cpf.save()
        res.podcastid      =      get_object_or_404(Episode,
pk=request.id)
        res.save()
        return redirect('user_podcast_episodes', pk=cpf.pk)
    return render(request, 'episode_create', {'form': form})

class EditEpisodeView (LoginRequiredMixin, UpdateView):
    model = Episode
    fields = ['name', 'description', 'textpresent', 'audio',
'tags', 'mode']
    template_name = 'episode_edit.html'

    def get_form(self):
        cpf = super(EditEpisodeView, self).get_form()
        for f in form.visible_fields():
            f.field.widget.attrs['class'] = 'form-control'

        return cpf

    def get_context_data(self, **kwargs):
        context      =      super(EditEpisodeView,
self).get_context_data(**kwargs)
        context['tags'] = Tag.objects.all()
        context['action'] = reverse('episode_edit')

        return context

    def post(self, request, *args, **kwargs):
        cpf = get_object_or_404(Episode, pk=request.id)
        if cpf.is_valid():
            res = cpf.save()
            res.save()
            return redirect('episode_view', pk= cpf.pk)

        return render(request, 'episode_edit', {'form': form})

    def to_playlist(request):
        episode = get_object_or_404(Episode,
id=request.POST.get('episodeid'))

```

```

pl = Playlist.objects.filter(Q(userid=request.user))
flag = False
for p in pl:
    if p.episodeid == episode:
        flag = True
if not flag:
    p = Playlist()
    p.userid = request.user
    p.episodeid = request.POST.get('episodeid')
    pl.append(p)
    pl.save()

return HttpResponseRedirect(playlist.get_absolute_url())

