MALNAD COLEGE OF ENGINEERING

Unde rthe auspices of M.T.E.S

(Anautonomous institution under Visvesvaraya Technological University,Belgaum)
Hassan – 573201, Karnataka, India

Report on Course Title :Full Stack Development

"Zyra Music Player"

Submitted by team number : 02

Name	USN
Ajay Gowda G P	4MC23IS003
Bhuvan J H	4MC23IS019
Darshan R G	4MC23IS027
Kishan C M	4MC23IS055

Mr. Krishna Swaroop A

(Assistant Professor Dept.of ISE)

Department of Information Science &Engineering

Malnad College of Engineering

Hassan–573202

2025-26

INDEX

1.Abstract:

Zyra is a Django-based music streaming platform designed to help users discover, play, and organize songs while offering artists a simple way to upload and publish their tracks. The system provides a smooth interface for listening to music, browsing genres, creating playlists, and interacting through likes and comments. Artists can upload songs, add metadata, and engage with listeners, whereas the admin manages moderation and platform security.

The project demonstrates core full-stack development concepts including secure authentication, media handling, backend processing, database management, and cloud storage integration. Zyra successfully delivers a prototype of a modern music streaming service with a clean interface and scalable architecture.

2.Introduction:

Background of the problem

Music streaming has become one of the most widely used digital services, but many platforms primarily focus on mainstream artists and commercial content. As a result, independent and emerging artists face challenges in publishing their music due to complicated upload workflows, strict content policies, or costly distribution requirements. This reduces their chances of visibility and makes it harder for listeners to discover unique or niche music genres.

On the listener side, modern music apps often rely heavily on algorithm-driven recommendations that may not accurately represent a user's actual preferences. Discovering fresh music becomes time-consuming because interfaces are crowded, search options are limited, and lesser-known artists are overshadowed.

Why the Domain Was Chosen?

The music streaming domain provides an excellent real-world environment to apply full-stack development concepts. It involves complex backend operations such as media storage, metadata handling, search filters, user management, and content delivery—all of which are ideal for learning Django. Additionally, building such a system encourages understanding of modern web application architecture and cloud integration.

From a user interface perspective, music apps demand clean navigation, fast loading, and visually appealing layouts. This offers opportunities to practice frontend development skills, including responsive design, user experience optimization, and dynamic content loading.

Real-World Scenario

In a real-world situation, an independent artist might record a song and look for a platform to upload it without technical complexity or high fees. They expect an easy interface where they can publish music, add album details, track performance, and engage with listeners. Existing platforms often fail to offer such simplicity, especially for newcomers.

On the listener's side, a music enthusiast wants to browse new songs, follow artists they like, and organize playlists effortlessly. They prefer a platform that provides personalized discovery without overwhelming advertisements or cluttered designs. Zyra addresses both needs by offering an intuitive system where artists and users can interact with minimal friction..

Why the domain was chosen

The music streaming domain combines backend development, media processing, user engagement features, and UI/UX challenges, making it ideal for a full-stack project using Django.

Real-world scenario

A new artist wants to upload songs and reach listeners. A user wants to explore music, create playlists, and follow artists. Zyra connects both through a simple and accessible platform.

Issues in existing systems

Complex upload processes for artists

Limited visibility for new or independent artists

Weak playlist and personalization features

Poor content moderation

Cluttered and confusing user interfaces

Dependence on external distributors for uploads

How the proposed solution helps?

Zyra simplifies the process for artists by providing a direct and user-friendly upload interface. Artists can add metadata, manage albums, and publish tracks instantly, without external tools or complex verification. This increases their visibility and empowers them to connect with listeners who appreciate diverse music styles

3.Objectives of the Project

To build an online platform for streaming and managing music content, enabling users to browse songs, explore artists, listen to tracks, organize playlists, and interact socially through likes and comments. The platform will act as a centralized digital system connecting listeners, artists, and administrators while ensuring seamless music playback, smooth navigation, and structured data handling.

