

Melodify (Online Music Library)

A Project Report

In the partial fulfillment of the award of the degree of

Masters in Computer Applications (MCA)

under

Academy of Skill Development



Submitted by

**Tanmoy Banerjee
Aishee Mitra
Agnishwar Khandait
Suvadip Shee
Rajdeep Kangsha Banik**



Heritage Institute of Technology



Certificate from the Mentor

This is to certify that **Agnishwar Khandait, Aishee Mitra, Tanmoy Banerjee, Rajdeep Kangsha Banik, Suvadip Shee** has completed the project titled **Melodify** under my supervision during the period from **January** to **April** which is in partial fulfillment of requirements for the award of the **MCA** and submitted to Department **Masters of Computer Applications** of **Heritage Institute of Technology**

Signature of the Mentor

Date:

Acknowledgment

I take this opportunity to express my deep gratitude and sincerest thanks to my project mentor, **Mr. SUBHOJIT SANTRA** for giving the most valuable suggestions, helpful guidance, and encouragement in the execution of this project work.

I would like to give a special mention to my colleagues. Last but not least I am grateful to all the faculty members of the **Academy of Skill Development** for their support.

Project Responsibility Form

Serial Number	Name of Member	Responsibility
1.	Aishee Mitra	Team Lead, Frontend Programming
2.	Agnishwar Khandait	Middleware Programming, Authentication
3.	Tanmoy Banerjee	Backend Programming, Documentation
4.	Suvadip Shee	Documentation, Additional Features Implementation
5.	Rajdeep Kangsha Banik	Documentation, Testing

Self Certificate

This is to certify that the dissertation/project proposal entitled “**Melodify (Online Music Platform)**”

Done by us, is an authentic work carried out for the partial fulfillment of the requirements for the award of the certificate of **Masters of Computer Applications** under the guidance of Mr. **Subhojit Santra**. The matter embodied in this project work has not been submitted earlier for the award of any certificate to the best of our knowledge and belief.

Name of the Student

- ☐ Aishee Mitra
- ☐ Agnishwar Khandait
- ☐ Tanmoy Banerjee
- ☐ Suvadip Shee
- ☐ Rajdeep Kangsha Banik

Signature of the students

a.

b.

c.

d.

e.

Certificate by Guide

This is to certify that this project entitled “**Melodify (Online Music Platform)**” submitted in partial fulfillment of the certificate of Masters of Computer Application through **Ardent Computech Pvt Ltd**, done by the Group Members

- ☐ Aishee Mitra
- ☐ Agnishwar Khandait
- ☐ Tanmoy Banerjee
- ☐ Suvadip Shee
- ☐ Rajdeep Kangsha Banik

Is an authentic work carried out under my guidance & best of our knowledge and belief.

- a.
- b.
- c.
- d.
- e.

Signature of the students

Signature of the Guide

Date :

Certificate of Approval

This is to certify that this proposal of Minor project, entitled “**Melodify** ” is a record of bona-fide work, carried out by: **Aishee Mitra, Agnishwar Khandait, Tanmoy Banerjee, Suvadip Shee, Rajdeep Kangsha Banik** under my supervision and guidance through the Ardent Computech Pvt Ltd. In my opinion, the report in its present form partially fulfils all the requirements, as specified by the **Heritage Institute of Technology** as per regulations of the **Ardent®**. It has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original and worthy of incorporation in the present version of the report for Masters of Computer Applications.

Guide/Supervisor

Mr. Subhojit Santra

Subject Matter Expert & Technical Head (Full Stack Web Development (MERN STACK)) Ardent Computech Pvt Ltd (An ISO 9001:2015 Certified) Module #132, Ground Floor, SDF Building, GP Block, Sector V, Bidhannagar, Kolkata, West Bengal, Kolkata
- 700091

External Examiner(s)

Head of the Department Department of Computer Applications **Anandapur , W.B(Heritage Institute of Technology)**

Contents:-

SL. NO.	NAME OF THE TOPIC
1	Company Profile
2	Introduction
3	Scope of the Project
4	Software Requirements Specification (SRS)
5	Data Flow Diagram
6	Sequence Diagram
7	Use – Case Diagram
8	System Implementation and Screenshots
9	Testing
10	Further Scope and Enhancements
11	Conclusion and Bibliography

1.ARDENT COMPUTECH PVT.LTD.

Ardent Computech Private Limited is an ISO 9001-2015 certified Software Development Company in India. It has been operating independently since 2003. It was recently merged with ARDENT TECHNOLOGIES.

Ardent Technologies

ARDENT TECHNOLOGIES is a Company successfully providing its services currently in the UK, USA, Canada and India. The core line of activity at ARDENT TECHNOLOGIES is to develop customized application software covering the entire responsibility of performing the initial system study, design, development, implementation and training. It also deals with consultancy services and Electronic Security systems. Its primary clientele includes educational institutes, entertainment industries, resorts, theme parks, service industry, telecommute operators, media and other business houses working in various capacities.

Ardent Collaborations

ARDENT COLLABORATIONS, the Research Training and Development Department of ARDENT COMPUTECH PVT LTD is a professional training Company offering IT enabled services & industrial training's for B-Tech, MCA, BCA, MSc and MBA freshers and experienced developers/programmers in various platforms. Summer Training / Winter Training / Industrial training will be provided for the students of B.TECH, M.TECH, MBA and MCA only. Deserving candidates may be awarded stipends, scholarships and other benefits, depending on their performance and recommendations of the mentors.

Associations

Ardent is an ISO 9001:2015 company.

It is affiliated to National Council of Vocational Training (NCVT), Directorate General of Employment & Training (DGET), Ministry of Labor & Employment, and Government of India.

INTRODUCTION: -

This report presents **Melodify**, a full-stack music streaming application developed as a Spotify clone using the **MERN** (MongoDB, Express.js, React, Node.js) stack. Designed to deliver a modern, interactive, and real-time user experience, Melodify incorporates cutting-edge web technologies to ensure performance, scalability, and usability. The frontend is built with **React and Vite**, offering a fast and responsive interface for seamless music playback and user interaction. On the backend, **Node.js and Express.js** handle routing, authentication, and data processing, with **MongoDB** serving as the persistent data store.

To enhance functionality, the project integrates various APIs and tools, including **Clerk** for authentication, **Socket.io** for real-time communication, and additional libraries to support robust features. Users can listen to music with next/previous track controls, adjust volume through a slider, and access a real-time chat system. A **search bar with integrated voice search** allows users to quickly find songs and artists using text or speech input, improving accessibility and user experience.

An interactive **customer support chatbot** provides real-time assistance to users, addressing queries and guiding them through app features. **Stripe** is integrated as a seamless payment gateway, enabling smooth and secure subscription purchases for premium features.

An **admin dashboard** enables content management (albums and songs), while features like online/offline status, live user activity tracking, and analytics dashboards elevate the platform beyond basic streaming. This report details the development process, technology stack, key features, and architectural decisions that shaped the creation of **Melodify**.

Scope of The Project

The scope of **Melodify** encompasses the design, development, and deployment of a fully functional music streaming web application that replicates and extends the core features of Spotify. This project is intended to demonstrate the practical implementation of a full-stack architecture using the **MERN** stack, while integrating real-time and data-driven functionalities to enhance user engagement.

The core features within the scope of this project include:






- Music playback capabilities with play, pause, next, and previous controls.
- A responsive and intuitive user interface built with React and Vite.
- A backend API developed using Node.js and Express.js to manage users, songs, albums, and real-time interactions.
- **Authentication and user management** using Clerk.
- Real-time features such as chat and user status updates via **Socket.io**.
- An **admin dashboard** for creating and managing albums and songs.
- Visibility into **what other users are listening to**, promoting community interaction.
- **Analytics and data aggregation** for monitoring platform usage and user activity.
- Storage and retrieval of data using **MongoDB**, ensuring scalability and persistence.
- A **search bar with integrated voice search** to help users easily find songs and artists using either text input or voice commands.
- A **customer support chatbot** to assist users in real-time, addressing queries and providing feature guidance.
- **Stripe integration** for secure and seamless payment processing, enabling users to subscribe to premium features.

The scope also includes deployment and testing of the application to ensure cross-device compatibility, system performance, and user experience. However, the project does not cover mobile application development or licensing of commercial music content; all music used is for demonstration purposes only.

Software Requirement Specification (SRS)

Software Requirements Specification provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specification document enlists enough necessary requirements that are required for the project development. To derive the requirements, we need to have a clear and thorough understanding of the project to be developed. This is prepared after detailed communication with the project team and the customer.

The developer is responsible for: -

-  Developing the system, which meets the SRS and solves all the requirements of the system.
-  Demonstrating the system and installing the system at the client's location after acceptance testing is successful.
-  Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.
-  Conducting any user training that might be needed for using the system.
-  Maintain the system for a period of one year after installation.

Product Perspective

Melodify is a standalone full-stack web application. It replicates core Spotify functionalities with added real-time communication features and administrative controls. It does not require integration with any existing system.

Product Functions

- User registration, login, and authentication (Clerk)
- Music playback with controls (play, pause, next, previous)
- Volume adjustment via slider
- Real-time chat system with online/offline presence (Socket.io)
- Display of current activity (what users are listening to)
- Admin dashboard for content creation (songs, albums)
- Data aggregation and display on analytics dashboard
- **Search functionality with voice input** to find songs and artists efficiently
- **Customer support chatbot** to assist users with real-time help and guidance
- **Stripe-powered subscription system** for secure premium feature payments

User Classes and Characteristics

- **Guest Users:** Can view basic features but cannot stream music or chat.
- **Registered Users:** Can stream music, chat, and view user activity.
- **Administrators:** Have full access including song/album management and analytics.

Operating Environment

- Web browsers: Chrome, Firefox, Edge, Safari
- Operating systems: Windows, macOS, Linux
- Backend: Node.js runtime
- Database: MongoDB Atlas

Design and Implementation Constraints

- Real-time features require persistent WebSocket connections (Socket.io)
- Authentication is dependent on Clerk service availability
- Only demo music files will be used (no copyrighted content)

Specific Requirements

Functional Requirements

- **FR1:** The system shall allow users to register and log in using Clerk.
- **FR2:** The system shall allow users to play, pause, skip, and adjust volume.
- **FR3:** The system shall maintain and display online/offline user status.
- **FR4:** The system shall enable real-time chat between active users.
- **FR5:** The system shall allow administrators to create and manage songs/albums.
- **FR6:** The system shall display what music each user is currently listening to.
- **FR7:** The system shall display usage data and analytics to administrators.
- **FR8:** The system shall provide a **search bar with voice search** to find songs and artists.
- **FR9:** The system shall provide a **customer support chatbot** for real-time assistance.
- **FR10:** The system shall support **Stripe-powered payment** for premium subscriptions.

Non-Functional Requirements

- **NFR1:** The system should be responsive and accessible across all modern browsers.
- **NFR2:** The system should support up to 100 concurrent users in real-time chat.
- **NFR3:** The backend should respond to API requests within 300ms on average.

- **NFR4:** All user data should be securely stored and managed using encrypted connections.
- **NFR5:** The frontend should load within 2 seconds on a standard broadband connection.

Software Specifications

- **Frontend Framework:** React.js with Vite
- **Backend Runtime:** Node.js
- **Server Framework:** Express.js
- **Database:** MongoDB (Atlas)
- **Authentication:** Clerk API
- **Payment Gateway:** Stripe
- **Voice Search:** Web Speech API (or similar) integrated in frontend
- **Chatbot:** Custom or third-party chatbot service integrated in frontend
- **Real-Time Communication:** Socket.io
- **Version Control:** Git and GitHub
- **Other Tools:** VS Code (IDE)

Hardware Specifications

For Development:

- **Processor:** Intel i5 or higher / AMD Ryzen 5 or higher
- **RAM:** Minimum 8 GB
- **Storage:** At least 20 GB available
- **OS:** Windows 10+, macOS, or Linux
- **Internet:** Stable broadband connection for real-time testing and deployments

For Deployment (Server Requirements):

- **RAM:** 1-2 GB (minimum for backend hosting)
- **CPU:** 1 vCPU or higher (Render/Heroku tier)
- **Storage:** Sufficient for static assets and logs (~1-2 GB)
- **Database:** MongoDB Atlas cluster (M0 or higher)

Data Flow Diagram: -

A *data flow diagram (DFD)* is a graphical representation of the "flow" of data through an information system, modeling its process aspects. **ADFD** is often used as a preliminary step to create an overview of the system, which can later be elaborated.