def archive_podcast(request):
    podcast = get_object_or_404(Podcast,
id=request.POST.get('podcastid'))
    podcast.mode = 1
    podcast.save()
    return redirect('index.html')

```

ДОДАТОК В
Слайди презентації

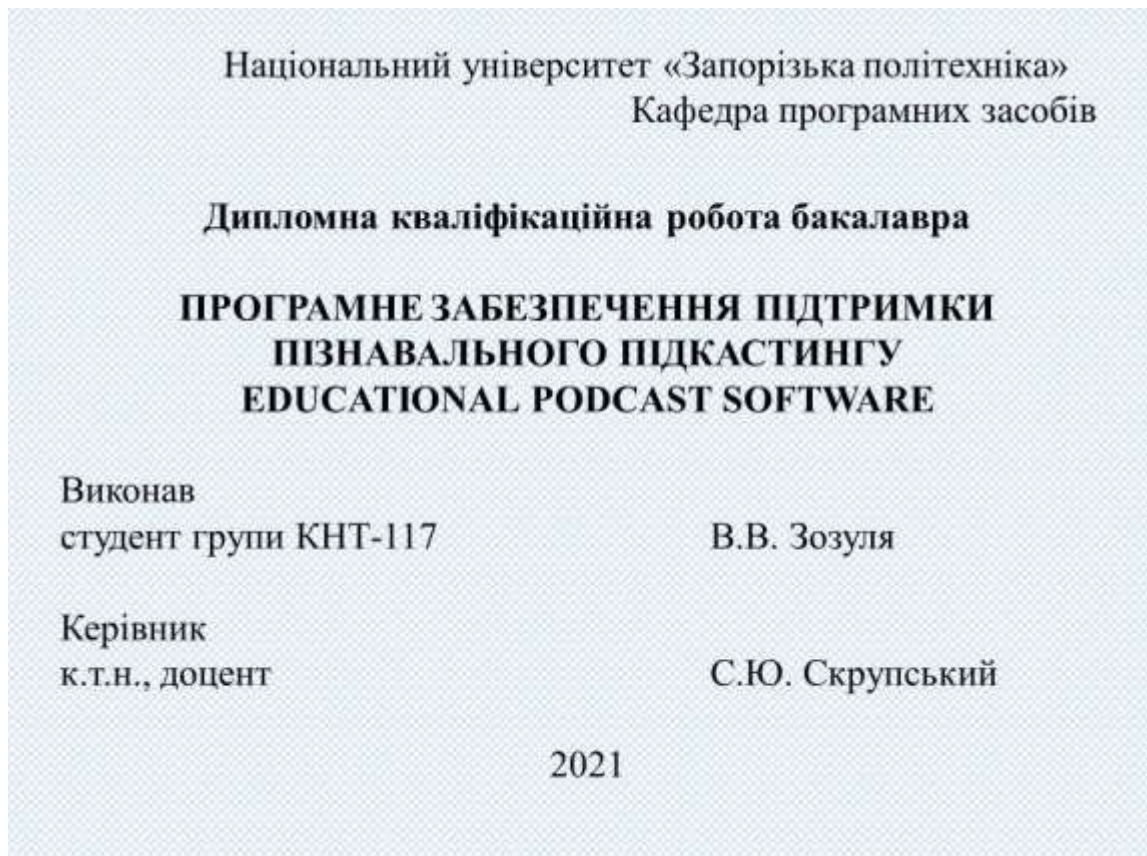


Рисунок В.1 – Слайд № 1

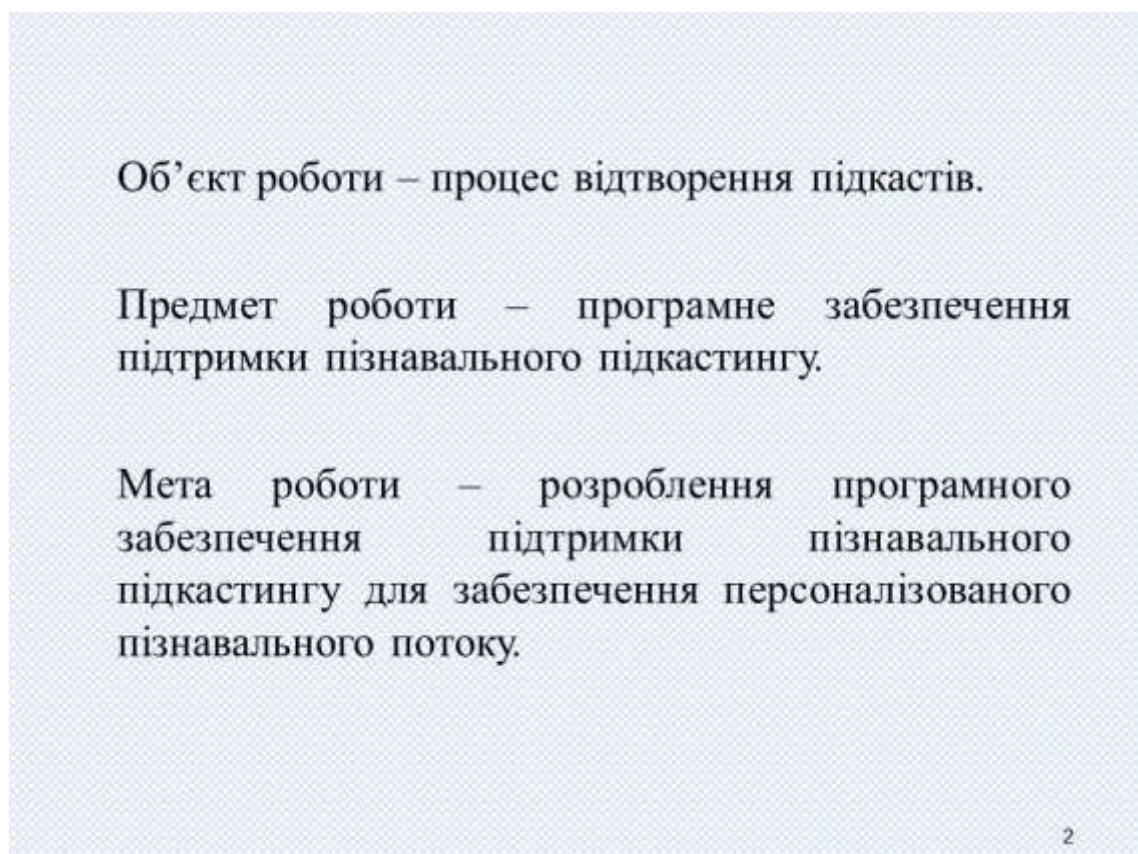


Рисунок В.2 – Слайд № 2

Завдання

- ✓ виділення вимог до програмного забезпечення підтримки пізнавального підкастингу;
- ✓ вибір мови і засобів проєктування і розробки;
- ✓ проєктування програмного забезпечення;