To provide users with intuitive tools for song discovery, including search by genre, artist, mood, or popularity, playlist creation, and personalized browsing options. The platform aims to enhance user experience by offering clarity, convenience, and an enjoyable interface for listening to music. Zyra allows users to interact with tracks and artists through follows, likes, comments, and playlist sharing, fostering active engagement and meaningful digital connections.

To digitize and automate user–artist interactions, replacing manual or external communication with built-in features such as comments, notifications, and engagement analytics. This digital transformation ensures faster communication, reduces dependency on external platforms, and makes it easier for artists to understand listener preferences. It enables streamlined interactions, efficient feedback processes, and improved visibility for rising artists.

4.System Requirements

4.1 Software Requirements

The Zyra platform runs on Python 3.10+ with Django 4.2 LTS, providing a stable and modern backend framework for web development. It supports PostgreSQL or SQLite as the database, allowing flexible data management depending on the scale of deployment.

For the frontend, Zyra uses HTML, CSS, and JavaScript, enabling interactive and responsive user interfaces. Additional libraries and frameworks such as Pillow (for image handling), Mutagen (for audio metadata), and Django REST Framework (for API management) are required to enhance functionality and performance.

Development can be done using VS Code or PyCharm, both of which offer rich coding support and debugging tools. For optional cloud storage, AWS S3 can be integrated to handle large volumes of music files efficiently, providing scalability for growing platforms.

4.2 Hardware Requirements:

The minimum hardware for Zyra includes a dual-core processor and 4GB RAM, ensuring the platform can run on modest systems without performance issues. Basic storage is required to hold the application, media files, and database efficiently.

For optimal performance, a quad-core processor with 8GB RAM is recommended. This configuration allows smooth handling of multiple concurrent users, faster processing of uploads, and efficient content streaming. A storage capacity of 250GB is advised to accommodate extensive music libraries.

A stable internet connection is crucial for both artists and users. It ensures seamless uploading, streaming, and interaction on the platform, maintaining a reliable and uninterrupted user experience across the ecosystem.

5. System Design

5.1Architecture Overview:

The Zyra platform is built using a layered architecture that separates user interface, application logic, and data storage for better organization and scalability. The User Interface (UI) provides users with song listings, playlists, artist pages, and upload forms, offering an intuitive and interactive experience.

The Django application forms the core of the platform, handling tasks such as user authentication, music uploads, streaming logic, and playlist creation. It ensures that requests from users are processed efficiently and securely.

The Database stores essential information including user profiles, songs, playlists, comments, and artist data, while File Storage (either local or cloud-based like AWS S3) keeps audio files separate for efficient retrieval and management. The Admin Panel oversees content moderation, user management, and overall platform stability.:

When a user sends a request, Django processes it by interacting with the database or file storage as needed. Song uploads are stored in file storage, and metadata such as title, artist, and album information is saved in the database. User requests follow the Models → Views → Templates → Browser flow, with Django ORM managing create, read, update, and delete operations to ensure smooth interaction between the application and the database.

5.2 Explanation of architecture

- **Explanation of each block:**

1.Presentation Layer :

This layer is responsible for everything the user interacts with directly. It displays songs, playlists, artist pages, and upload forms. Users can search for tracks, play music, create and manage playlists, and interact with social features like likes and comments. This layer ensures a responsive and intuitive experience across web and mobile platforms**.**

2. Application Layer (Django Application):

The application layer handles the business logic of the platform. It processes user requests such as authentication, music uploads, streaming, and playlist creation. Django acts as the intermediary between the presentation layer and data layer, validating requests, enforcing permissions, and coordinating data retrieval and storage.

3. Data Layer (Database and File Storage):

The data layer stores all the structured and unstructured data required by the platform. The Database manages users, songs, playlists, comments, and artist metadata, while File Storage (local or AWS S3) holds the actual audio files. This separation ensures efficient access to large media files without overloading the database**.**

4. Admin Panel:

Although not strictly a separate layer, the admin panel interacts with all layers to manage content, moderate uploads, oversee user activity, and ensure system stability. It allows administrators to maintain a clean, secure, and organized platform.