DFD can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).


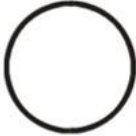


This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows towards or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present for the system to do its job and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data-flow diagrams can be drawn up and compared with

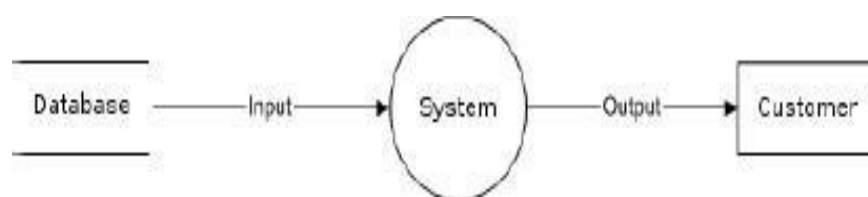
How any system is developed can be determined through a data flow diagram model.

In the course of developing a set of *leveled* data flow diagrams, the analyst/designer is forced to address how the system may be decomposed into component sub-systems and to identify the transaction data in the data model. Data flow diagrams can be used in both the Analysis and Design phase of the **SDLC**. There are different notations to draw data flow diagrams. Defining different visual representations for processes, data stores, data flow, and external entities.

DFD Notation: -

	dataflow	Arrows showing direction of flow
	process	circles
	file	horizontal pair of lines
	data-source, sink	rectangular box

Example of DFD: -



Steps to Construct Data Flow Diagram: -

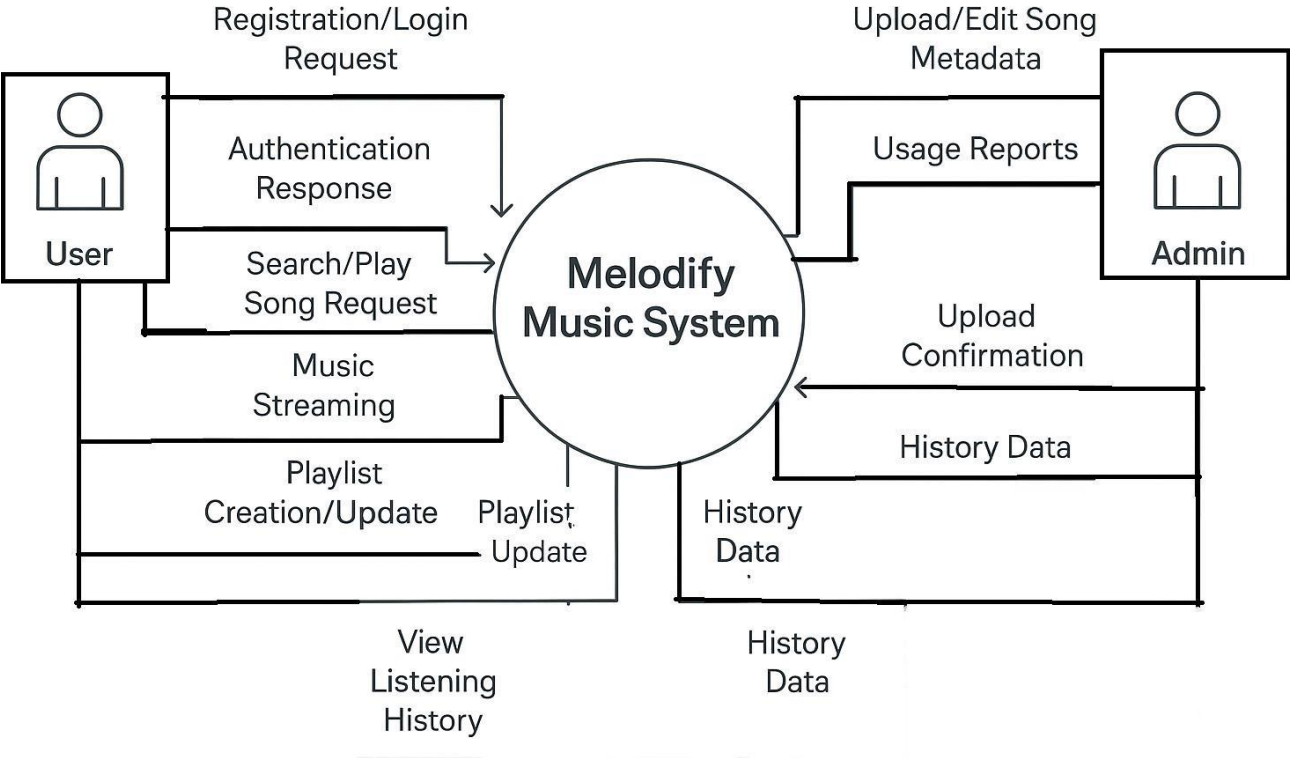
Four Steps are generally used to construct a DFD.

- ✚ Process should be named and referred for easy reference. Each name should be representative of the reference.
- ✚ The destination of flow is from top to bottom and from left to right.
- ✚ When a process is distributed into lower-level details they are numbered.
- ✚ The names of data stores, sources, and destinations are written in capital letters.

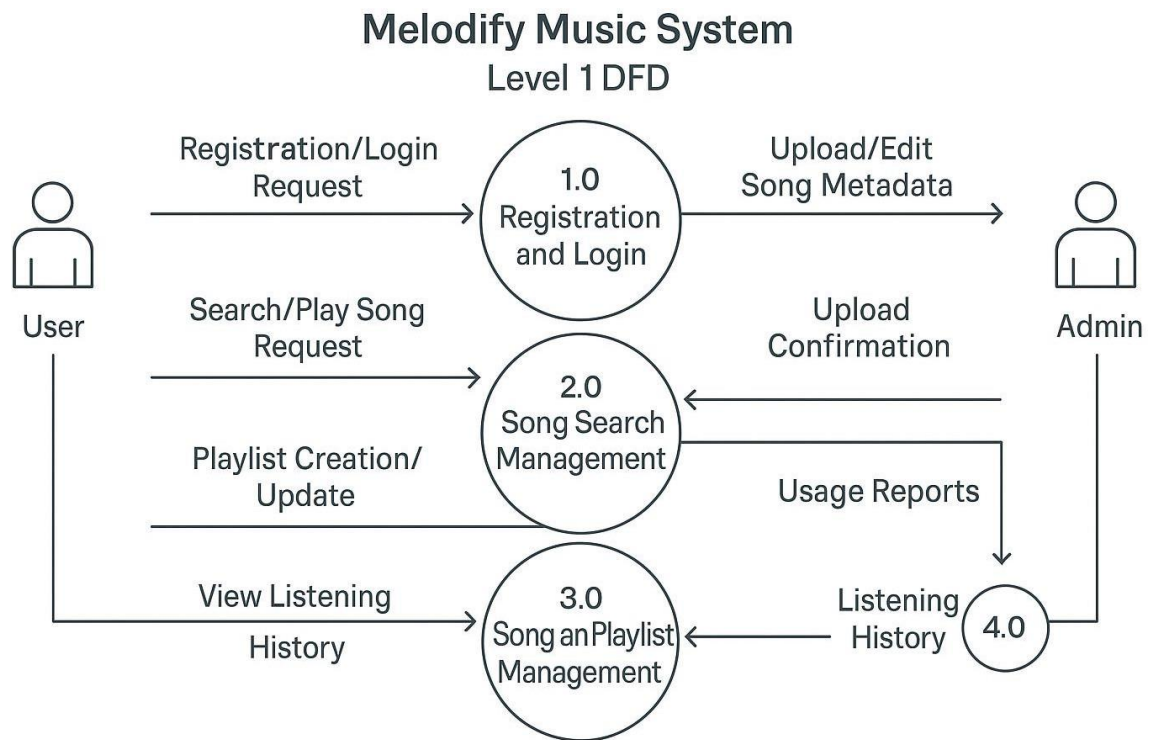
Rules for constructing a Data Flow Diagram: -

- ✚ Arrows should not cross each other.
- ✚ Squares, Circles, and Files must bear a name.
- ✚ Decomposed data flow squares and circles can have the same names.
- ✚ Draw all data flow around the outside of the diagram.

Level 0 DFD (Context Diagram) :-



Level 1 DFD: -



SEQUENCE DIAGRAM: -

A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

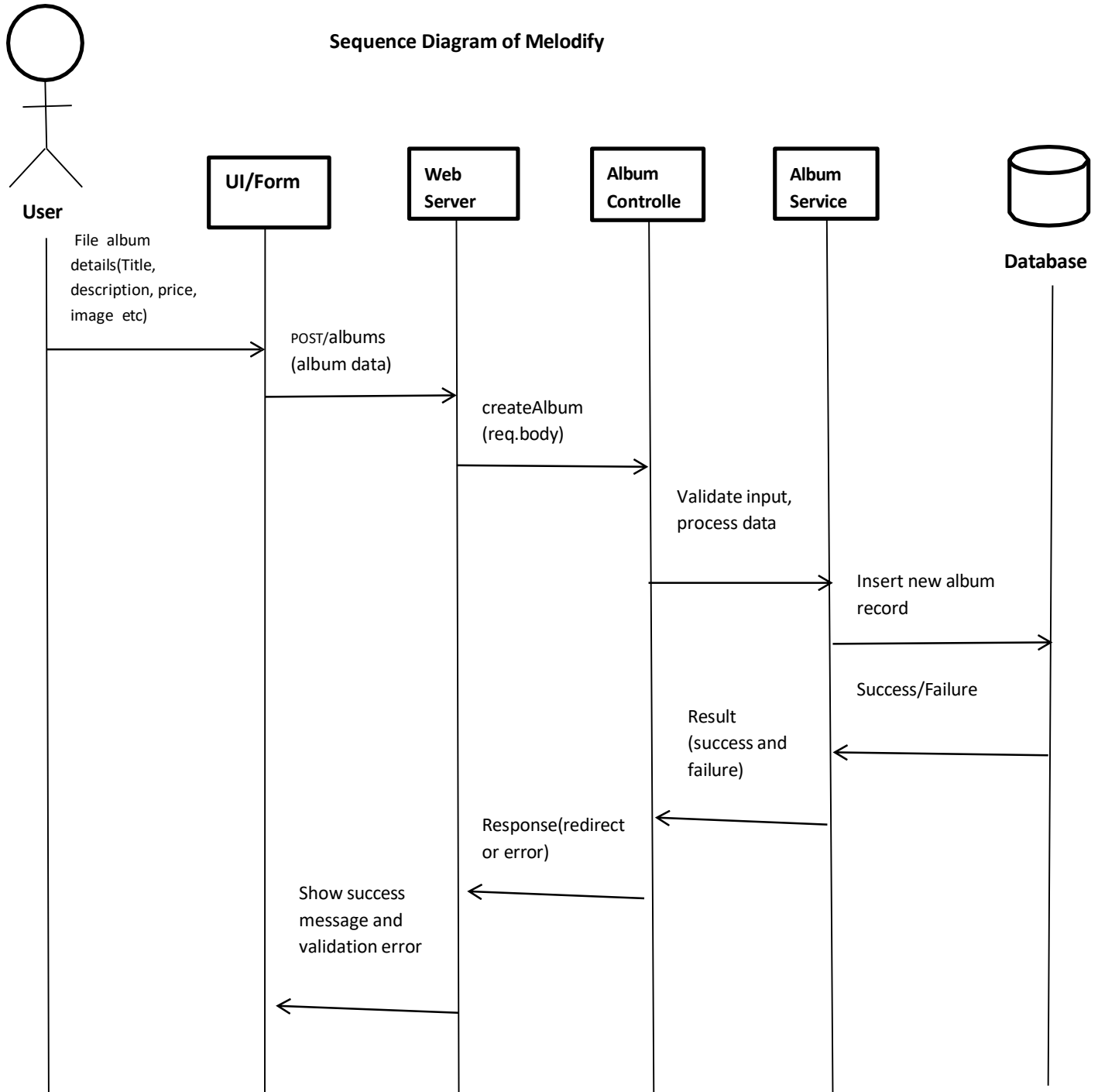
Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between several lifelines. A sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a UML sequence diagram: lifeline, execution specification, message, fragment, interaction, state invariant, continuation, and destruction occurrence.