- ✓ реалізація і налагодження програмного забезпечення.

3

Рисунок В.3 – Слайд № 3

Обґрунтування вибору мови програмування

Критерій порівняння	Python	Java
Лаконічність коду	Вища	Нижча
Швидкість розробки	Вища	Нижча
Складність підтримки	Нижча	Вища
Швидкість запуску	Вища	Нижча
Швидкість виконання	Нижча	Вища
Типізація	Динамічна	Статична
Запуск під різними операційними системами	Так	Так
Веброзробка	Так	Так
Доступність фреймворків розробки	Так	Так

4

Рисунок В.4 – Слайд № 4

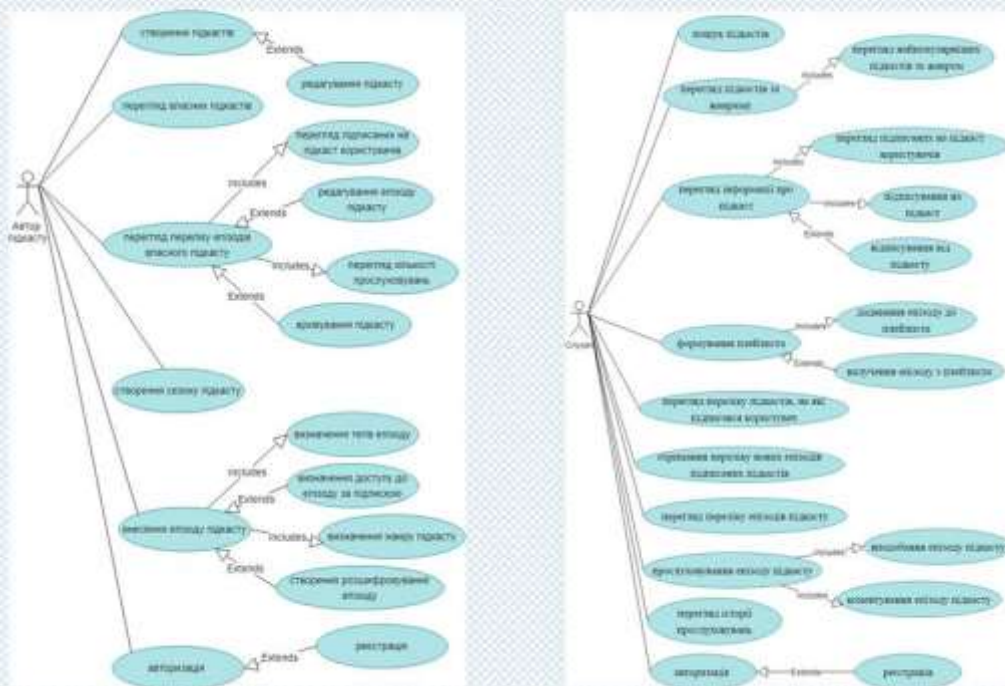