- **Data flow explanation:**

1.User Request Flow

- Users access the Zyra platform via the web or mobile UI.
- Users can search for songs, play tracks, create playlists, or upload music
- Each interaction generates a request sent from the UI to the Django backend.
- The request is captured by Django's URL routing system.
- The request is mapped to the corresponding view function or API endpoint.
- Input validation occurs to ensure data is correct and secure.
- User session and authentication details are verified if required.
- The request is forwarded to the relevant business logic for processing..

2. Request Processing in Django

- Django receives the request and identifies the required operation.
- Business logic handles tasks such as uploading files, streaming music, or updating playlists.
- Django validates user permissions and roles for restricted actions.
- For uploads, Django processes the file and prepares it for storage.
- For streaming, Django retrieves the file path and metadata.

3.Business Logic Layer

- Django ensures that API responses follow the expected format (JSON or HTML).
- Errors or exceptions are caught and handled gracefully.
- Once processing is complete, Django forwards the necessary data to the database or file storage.

4.Data Access Layer

- Interaction with Database and File Storage
- Django ORM manages communication with the relational database.
- CRUD operations (Create, Read, Update, Delete) are performed for songs, playlists, and user data.
- Metadata such as title, artist, and album is stored in the database.
- Comments and likes are recorded in real-time.
- File storage (local or AWS S3) stores the actual audio files.
- Files are indexed and linked with their metadata in the database.

5.output/response Layer

- Processed data is sent from Django back to the UI.
- The UI renders the response, showing songs, playlists, or notifications.
- For uploads, the user receives a confirmation message.
- For streaming, audio is played through the UI player.
- Search results are displayed with relevant filters and sorting.
- User interactions like likes or comments are updated in real-time.
- Errors or validation messages are shown clearly to guide the user.

6. Implementation

Models ovreview

The models in Zyra define the structure of the database and represent the core entities of the platform. This includes models for users, songs, playlists, comments, and artist profiles. Each model contains fields and relationships that capture essential information, such as song metadata, playlist details, and user interactions. By organizing data into well-structured models, Zyra ensures consistency, easy querying, and smooth integration with the rest of the application.

Models also enable the Django ORM to perform database operations efficiently, handling tasks like creating, reading, updating, and deleting records. Relationships between models, such as many-to-many for playlists and songs or one-to-many for artist uploads, allow complex queries to be executed seamlessly. This structured approach provides a solid foundation for the platform’s functionality and scalability.

URL Routing Overview

The URL routing system in Django maps incoming user requests to the appropriate view functions or API endpoints. When a user searches for a song, plays music, or uploads a track, the URL routing ensures the request reaches the correct handler. This makes navigation smooth and allows the application to respond accurately to user actions.

URL routing also supports dynamic routing, where URLs can include parameters like song IDs, playlist IDs, or artist names. This enables the platform to generate personalized content dynamically, such as showing a user’s playlists or an artist’s uploaded tracks. Overall, URL routing is crucial for connecting the user interface with the backend logic effectively.

Important Functionality

Zyra includes several important functionalities that define the user experience. Key features include music uploads, streaming, playlist creation and management, search, and social interactions like likes and comments. These functionalities ensure that both artists and listeners can interact with the platform efficiently.