Sequence Diagram of Melodify



Use-Case Diagram: -

A **Use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams consist of actors, use cases, and their relationships. The diagrams used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.

So to model the entire system numbers of use case diagrams are used. The purpose of a use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because the other four diagrams (activity, sequence, collaboration, and State chart) are also having the same purpose. So we will look into some specific purpose that will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modeled to present the outside view.

So in brief, the purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interaction among the requirements actors.

How to draw Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed, the functionalities are captured in use cases.

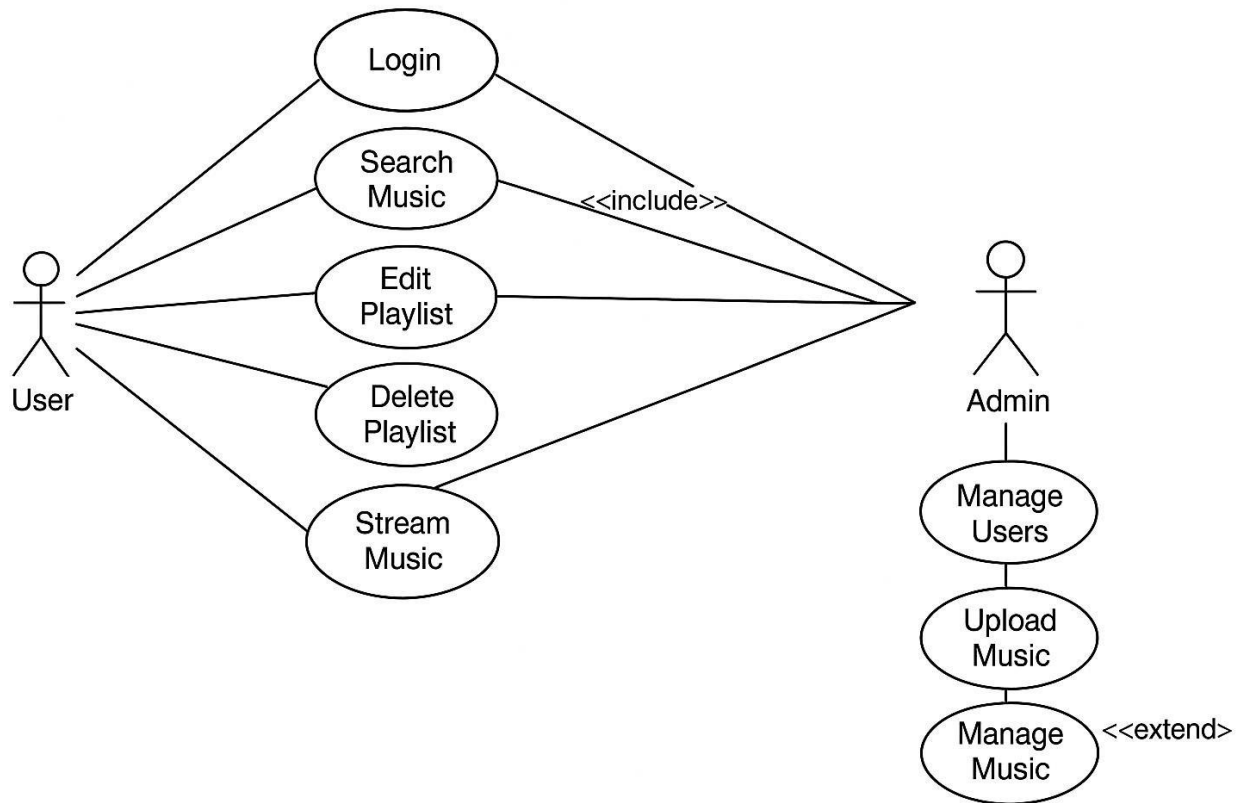
So, we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system. The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw a use case diagram, we should have the following items identified.

- ✚ **Functionalities** to be represented as a use case
- ✚ **Actors**
- ✚ **Relationships** among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

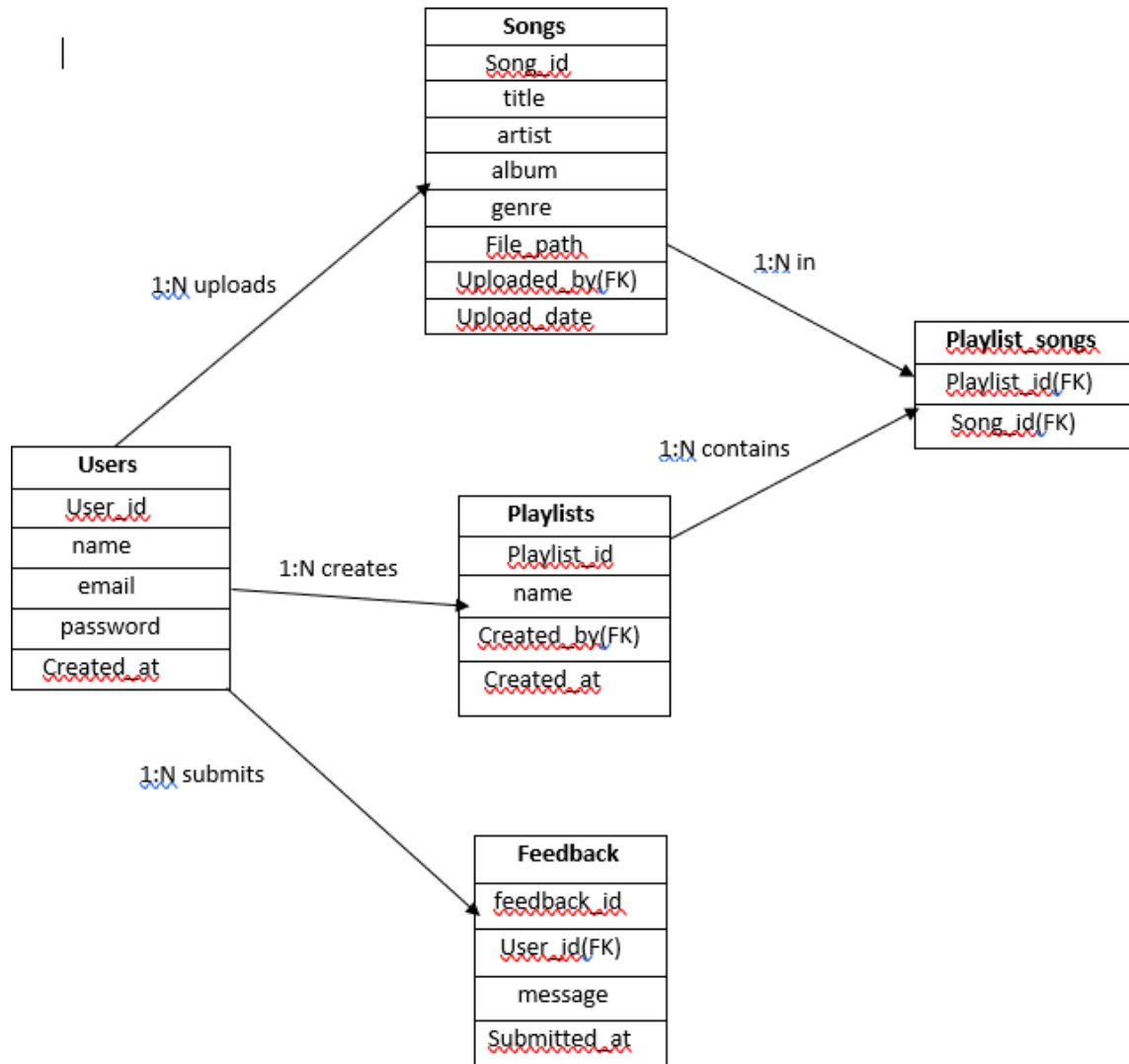
- ✚ The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.
- ✚ Give a suitable name for actors.
- ✚ Show relationships and dependencies clearly in the diagram.
- ✚ Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- ✚ Use note whenever required to clarify some important point

Use-Case Diagram: -



Schema Diagram: -

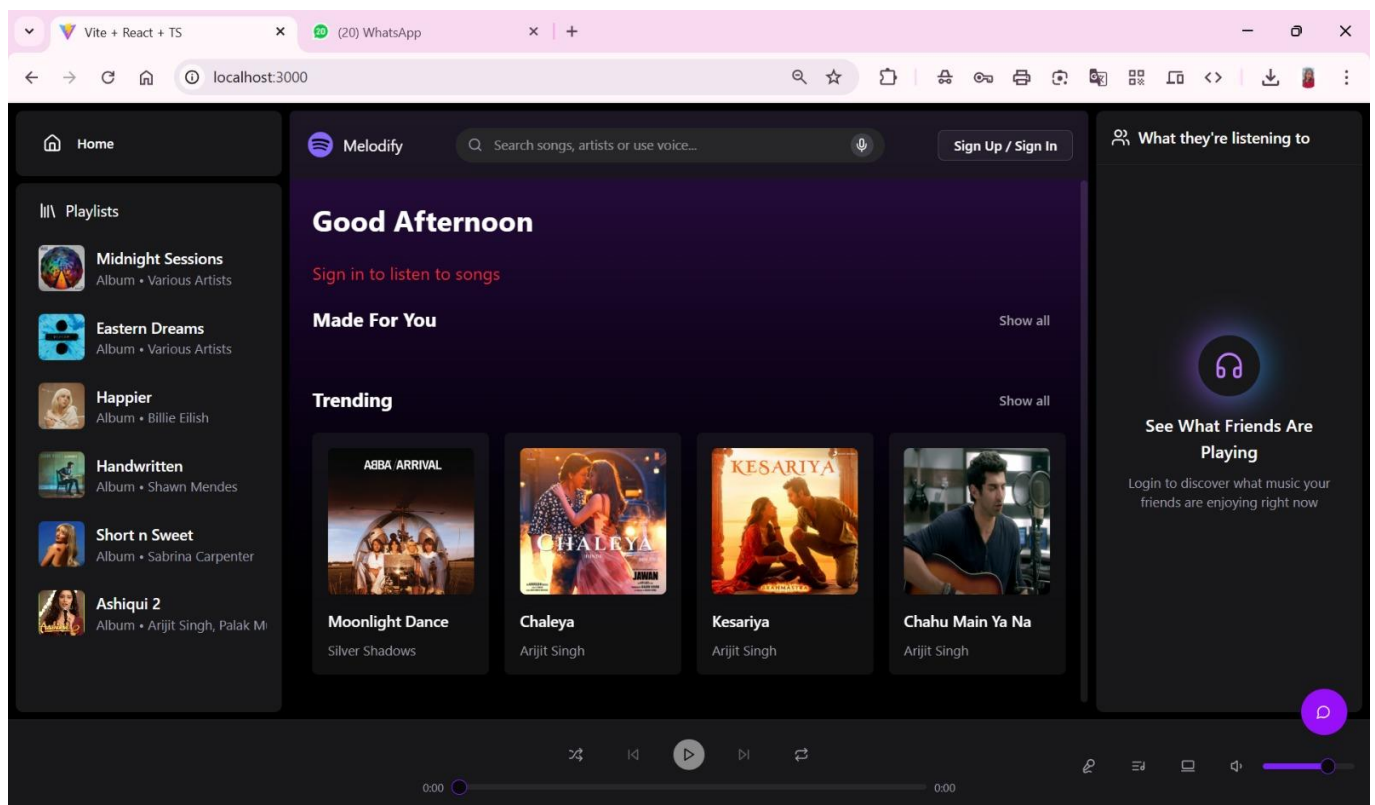
Schema Diagram



System Implementation and Screenshots: -

The following screenshots provide a visual overview of **Melodify's** key features and user interface components. They illustrate the core functionality of the application, including music playback, real-time chat, admin dashboard, user activity tracking, and more. These images showcase both frontend design and backend integration, highlighting the seamless user experience and real-time capabilities built using the MERN stack and supporting technologies.