Діаграма прецедентів неавторизованого користувача



5

Рисунок В.5 – Слайд № 5

Діаграми прецедентів авторів підкастів і слухачів



6

Рисунок В.6 – Слайд № 6



Рисунок В.7 – Слайд № 7

Взаємодія користувача з вебзастосунком, побудованим на основі використання вебфреймворку Django

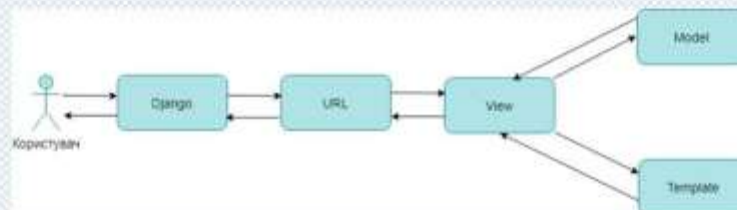


Рисунок В.8 – Слайд № 8



Рисунок В.9 – Слайд № 9

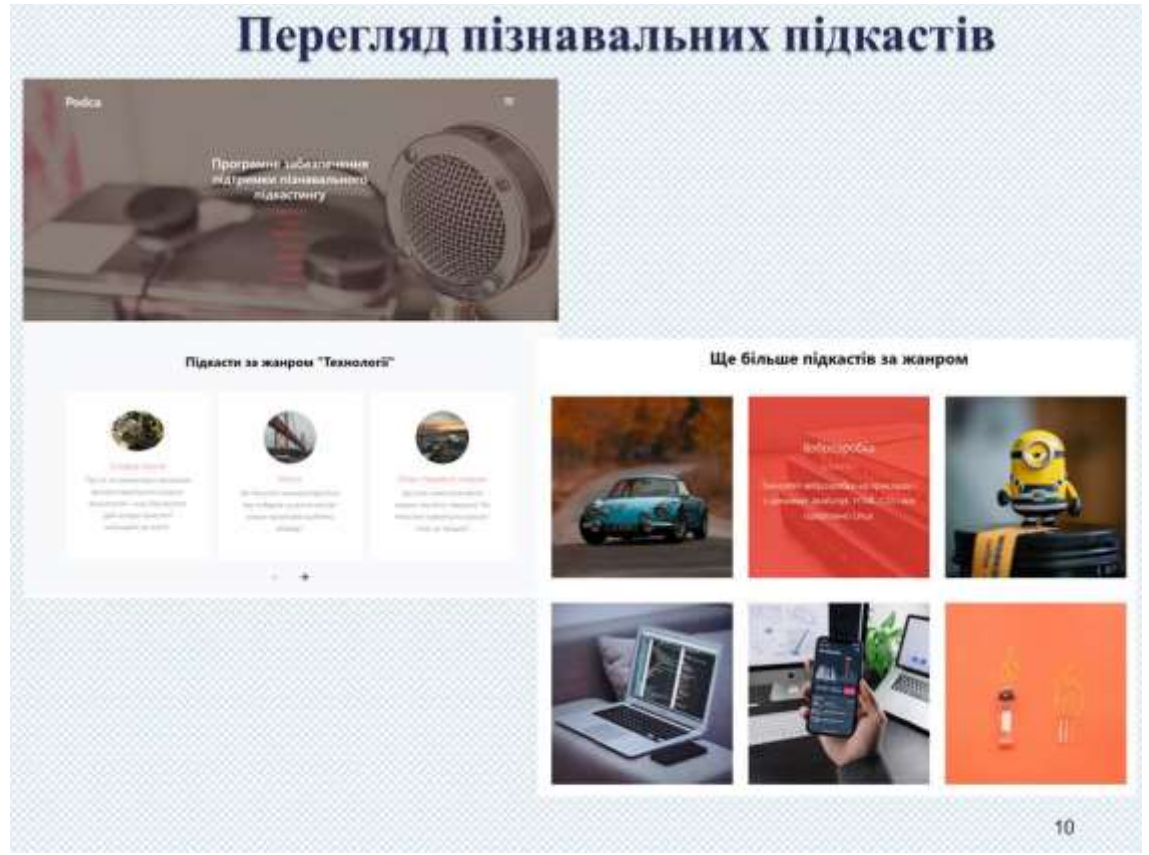


Рисунок В.10 – Слайд № 10

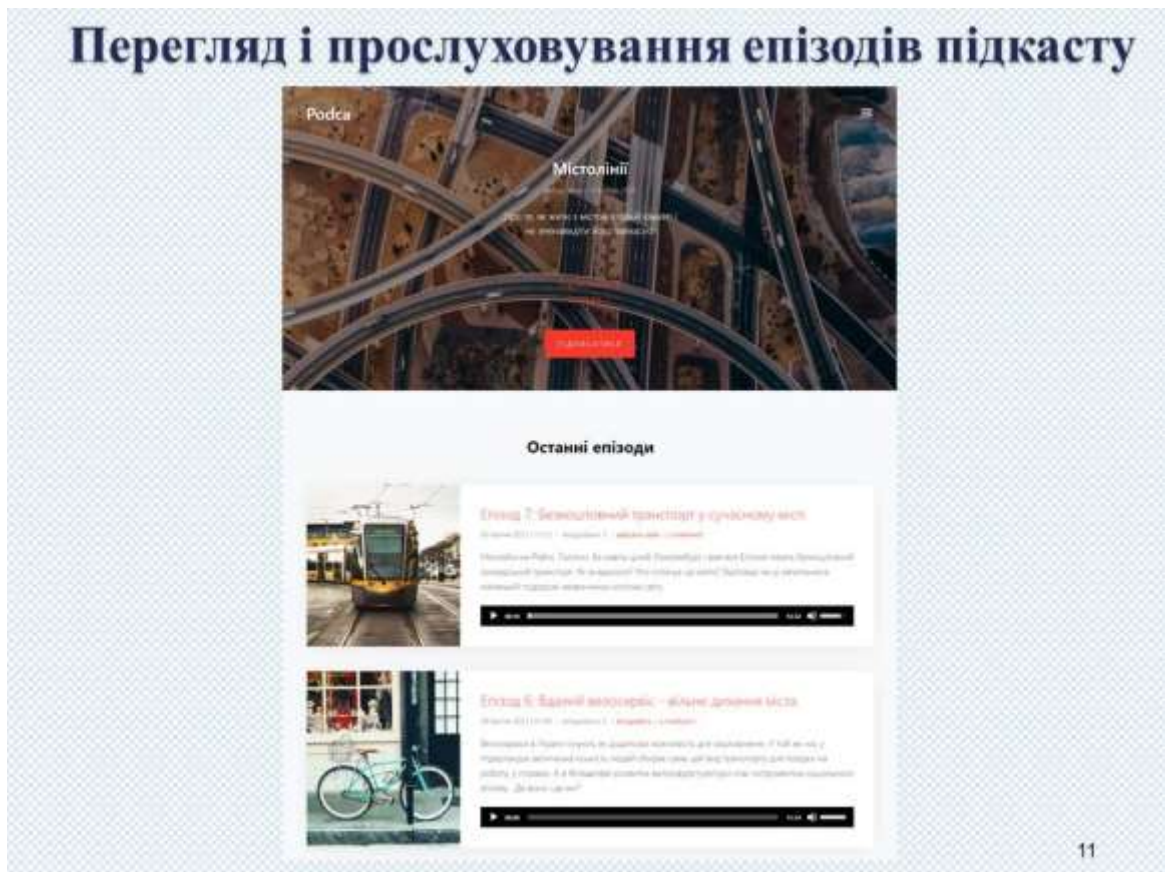