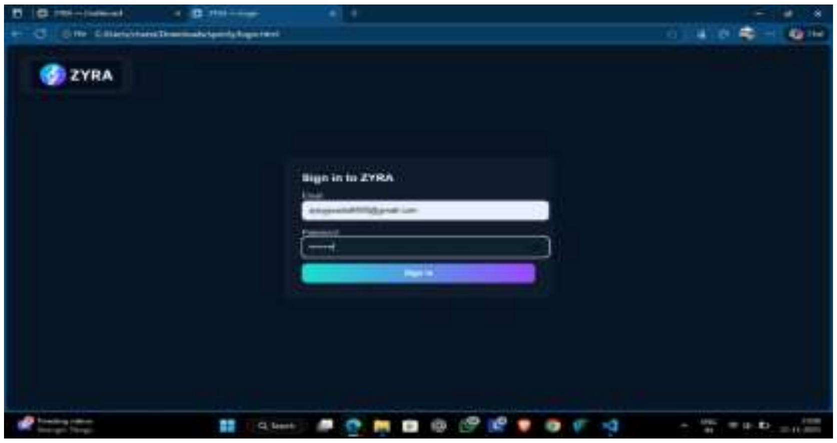
Each feature is integrated seamlessly with the models and URL routing. For example, when a user creates a playlist, the system updates the database and reflects the changes in the UI instantly. Similarly, streaming a song retrieves the file from storage while displaying metadata, providing a smooth and interactive experience.

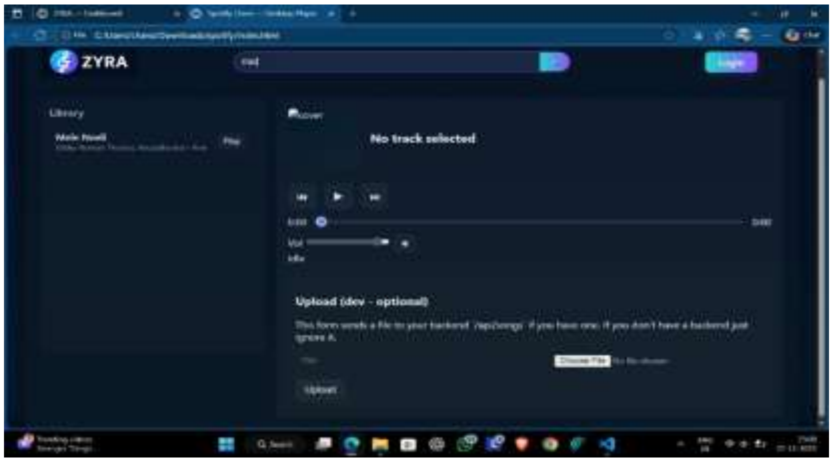
Special Logic

The platform incorporates special logic to handle complex tasks and ensure reliability. This includes validating uploaded files, linking audio files with metadata, enforcing user permissions, and optimizing data retrieval for performance. Such logic guarantees that only valid and authorized actions are performed, maintaining platform integrity.

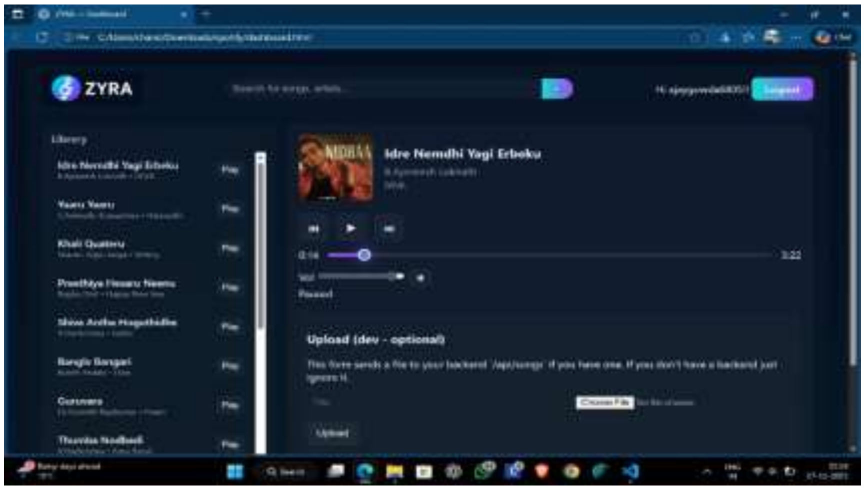
Special logic also enhances user experience by ensuring quick access to songs and playlists, managing concurrent requests efficiently, and handling exceptions gracefully. By implementing these intelligent mechanisms, Zyra delivers a robust and seamless environment for both artists and listeners.