Homepage Overview: -



Homepage's Code:-

```
import Topbar from "@/components/ui/Topbar";
import { useMusicStore } from "@/stores/useMusicStore";
import { useEffect } from "react";
import FeaturedSection from "@/components/FeaturedSection";
import { ScrollArea } from "@/components/ui/scroll-area";
import SectionGrid from "@/components/SectionGrid";
import { usePlayerStore } from "@/stores/usePlayerStore";

const HomePage = () => {
  const {
    fetchFeaturedSongs,
    fetchMadeForYouSongs,
    fetchTrendingSongs,
    isLoading,
    madeForYouSongs,
    featuredSongs,
    trendingSongs,
    searchResults
  } = useMusicStore();

  const { initializeQueue } = usePlayerStore();

  useEffect(() => {
    fetchFeaturedSongs();
    fetchMadeForYouSongs();
    fetchTrendingSongs();
  }, [fetchFeaturedSongs, fetchMadeForYouSongs, fetchTrendingSongs]);

  useEffect(() => {
    if(madeForYouSongs.length > 0 && featuredSongs.length > 0 && trendingSongs.length > 0) {
      const allSongs = [...featuredSongs, ...madeForYouSongs, ...trendingSongs];
      initializeQueue(allSongs);
    }
  }, [initializeQueue, madeForYouSongs, trendingSongs, featuredSongs]);

  return (
    <main className="rounded-md overflow-hidden h-full bg-gradient-to-b from-[#2c1045]/100 to-black">
      <Topbar />
      <ScrollArea className="h-[calc(100vh-180px)]">
        <div className="p-4 sm:p-6">
          {searchResults.length > 0 ? (
            <>
              <h1 className="text-2xl sm:text-3xl font-bold mb-6">Search Results</h1>
              <SectionGrid
                title="Search Results" // Added title prop
                songs={searchResults}
                isLoading={isLoading}
              />
            </>
          ) : (
            <FeaturedSection
              madeForYouSongs={madeForYouSongs}
              featuredSongs={featuredSongs}
              trendingSongs={trendingSongs}
              isLoading={isLoading}
            />
          )
        }
      </div>
    </main>
  )
}
```

```
    ): (  
      <>  
        <h1 className="text-2xl sm:text-3xl font-bold mb-6">Good Afternoon</h1>  
        <FeaturedSection />  
        <div className="space-y-8">  
          <SectionGrid  
            title="Made For You"  
            songs={madeForYouSongs}  
            isLoading={isLoading}  
          />  
          <SectionGrid  
            title="Trending"  
            songs={trendingSongs}  
            isLoading={isLoading}  
          />  
        </div>  
      </>  
    )}  
  </div>  
</ScrollArea>  
</main>  
)  
};  
export default HomePage;
```

Topbar Code :-

```
import { LayoutDashboardIcon } from "lucide-react";
import { Link } from "react-router-dom";
import { SignedIn, SignedOut, UserButton, useUser } from "@clerk/clerk-react";
import { useAuthStore } from "@/stores/useAuthStore";
import { buttonVariants } from "./button";
import { cn } from "@/lib/utls";
import { useEffect } from "react";

const Topbar = () => {
  const { isSignedIn } = useUser();
  const { isAdmin, checkAdminStatus } = useAuthStore();

  useEffect(() => {
    if (isSignedIn) {
      checkAdminStatus(); // fetch admin status after sign in
    }
  }, [isSignedIn]);

  return (
    <div className="flex items-center justify-between p-4 sticky top-0 bg-zinc-900/75 backdrop-blur-md z-10">
      <div className="flex gap-2 items-center">
        
        Melodify
      </div>

      <div className="flex items-center gap-4">
        <SignedIn>
          {isAdmin && (
            <Link to="/admin" className={cn(buttonVariants({ variant: "outline" })))}>

```

```

        <LayoutDashboardIcon className="size-4 mr-2" />
        Admin Dashboard
      </Link>
    })
    <UserButton />
  </SignedIn>

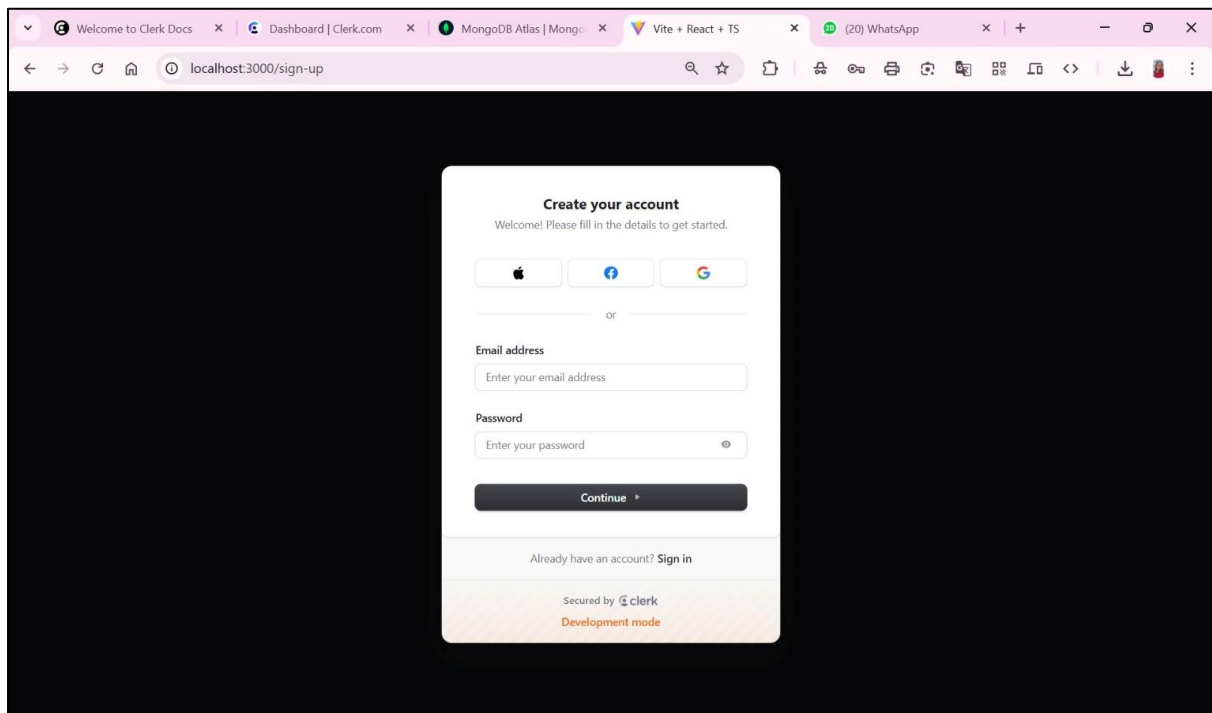
  <SignedOut>
    <div className="flex gap-2 items-center">
      <Link to="/sign-up" className={cn(buttonVariants({ variant:
"outline" })), "h-8 px-4")}>
        Sign Up / Sign In
      </Link>
    </div>
  </SignedOut>

</div>
</div>
);
};

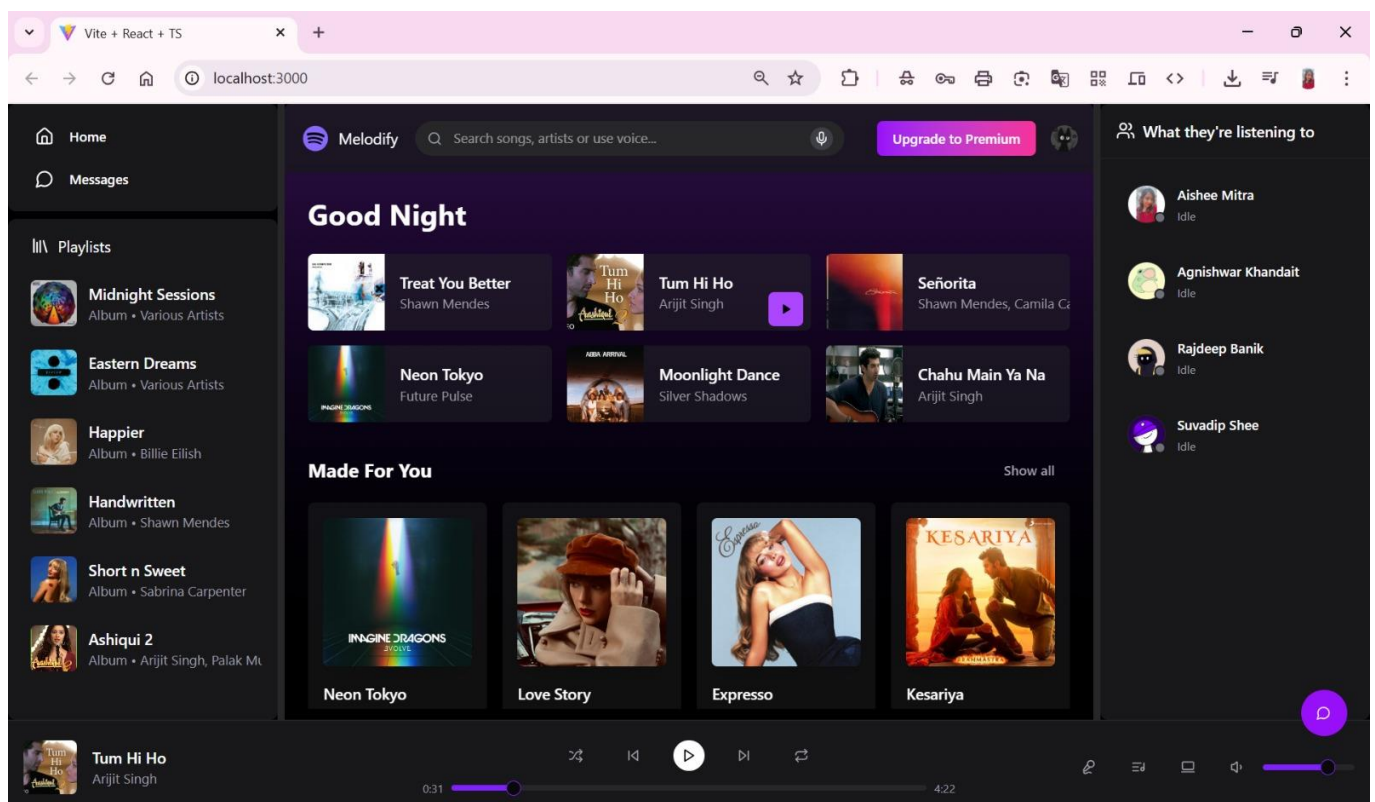
export default Topbar;