Рисунок В.11 – Слайд № 11

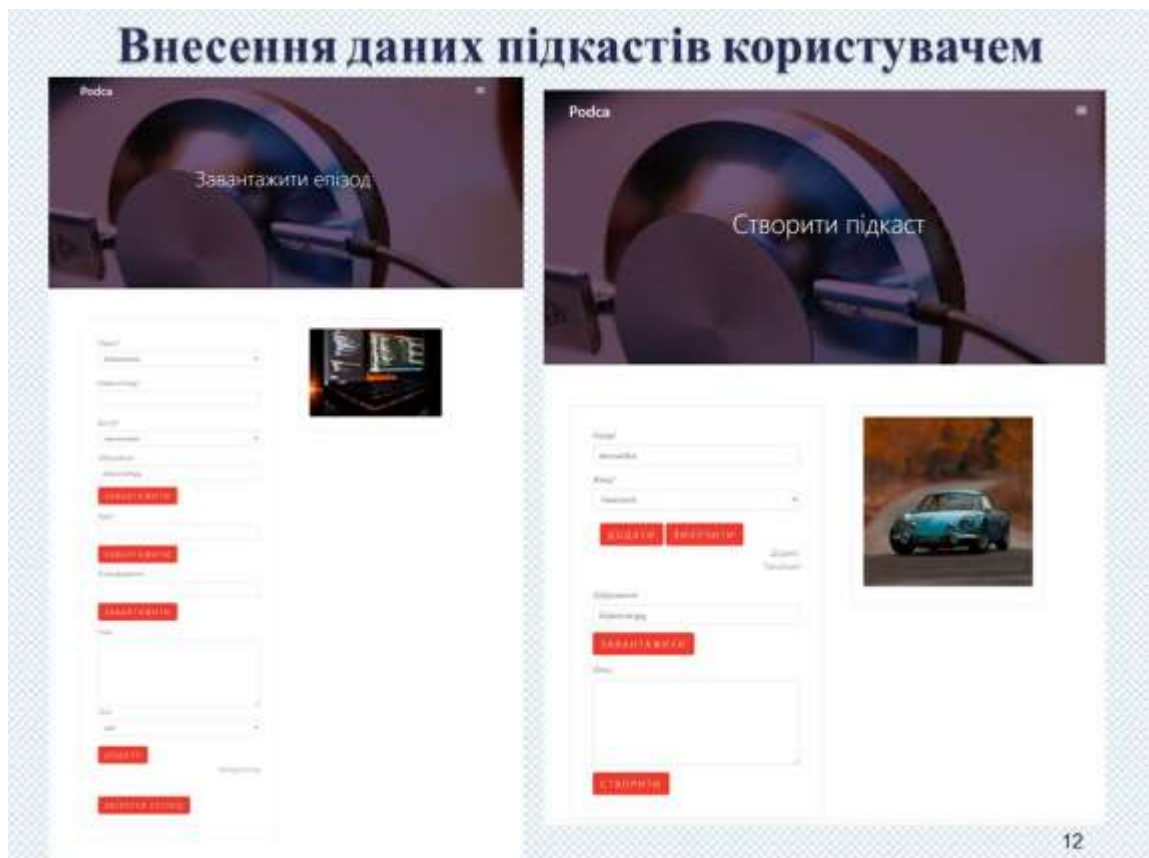


Рисунок В.12 – Слайд № 12

Висновки

У роботі проаналізовано концепцію підкастингу, виділено основні особливості підкастів, причини сучасної популярності підкастів та принципи їх організації. Визначено пізнавальні підкасти в якості основного призначення розроблюваного програмного забезпечення. Розглянуто основні програмні засоби хостингу підкастів та проаналізовано їх функціональні можливості та принципи організації.

Програмне забезпечення спроектовано та реалізовано у вигляді вебзастосунку. Обрано мову програмування Python для реалізації програмного забезпечення. Визначено функціональні вимоги до програми, побудовано діграми прецедентів неавторизованого користувача, автора підкасту та слухача. Визначено структуру бази даних, обрано MySQL в якості системи управління базою даних, обрано вебфреймворк Django для створення програмного забезпечення.

Усі завдання, які було визначено на дипломну кваліфікаційну роботу, повністю виконано.

13

Рисунок В.13 – Слайд № 13