8.Screenshots





This form sends a file to your backend /api/songs if you have one. If you don't have a backend just ignore it.



ZYRA

Search for songs, artists...

Hi, [illegible] Logout

Library

Idre Nemdhi Yagi Erbeku	Play
Yaaru Yaaru	Play
Khali Quateru	Play
Preethiya Hesaru Neenu	Play
Shiva Antha Hoguthidhe	Play
Banglu Bangari	Play
Gurunera	Play
Thumba Nodbedi	Play

Idre Nemdhi Yagi Erbeku

0:14 3:22

Vol

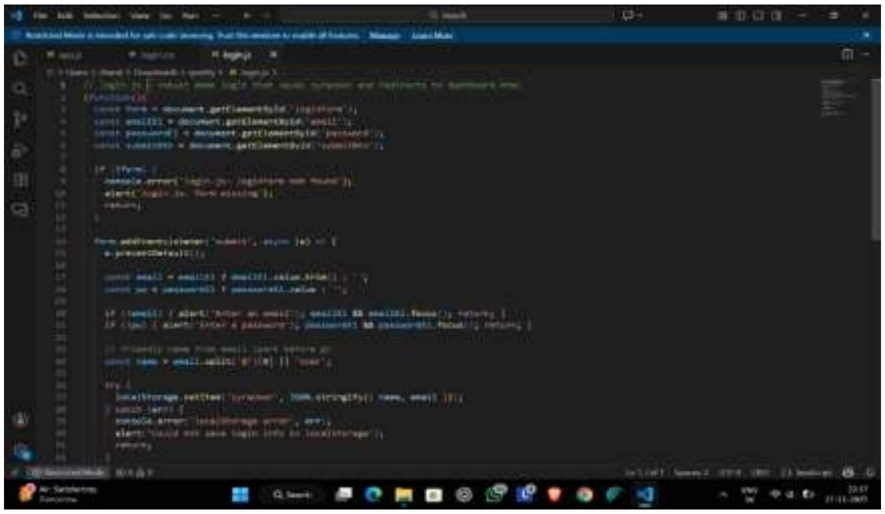
Paused

Upload (dev - optional)

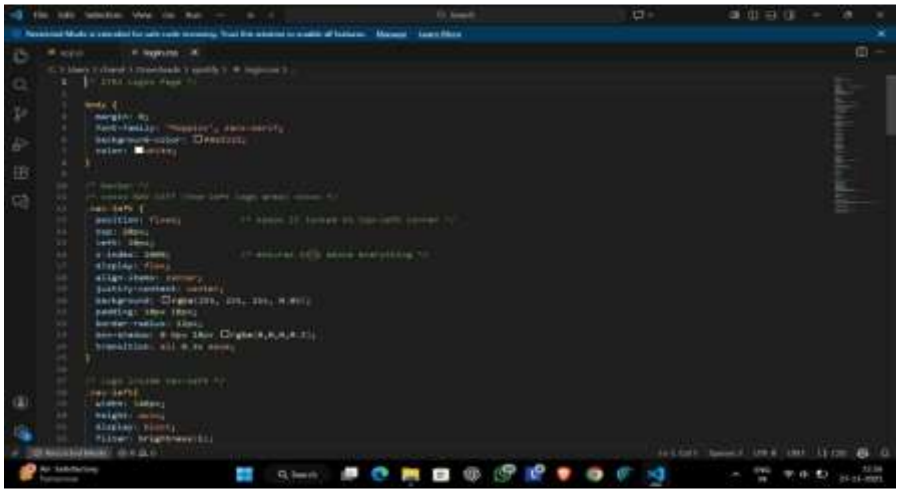
This form sends a file to your backend /api/songs if you have one. If you don't have a backend just ignore it.