```

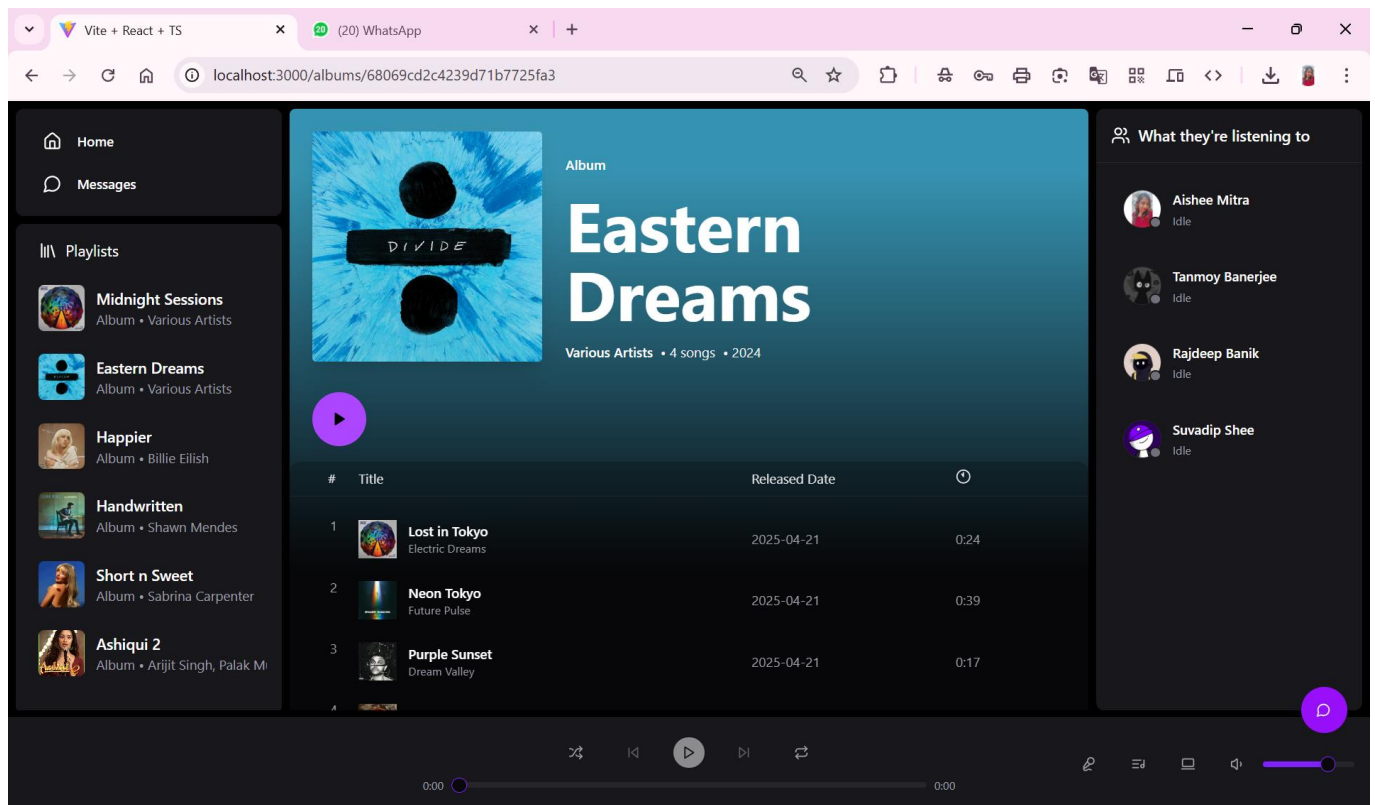
Signup Page :-



After Login Homepage: -



Album Page :-



Album Page Code :-

```
import { Button } from "@components/ui/button";
import { useDominantColor } from "@lib/useDominantColor";
import { useMusicStore } from "@stores/useMusicStore";
import { usePlayerStore } from "@stores/usePlayerStore";
import { ScrollArea } from "@radix-ui/react-scroll-area";
import { Clock11Icon, Pause, Play } from "lucide-react";
import { useEffect } from "react";
import { useParams } from "react-router-dom";

const formatDuration = (seconds: number) => {
  const minutes = Math.floor(seconds / 60);
  const remainingSeconds = seconds % 60;
  return `${minutes}:${remainingSeconds.toString().padStart(2, "0")}`;
}

const AlbumPage = () => {
  const { albumId } = useParams();
  const { fetchAlbumById, currentAlbum, isLoading } = useMusicStore();
```



```

const { currentSong, isPlaying, playAlbum, togglePlay } = usePlayerStore();

useEffect(() => {
  if (albumId) fetchAlbumById(albumId);
}, [fetchAlbumById, albumId]);

const dominantColor = useDominantColor(currentAlbum?.imageUrl);

if (isLoading || !currentAlbum) return null;

const handlePlayAlbum = () => {
  if(!currentAlbum) return

  const isCurrentAlbumPlaying = currentAlbum?.songs.some(song => song._id
=== currentSong?._id);
  if(isCurrentAlbumPlaying) togglePlay();
  else {
    //start playing the album from the beginning
    playAlbum(currentAlbum?.songs, 0)
  }
}

const handlePlaySong = (index: number) => {
  playAlbum(currentAlbum.songs, index);
}

return (
  <div className="h-full">
    <ScrollArea className="h-full rounded-md">
      {/* Main Content */}
      <div className="relative min-h-full rounded-t-md overflow-hidden">
        {/* Dynamic background gradient */}
        <div
          className="absolute inset-0 pointer-events-none"
          style={{
            background: `linear-gradient(to bottom, ${dominantColor} 0%,
${dominantColor} 10%, #09090b 70%, #09090b 100%)`,
          }}
          aria-hidden="true"
        />

        {/* Content */}
        <div className="relative z-10">
          <div className="flex p-6 gap-6 pb-8">
            <img
              src={currentAlbum?.imageUrl}
              alt={currentAlbum?.title ?? "Album cover"}
              className="w-[240px] h-[240px] shadow-xl rounded"
            />
            <div className="flex flex-col justify-end">

```

```

    <p className="text-sm font-medium">Album</p>
    <h1 className="text-7xl font-bold my-4">{currentAlbum.title}</h1>
    <div className="flex items-center gap-2 text-sm text-zinc-100">
      <span className="font-medium text-
white">{currentAlbum.artist}</span>
      <span>• {currentAlbum.songs.length} songs</span>
      <span>• {currentAlbum.releaseYear}</span>
    </div>
  </div>
</div>

{/* Play button */}
<div className="px-6 pb-4 flex items-center gap-6">
  <Button
    onClick={handlePlayAlbum}
    size="icon"
    className="w-14 h-14 rounded-full bg-purple-500 hover:bg-purple-
400 hover:scale-105 transition-all"
    >
    {isPlaying && currentAlbum?.songs.some(song => song._id ===
currentSong?._id) ? (
      <Pause className="h-7 w-7 stroke-black fill-black" />
    ) : (
      <Play className="h-7 w-7 stroke-black fill-black" />
    )}
  </Button>
</div>

{/* Table Section */}
<div className="bg-black/20 backdrop-blur-sm rounded-xl overflow-
hidden">
  {/* Table header */}
  <div
    className="grid grid-cols-[16px_4fr_2fr_1fr] gap-4 px-10 py-2 text-
sm
    text-zinc-200 border-b border-white/5"
    >
    <div>#</div>
    <div>Title</div>
    <div>Released Date</div>
    <div>
      <Clock1Icon className="h-4 w-4" />
    </div>
  </div>

  {/* Songs list */}
  <div className="px-6">
    <div className="space-y-2 py-4">
      {currentAlbum?.songs.map((song, index) => {
        const isCurrentSong = currentSong?._id === song._id

```

```

        return(
        <div
          key={song._id}
          onClick={() => handlePlaySong(index)}
          className="grid grid-cols-[16px_4fr_2fr_1fr] gap-4 px-4 py-2
text-sm
pointer"
          text-zinc-400 hover:bg-white/5 rounded-md group cursor-
          >
3">
            <div className="flex items-center justify-center relative w-3 h-
              {isCurrentSong && isPlaying ? (
                <div className="text-purple-400">🎵 </div>
              ) : (
                <span className="group-hover:hidden">{index + 1}</span>
              )}
              {!isCurrentSong && (
                <Play className="h-3 w-3 absolute hidden group-hover:block
stroke-white fill-white" />
              )}
            </div>

            <div className="flex items-center gap-3">
              <img
                src={song.imageUrl}
                alt={song.title}
                className="size-10"
              />
              <div className="flex flex-col">
                <div className="font-medium text-white">{song.title}</div>
                <div className="text-xs text-zinc-400">{song.artist}</div>
              </div>
            </div>

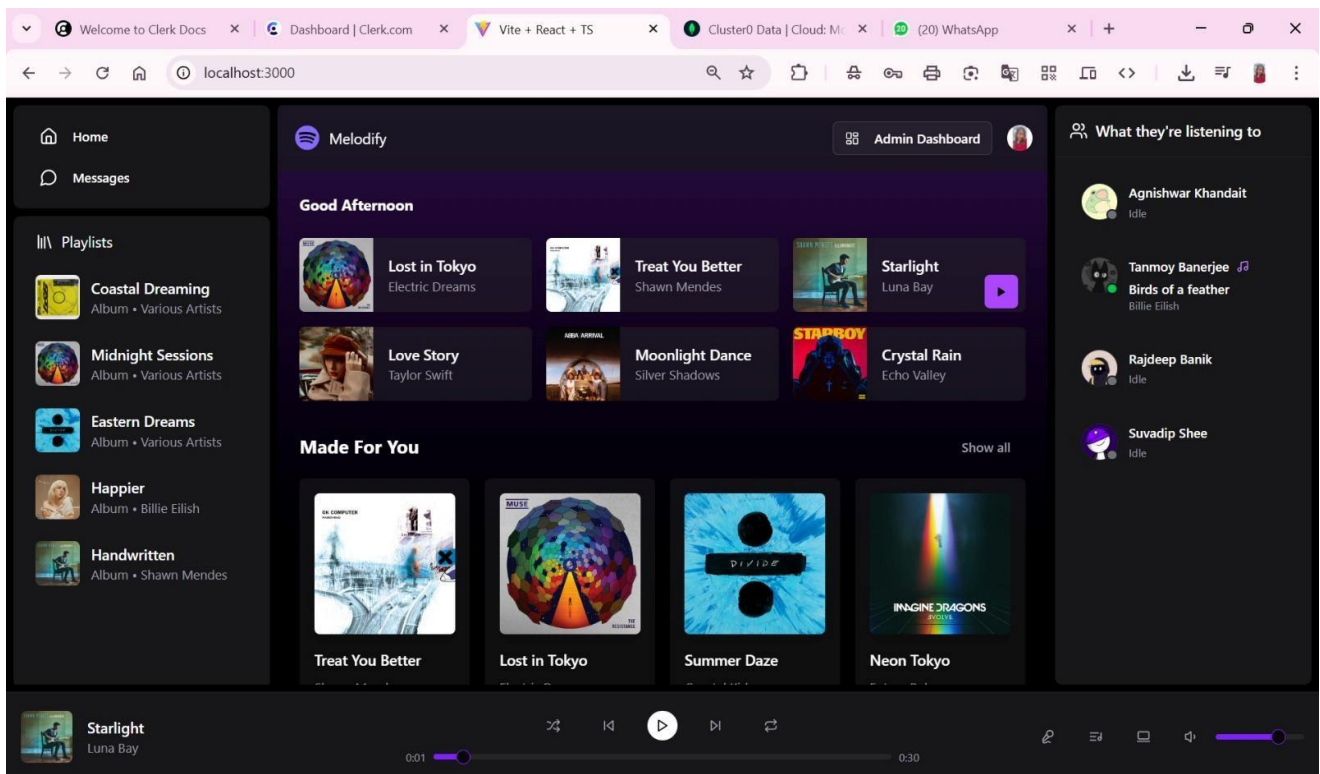
            <div className="flex items-center">
              {song.createdAt.split("T")[0]}
            </div>
            <div className="flex items-
center">{formatDuration(song.duration)}</div>
          </div>
        )
      }
    )}
  </div>
</div>
</div>
</div>
</div>
</ScrollArea>

```

```
</div>
);
};

export default AlbumPage;
```

Admin Homepage: -



Admin Homepage Code :-

```
import { useAuthStore } from "@stores/useAuthStore";
import Header from "../components/Header";
import DashboardStats from "../components/DashboardStats";
import { Tabs, TabsContent, TabsList, TabsTrigger } from
"@components/ui/tabs";
import { Album, Music } from "lucide-react";
import SongsTabContent from "../components/SongsTabContent";
import AlbumsTabContent from "../components/AlbumsTabContent";
import { useEffect } from "react";
import { useMusicStore } from "@stores/useMusicStore";

const AdminPage = () => {
  const { isAdmin, isLoading } = useAuthStore();

  const { fetchAlbums, fetchSongs, fetchStats } = useMusicStore();

  useEffect(() => {
    fetchAlbums()
    fetchSongs()
    fetchStats()
  },[fetchAlbums, fetchSongs, fetchStats]);

  if(!isAdmin && !isLoading) return <div>Unauthorized</div>;

  return <div className="min-h-screen bg-gradient-to-b from-zinc-900 via-
zinc-900 to-black text-zinc-100 p-8">
    <Header />

    <DashboardStats />

    <Tabs defaultValue="songs" className="space-y-6">
      <TabsList className="p-1 bg-zinc-800/50">
        <TabsTrigger value="songs" className="data-[state=active]:bg-zinc-
700">
          <Music className="mr-2 size-4" />
          Songs
        </TabsTrigger>
        <TabsTrigger value="albums" className="data-[state=active]:bg-zinc-
700">
          <Album className="mr-2 size-4" />
          Albums
        </TabsTrigger>
      </TabsList>

      <TabsContent value='songs' >
        <SongsTabContent />
      </TabsContent>
    </Tabs>
  </div>
```

```

    </TabsContent>
    <TabsContent value='albums' >
      <AlbumsTabContent />
    </TabsContent>
  </Tabs>

</div>;
};

export default AdminPage;

```

Admin Songs Dashboard: -

Music Manager
Manage your music catalog

Total Songs: 13 | Total Albums: 5 | Unique Artists: 14 | Total Users: 5

Songs Library
Manage your music tracks

[+ Add Song](#)

Title	Artist	Release Date	Actions
Treat You Better	Shawn Mendes	2025-05-03	
Birds of a feather	Billie Eilish	2025-05-01	
Love Story	Taylor Swift	2025-05-01	
Summer Daze	Coastal Kids	2025-04-21	
Ocean Waves	Coastal Drift	2025-04-21	

Songs Dashboard Code :-

```
import { Button } from "@components/ui/button";
import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from
"@components/ui/table";
import { useMusicStore } from "@stores/useMusicStore";
import { Calendar, Trash2 } from "lucide-react";

const SongsTable = () => {
  const { songs, isLoading, error, deleteSong } = useMusicStore();

  if (isLoading) {
    return (
      <div className='flex items-center justify-center py-8'>
        <div className='text-zinc-400'>Loading songs...</div>
      </div>
    );
  }

  if (error) {
    return (
      <div className='flex items-center justify-center py-8'>
        <div className='text-red-400'>{error}</div>
      </div>
    );
  }

  return (
    <Table>
      <TableHeader>
        <TableRow className='hover:bg-zinc-800/50'>
          <TableHead className='w-[50px]'></TableHead>
          <TableHead>Title</TableHead>
          <TableHead>Artist</TableHead>
          <TableHead>Release Date</TableHead>
          <TableHead className='text-
right'>Actions</TableHead>
        </TableRow>
      </TableHeader>

      <TableBody>
        {songs.map((song) => (
          <TableRow key={song._id} className='hover:bg-
zinc-800/50'>
            <TableCell>
              <img src={song.imageUrl}
alt={song.title} className='size-10 rounded object-cover' />
            </TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>
  );
}
```

```

        medium'>{song.title}</TableCell>
        <TableCell className='font-
        <TableCell>{song.artist}</TableCell>
        <TableCell>
            <span className='inline-flex items-
            <Calendar className='h-4 w-
            {song.createdAt.split("T")[0]}
            </span>
        </TableCell>
        <TableCell className='text-right'>
            <div className='flex gap-2 justify-
            end'>
                <Button
                    variant={"ghost"}
                    size={"sm"}
                    className='text-red-
                    400 hover:text-red-300 hover:bg-red-400/10'
                    deleteSong(song._id)}
                >
                    <Trash2
                    className='size-4' />
                </Button>
            </div>
        </TableCell>
    </TableRow>
    )}
</TableBody>
</Table>
);
};
export default SongsTable;

```


Admin Albums Library: -

The screenshot shows the 'Admin Albums Library' interface of the Music Manager application. The browser address bar indicates the URL is `localhost:3000/admin`. The application header includes the Spotify logo, the title 'Music Manager', and the subtitle 'Manage your music catalog'. A user profile icon is visible in the top right corner. Below the header, there are four summary cards: 'Total Songs' (13), 'Total Albums' (5), 'Unique Artists' (14), and 'Total Users' (5). A toggle switch allows switching between 'Songs' and 'Albums' views, with 'Albums' currently selected. The main section is titled 'Albums Library' with the subtitle 'Manage your album collection' and an '+ Add Album' button. It contains a table with the following data:

Title	Artist	Release Year	Songs	Actions
Coastal Dreaming	Various Artists	2024	4 songs	
Midnight Sessions	Various Artists	2024	3 songs	
Eastern Dreams	Various Artists	2024	4 songs	
Happier	Billie Eilish	2025	1 songs	
Handwritten	Shawn Mendes	2016	1 songs	

Add New Song Interface: -

The screenshot shows the 'Add New Song' modal interface of the Music Manager application. The browser address bar indicates the URL is `localhost:3000/admin`. The application header is the same as the previous screenshot. The modal is titled 'Add New Song' with the subtitle 'Add a new song to your music library'. It contains the following fields and controls:

- An 'Upload artwork' section with a dashed box and a 'Choose File' button.
- An 'Audio File' section with a 'Choose Audio File' button.
- A 'Title' text input field.
- An 'Artist' text input field.
- A 'Duration (seconds)' text input field with the value '0'.

The background shows the 'Songs Library' section with a table of songs:

Title	Artist	Release Year	Actions
Treat You Better			
Birds of a feather			
Love Story			
Summer Daze			
Ocean Waves	Coastal Drift	2025-04-21	

Add Song Code :-

```
import { Button } from "@components/ui/button";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@components/ui/dialog";
import { Input } from "@components/ui/input";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
"@components/ui/select";
import { axiosInstance } from "@lib/axios";
import { useMusicStore } from "@stores/useMusicStore";
import { Plus, Upload } from "lucide-react";
import { useRef, useState } from "react";
import toast from "react-hot-toast";

interface NewSong {
  title: string;
  artist: string;
  album: string;
  duration: string;
}

const AddSongDialog = () => {
  const { albums } = useMusicStore();
  const [songDialogOpen, setSongDialogOpen] = useState(false);
  const [isLoading, setIsLoading] = useState(false);

  const [newSong, setNewSong] = useState<NewSong>({
    title: "",
    artist: "",
    album: "",
    duration: "0",
  });

  const [files, setFiles] = useState<{ audio: File | null; image: File | null
}>({
    audio: null,
    image: null,
  });

  const audioInputRef = useRef<HTMLInputElement>(null);
  const imageInputRef = useRef<HTMLInputElement>(null);
```

```

const handleSubmit = async () => {
    setIsLoading(true);

    try {
        if (!files.audio || !files.image) {
            return toast.error("Please upload both audio and
image files");
        }

        const formData = new FormData();

        formData.append("title", newSong.title);
        formData.append("artist", newSong.artist);
        formData.append("duration", newSong.duration);
        if (newSong.album && newSong.album !== "none") {
            formData.append("albumId", newSong.album);
        }

        formData.append("audioFile", files.audio);
        formData.append("imageFile", files.image);

        await axiosInstance.post("/admin/songs", formData, {
            headers: {
                "Content-Type": "multipart/form-data",
            },
        });

        setNewSong({
            title: "",
            artist: "",
            album: "",
            duration: "0",
        });

        setFiles({
            audio: null,
            image: null,
        });
        toast.success("Song added successfully");
    } catch (error: any) {
        toast.error("Failed to add song: " + error.message);
    } finally {
        setIsLoading(false);
    }
};

return (
    <Dialog open={songDialogOpen}
onOpenChange={setSongDialogOpen}>
        <DialogTrigger asChild>

```

```

500 text-black'>
    <Button className='bg-purple-400 hover:bg-purple-
    <Plus className='mr-2 h-4 w-4' />
    Add Song
  </Button>
</DialogTrigger>

  <DialogContent className='bg-zinc-900 border-zinc-700
max-h-[80vh] overflow-auto'>
    <DialogHeader>
      <DialogTitle>Add New Song</DialogTitle>
      <DialogDescription>Add a new song to your
music library</DialogDescription>
    </DialogHeader>

    <div className='space-y-4 py-4'>
      <input
        type='file'
        accept='audio/*'
        ref={audioInputRef}
        hidden
        onChange={(e) => setFiles((prev) => ({
...prev, audio: e.target.files![0] })))}
      />

      <input
        type='file'
        ref={imageInputRef}
        className='hidden'
        accept='image/*'
        onChange={(e) => setFiles((prev) => ({
...prev, image: e.target.files![0] })))}
      />

      {/* image upload area */}
      <div
        className='flex items-center justify-
center p-6 border-2 border-dashed border-zinc-700 rounded-lg cursor-pointer'
        onClick={() =>
imageInputRef.current?.click()}
      >

        <div className='text-center'>
          {files.image ? (
            <div className='space-y-2'>
              <div className='text-
sm text-emerald-500'>Image selected:</div>
              <div className='text-
xs text-zinc-400'>{files.image.name.slice(0, 20)}</div>
            </div>
          ) : (

```

```

        bg-zinc-800 rounded-full inline-block mb-2'>
        className='h-6 w-6 text-zinc-400' />

        sm text-zinc-400 mb-2'>Upload artwork</div>
        variant='outline' size='sm' className='text-xs'>

        Choose File
        </Button>
    </>
    )}
</div>
</div>

    { /* Audio upload */ }
    <div className='space-y-2'>
        <label className='text-sm font-
medium'>Audio File</label>
        <div className='flex items-center gap-
2'>
            <Button variant='outline'
onClick={() => audioInputRef.current?.click()} className='w-full'>
                {files.audio ?
files.audio.name.slice(0, 20) : "Choose Audio File"}
            </Button>
        </div>
    </div>

    { /* other fields */ }
    <div className='space-y-2'>
        <label className='text-sm font-
medium'>Title</label>
        <Input
            value={newSong.title}
            onChange={(e) => setNewSong({
...newSong, title: e.target.value })}
            className='bg-zinc-800 border-
zinc-700'
        />
    </div>

    <div className='space-y-2'>
        <label className='text-sm font-
medium'>Artist</label>
        <Input
            value={newSong.artist}