Choose File No file chosen

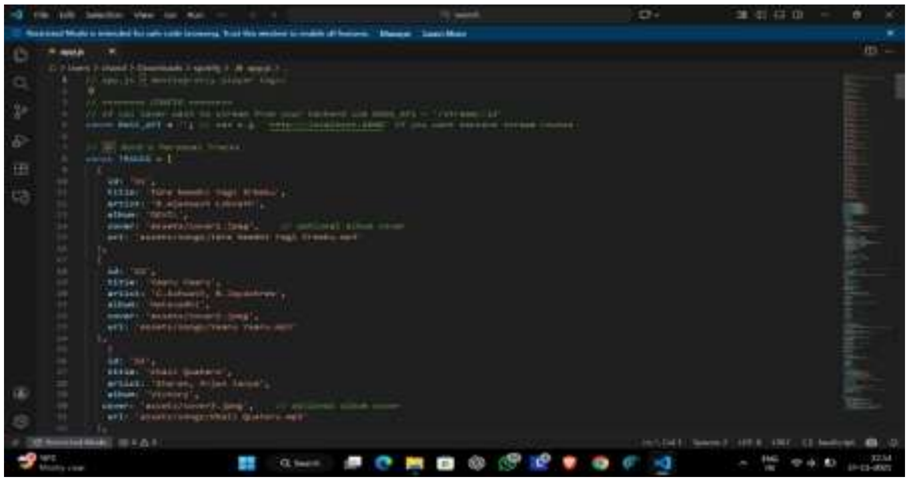
Upload



```
// login.js [illegible] robust demo login that saves 'zyrauser' and redirects to dashboard.html
(function(){
  const form = document.getElementById('loginForm');
  const emailEl = document.getElementById('email');
  const passwordEl = document.getElementById('password');
  const submitBtn = document.getElementById('submitBtn');

  if (!form) {
    console.error('login.js: loginForm not found');
    alert('login.js: form missing');
    return;
  }

  form.addEventListener('submit', async (e) => {
    e.preventDefault();

    const email = emailEl ? emailEl.value.trim() : '';
    const pw = passwordEl ? passwordEl.value : '';

    if (!email) { alert('Enter an email'); emailEl && emailEl.focus(); return; }
    if (!pw) { alert('Enter a password'); passwordEl && passwordEl.focus(); return; }

    // friendly name from email (part before @)
    const name = email.split('@')[0] || 'User';

    try {
      localStorage.setItem('zyrauser', JSON.stringify({ name, email }));
    } catch (err) {
      console.error('localStorage error', err);
      alert('Could not save login info to localStorage');
      return;
    }
```



```
/* ZYRA Login Page */

body {
  margin: 0;
  font-family: 'Poppins', sans-serif;
  background-color: [illegible];
  color: white;
}

/* Header */
/* [illegible] nav-left (top-left logo area) [illegible] */
.nav-left {
  position: fixed;          /* [illegible] it locked to top-left corner */
  top: 18px;
  left: 18px;
  z-index: 1000;            /* ensures it's above everything */
  display: flex;
  align-items: center;
  justify-content: center;
  background: rgba(255, 255, 255, 0.05);
  padding: 10px 18px;
  border-radius: 12px;
  box-shadow: 0 4px 18px rgba(0,0,0,0.2);
  transition: all 0.3s ease;
}

/* logo inside nav-left */
.nav-left{
  width: 140px;
  height: auto;
  display: block;
  filter: brightness(1);
```



```
// [illegible]

// ======== CONFIG ========
// [illegible]
const [illegible] = [illegible]

// [illegible]
const TRACKS = [
  {
    id: '[illegible]',
    title: '[illegible]',
    artist: '[illegible]',
    album: '[illegible]',
    cover: '[illegible]',    // [illegible]
    url: '[illegible]'
  },
  {
    id: '[illegible]',
    title: '[illegible]',
    artist: '[illegible]',
    album: '[illegible]',
    cover: '[illegible]',
    url: '[illegible]'
  },
  {
    id: '[illegible]',
    title: '[illegible]',
    artist: '[illegible]',
    album: '[illegible]',
   cover: '[illegible]',    // [illegible]
    url: '[illegible]'
  },
```

9.Testing

Form Validation

Form validation tests ensure that all input fields in the application, such as login forms, registration forms, and song upload forms, accept only valid data. This prevents incorrect or malicious inputs, such as empty fields, invalid email formats, or unsupported file types, from being processed by the system. Proper form validation improves user experience and maintains platform security.