```

```

...newSong, artist: e.target.value }}
zinc-700'
    />
  </div>

  <div className='space-y-2'>
    <label className='text-sm font-
medium'>Duration (seconds)</label>
    <Input
      type='number'
      min='0'
      value={newSong.duration}
      onChange={(e) => setNewSong({
...newSong, duration: e.target.value || "0" })}
      className='bg-zinc-800 border-
zinc-700'
    />
  </div>

  <div className='space-y-2'>
    <label className='text-sm font-
medium'>Album (Optional)</label>
    <Select
      value={newSong.album}
      onValueChange={(value) =>
setNewSong({ ...newSong, album: value })}
    >
      <SelectTrigger className='bg-zinc-
800 border-zinc-700'>
        <SelectValue
placeholder='Select album' />
      </SelectTrigger>
      <SelectContent className='bg-
zinc-800 border-zinc-700'>
        <SelectItem value='none'>No
Album (Single)</SelectItem>
        {albums.map((album) => (
          <SelectItem
            key={album._id} value={album._id}>
              {album.title}
            </SelectItem>
          )))}
      </SelectContent>
    </Select>
  </div>
</div>

<DialogFooter>

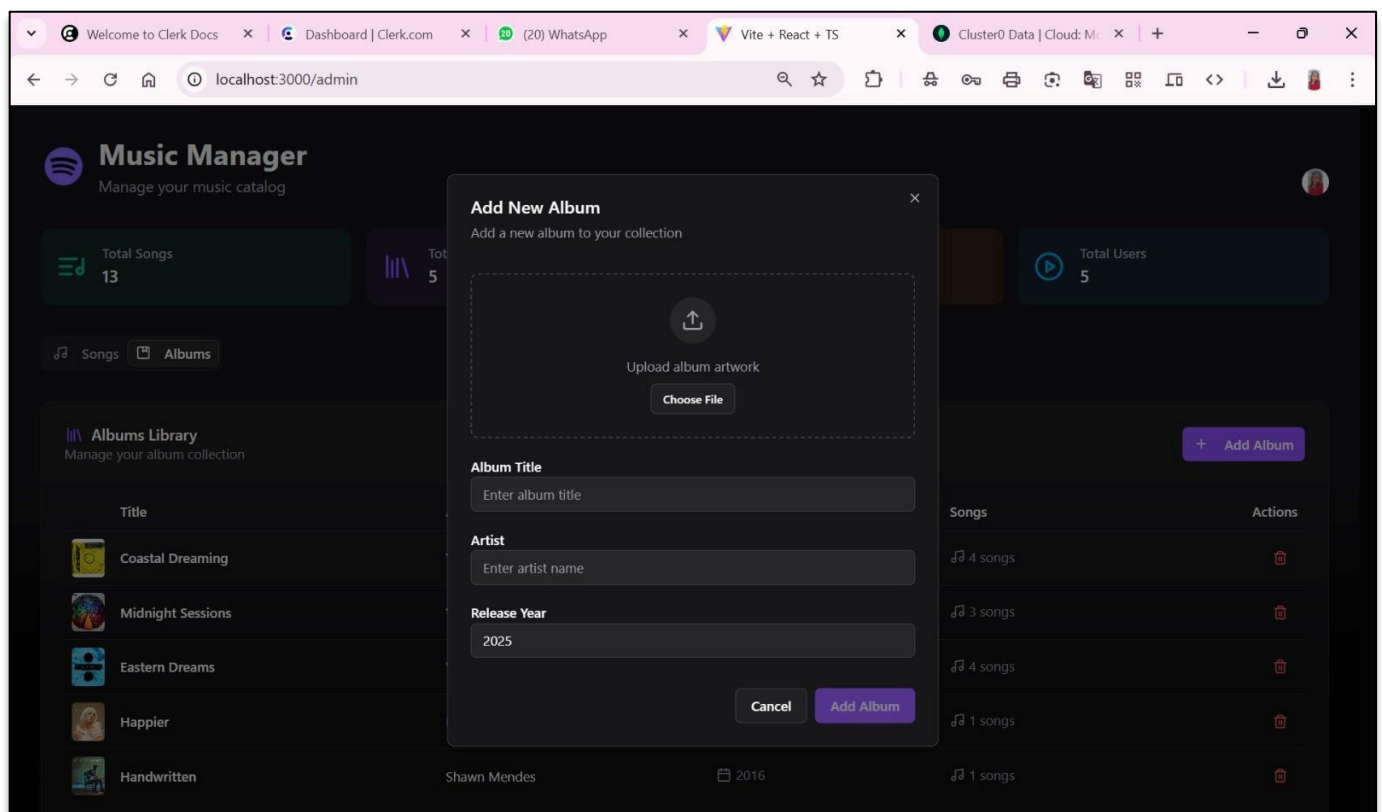
```

```

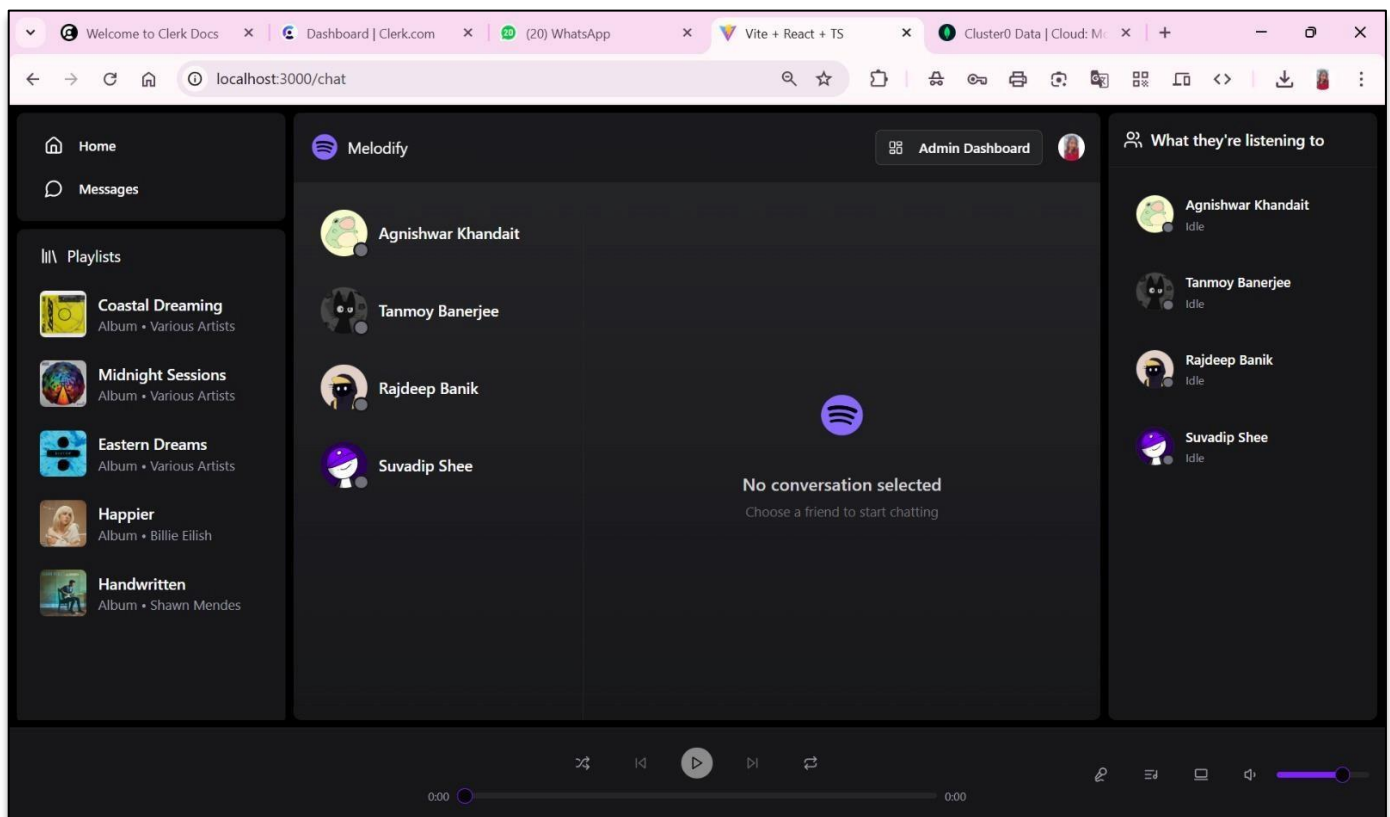
        <Button variant='outline' onClick={() =>
setSongDialogOpen(false)} disabled={isLoading}>
            Cancel
        </Button>
        <Button onClick={handleSubmit}
disabled={isLoading}>
            {isLoading ? "Uploading..." : "Add Song"}
        </Button>
    </DialogFooter>
</DialogContent>
</Dialog>
    );
};
export default AddSongDialog;

```

Add New Album Interface: -



Chat Page :-



Chat Page Code :-

```
import Topbar from "@/components/ui/Topbar";
import { useChatStore } from "@/stores/useChatStore";
import { useUser } from "@clerk/clerk-react";
import { useEffect } from "react";
import UsersList from "@/components/UsersList";
import ChatHeader from "@/components/ChatHeader";
import { ScrollArea } from "@/components/ui/scroll-area";
import { Avatar, AvatarImage } from "@/components/ui/avatar";
import MessageInput from "@/components/MessageInput";

const formatTime = (date: string) => {
  return new Date(date).toLocaleTimeString("en-US", {
    hour: "2-digit",
    minute: "2-digit",
    hour12: true,
  });
};

const ChatPage = () => {
```



```

const { user } = useUser();
const { messages, selectedUser, fetchUsers, fetchMessages } =
useChatStore();

useEffect(() => {
  if (user) fetchUsers();
}, [fetchUsers, user]);

useEffect(() => {
  if (selectedUser) fetchMessages(selectedUser.clerkId);
}, [selectedUser, fetchMessages]);

console.log({ messages });

return (
  <main className='h-full rounded-lg bg-gradient-to-b from-zinc-800
to-zinc-900 overflow-hidden'>
    <Topbar />

    <div className='grid lg:grid-cols-[300px_1fr] grid-cols-
[80px_1fr] h-[calc(100vh-180px)]'>
      <UsersList />

      {/* chat message */}
      <div className='flex flex-col h-full'>
        {selectedUser ? (
          <>
            <ChatHeader />

            {/* Messages */}
            <ScrollArea className='h-
[calc(100vh-340px)]'>
              <div className='p-4 space-
y-4'>
                {messages.map((message) => (
                  <div
                    key={message._id}
                    className={`flex items-start gap-3 ${
                      message.senderId === user?.id ? "flex-row-reverse" : ""
                    }`}
                    >
                      <Avatar
                        className='size-8'>
                        <AvatarImage