Login

Login testing verifies that the authentication system works correctly for users, artists, and admins. It checks that valid credentials allow access, while invalid credentials are rejected. Tests may also cover features like password reset, session handling, and account lockout for multiple failed attempts to ensure secure access to the platform**.**

CRUD Operations

CRUD testing ensures that Create, Read, Update, and Delete operations function properly across the platform. This includes managing songs, playlists, comments, and user profiles. Proper testing guarantees that data is stored, retrieved, updated, and removed correctly without affecting other parts of the application**.**

Error Handling

Error handling tests check how the system responds to unexpected inputs, server errors, or failed operations. This includes handling missing files, database connection issues, invalid requests, or permission errors. Effective error handling ensures that the application does not crash and provides informative messages to users when problems occur.

Payment and Review

The payment system in Zyra enables users to subscribe to premium plans, purchase music, or unlock exclusive content securely. It integrates with trusted payment gateways to handle transactions safely, ensuring user data is protected. The review system allows listeners to provide feedback on songs, playlists, and artists, helping improve content quality and user engagement. Reviews and ratings are stored in the database, linked to the corresponding songs or artists, and displayed on the UI for other users. Together, these features support a monetization model while fostering an interactive and community-driven platform.

10. Result

The Zyra platform successfully provides a seamless music streaming experience for both artists and listeners. Artists can easily upload and manage their songs, albums, and playlists, while users can discover, play, and organize music with intuitive search, filters, and social interaction features. The platform demonstrates stable performance, secure authentication, efficient data handling, and smooth playback of audio files. Overall, the system meets the project objectives by creating a balanced and interactive ecosystem that enhances music discovery, content management, and user engagement.

11.Conclusion

The Zyra platform successfully fulfills its goal of creating a seamless and interactive music streaming environment. By integrating features like easy music uploads for artists, intuitive discovery tools for users, secure authentication, and efficient data management, the system provides a balanced ecosystem for both content creators and listeners. The layered architecture, coupled with robust backend logic and responsive UI, ensures stability, scalability, and a smooth user experience.

Overall, Zyra demonstrates how a well-designed music platform can enhance user engagement, simplify content management for artists, and offer a reliable, enjoyable, and efficient streaming experience. The project serves as a practical implementation of modern web technologies in the entertainment domain.

Future Enhancements

- Here's a list of future enhancements for Zyra in bullet points:
- Introduce AI-based music recommendations to personalize user playlists.
- Enable offline playback for premium users.
- Add multi-language support to reach a global audience.
- Implement advanced analytics for artists to track song performance and listener engagement.
- Integrate social sharing features to connect with other platforms like Instagram and Twitter.
- Support live streaming and virtual concerts within the app.
- Enhance payment options to include digital wallets and international currencies.
- Develop a mobile app with push notifications for updates, new releases, and promotions.

12.References

Django Official Documentation – Guides on using Django framework for web development. https://docs.djangoproject.com/

Python Official Documentation – Covers Python programming language and libraries. https://www.python.org/doc/

PostgreSQL Documentation – Provides instructions for using and managing PostgreSQL databases. https://www.postgresql.org/docs/

HTML, CSS, and JavaScript Tutorials – Tutorials for creating interactive and responsive web pages. https://www.w3schools.com/

AWS S3 Documentation – Explains cloud storage setup and file management. https://aws.amazon.com/s3/

Pillow Library Documentation – Guidance on image processing in Python. https://pillow.readthedocs.io/

Mutagen Documentation – Instructions for handling audio file metadata. https://mutagen.readthedocs.io/

Django REST Framework Documentation – Guides for building RESTful APIs with Django. https://www.django-rest-framework.org/