```

```

src={
    message.senderId === user?.id
      ? user.imageUrl
      : selectedUser.imageUrl
}
      />

</Avatar>

      <div

className={`rounded-lg p-3 max-w-[70%]
  ${message.senderId === user?.id ? "bg-purple-500" : "bg-zinc-800"}
  `}
    >
      <p
className='text-sm'>{message.content}</p>

      <span className='text-xs text-zinc-300 mt-1 block'>
        {formatTime(message.createdAt)}
      </span>

      </div>
    </div>
  )}
</div>
</ScrollArea>

    <MessageInput />
  </>
) : (
  <NoConversationPlaceholder />
)
</div>
</div>
</main>
);
};
export default ChatPage;

const NoConversationPlaceholder = () => (
  <div className='flex flex-col items-center justify-center h-full space-y-6'>

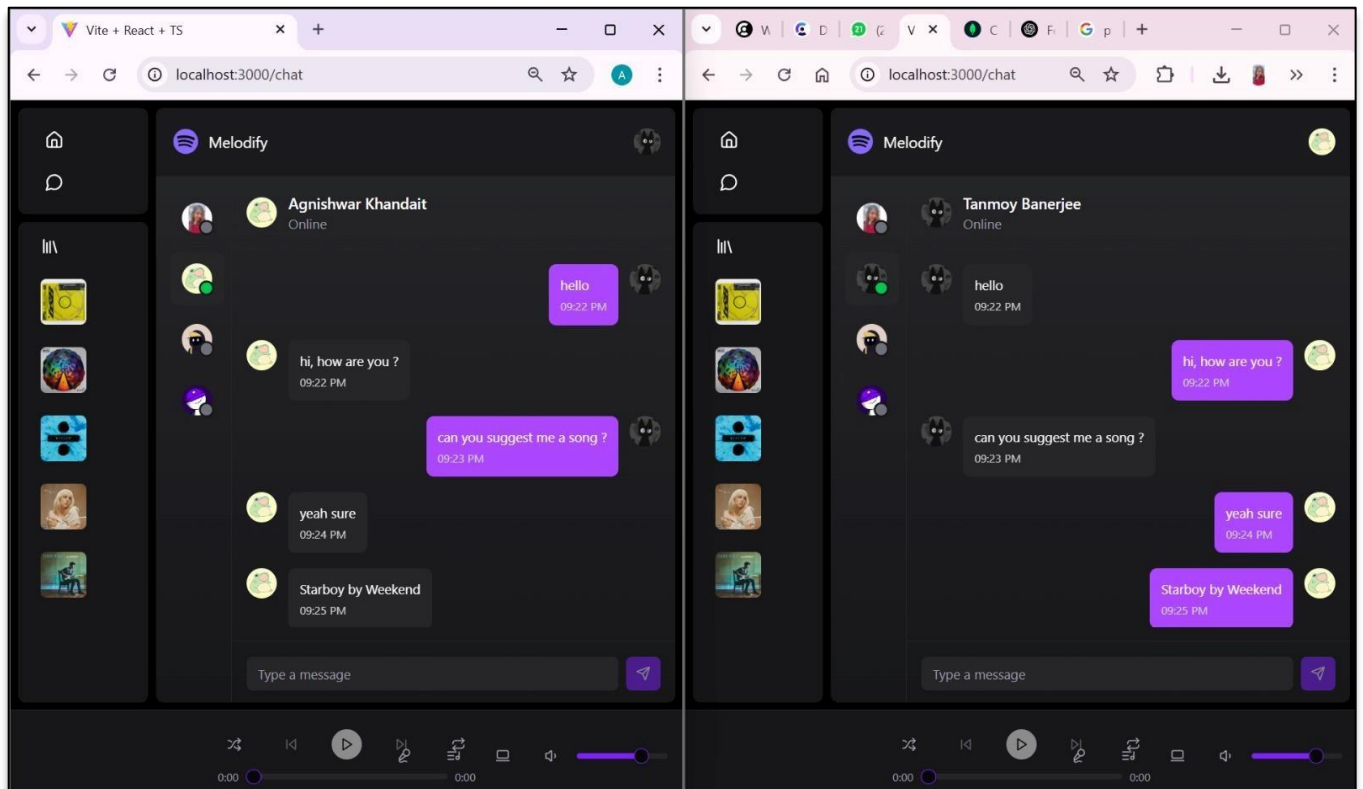
```

```

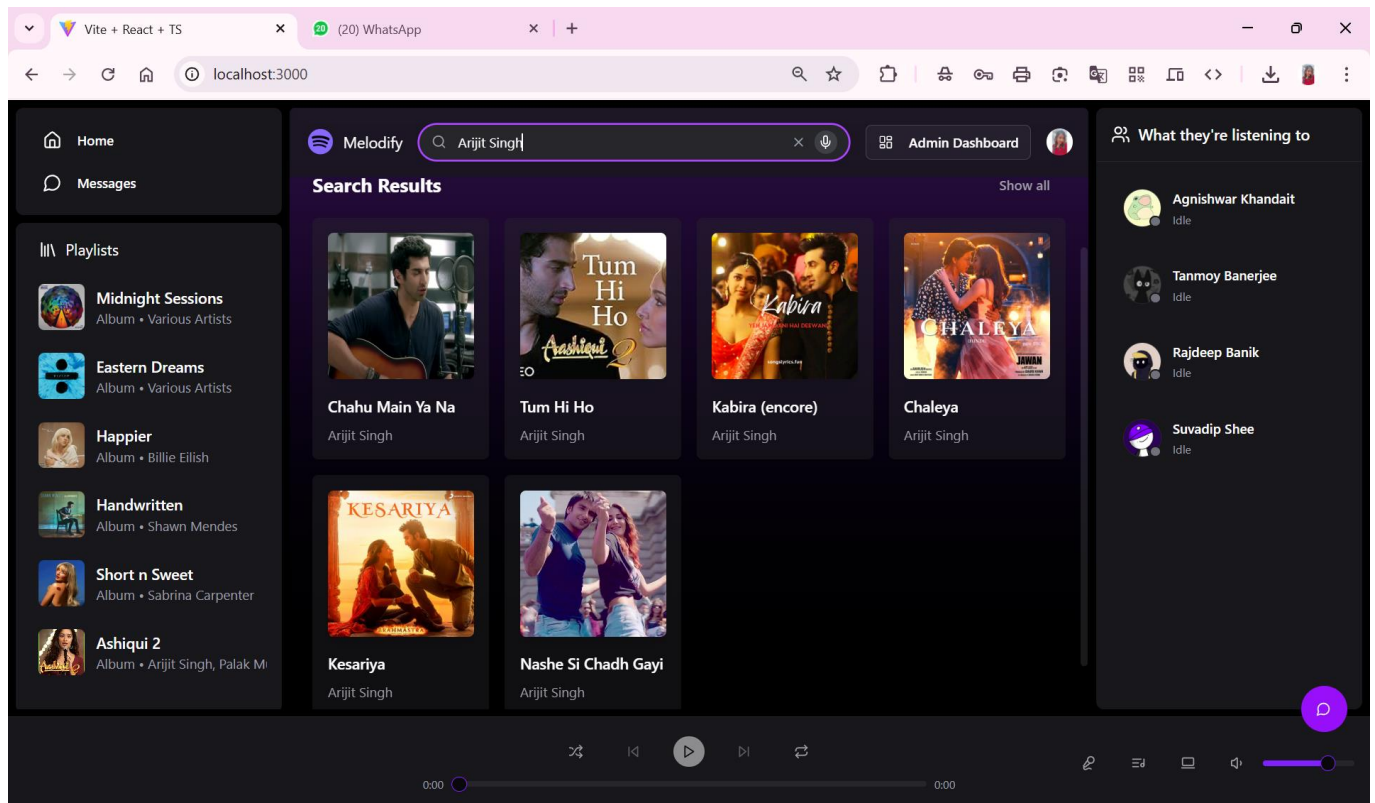
    <img src='/melodify.png' alt='Melodify' className='h-16 w-13
animate-bounce' />
    <div className='text-center'>
      <h3 className='text-zinc-300 text-lg font-medium mb-1'>No
conversation selected</h3>
      <p className='text-zinc-500 text-sm'>Choose a friend to
start chatting</p>
    </div>
  </div>
);

```

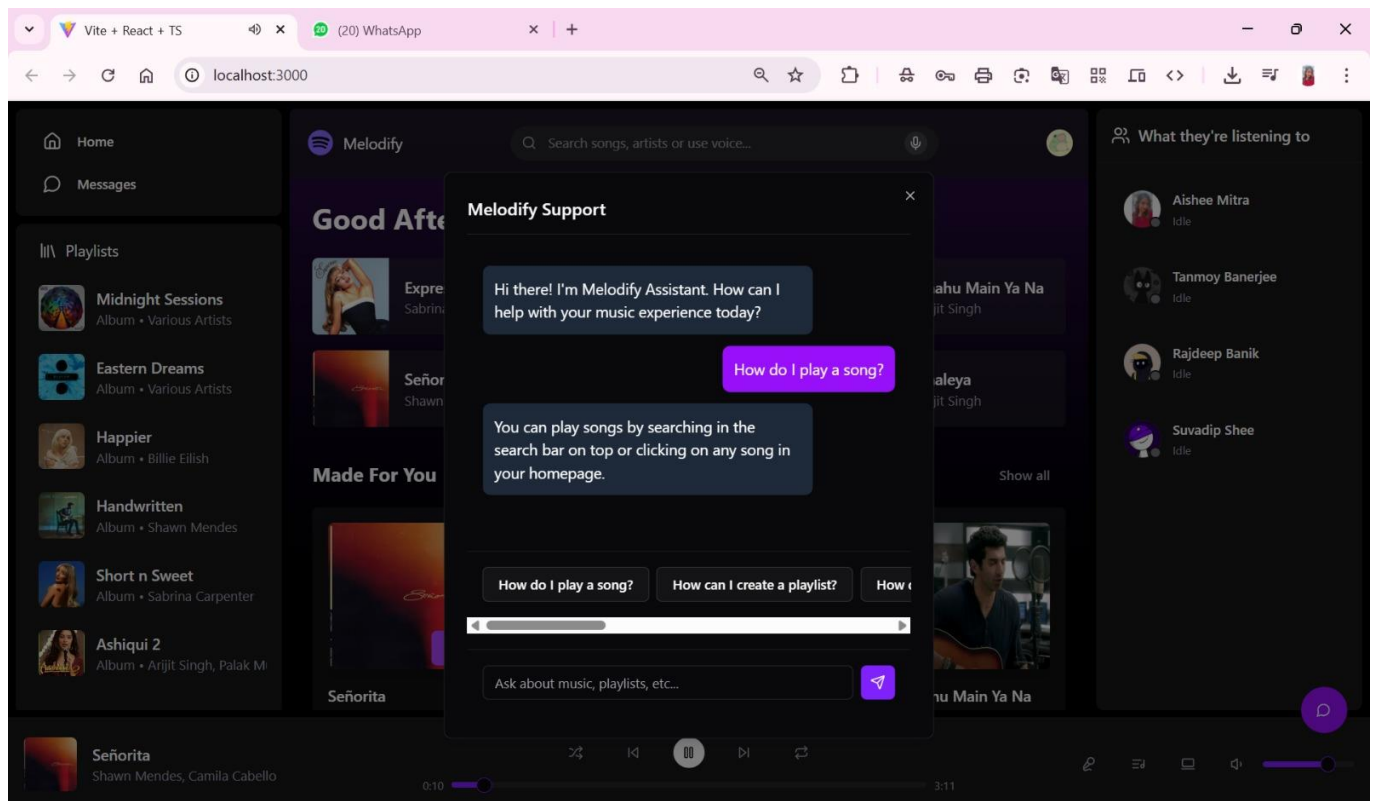
Real-Time Chat UI: -



Song Search Bar: -



Customer support chatbot : -



Chatbot code:-

```
import { Button } from "@components/ui/button";
import { Dialog, DialogContent, DialogHeader } from "@components/ui/dialog";
import { Input } from "@components/ui/input";
import { ScrollArea } from "@components/ui/scroll-area";
import { Send, MessageCircle } from "lucide-react";
import { useState, useRef, useEffect } from "react";
```

```
export const Chatbot = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [messages, setMessages] = useState<Array<{
    id: string;
    text: string;
    sender: 'user' | 'bot';
  }>>([
    {
      id: 'welcome',
      text: "Hi there! I'm Melodify Assistant. How can I help with your music experience today?",
      sender: 'bot'
    }
  ]);
  const [input, setInput] = useState("");
  const messagesEndRef = useRef<HTMLDivElement>(null);
  const [awaitingIssueDetail, setAwaitingIssueDetail] = useState(false);
```

```
const suggestedPrompts = [
  "How do I play a song?",
  "How can I create a playlist?",
  "How do I search for music?",
  "How do I download songs?",
  "What's the sound quality?",
  "Hi",
  "I'm facing an issue",
  "Why can't I play songs without signing in?"
];
```

```
const handleSend = (customText?: string) => {
  const messageText = customText ?? input;
  if (!messageText.trim()) return;
```

```

const userMessage = {
  id: Date.now().toString(),
  text: messageText,
  sender: 'user' as const
};
setMessages(prev => [...prev, userMessage]);
setInput("");

setTimeout(() => {
  const botResponse = {
    id: Date.now().toString(),
    text: getBotResponse(messageText),
    sender: 'bot' as const
  };
  setMessages(prev => [...prev, botResponse]);

  // Reset awaiting state
  setAwaitingIssueDetail(false);
}, 1000);
};

const getBotResponse = (userInput: string) => {
  const input = userInput.toLowerCase();

  if (awaitingIssueDetail) {
    return "Thanks for reaching out! For more specific help, mail us at  
melodify.support@gmail.com";
  }

  if (
    input.includes("why can't i play songs without signing in") ||
    input.includes("play songs without signing in")
  ) {
    return "To ensure a personalized and secure experience, song playback requires  
users to be signed in. Please log in to enjoy uninterrupted music!";
  }

  if (/playlist|collection|library/.test(input)) {
    return "If you have a premium subscription then you can create and manage  
playlists in the 'Your Library' section. Just click the '+' button to start a new  
playlist!";
  }
}

```

```
if (/^bplay(?:!list)\b|\bsong\b|\bmusic\b/.test(input)) {
  const responses = [
    "You can play songs by searching in the search bar on top or clicking on any song in your homepage.",
    "Having trouble playing a song? Try refreshing the page or checking your internet connection.",
  ];
  return responses[Math.floor(Math.random() * responses.length)];
}

if (/search|find/.test(input)) {
  return "Use the search bar at the top to find songs, artists, or albums. You can search by title, artist name, or even lyrics!";
}

if (/quality|bitrate|sound/.test(input)) {
  return "Melodify streams at the highest possible quality based on your connection.";
}

if (/discover|new music|recommend/.test(input)) {
  return "Check out our 'Made For You' and 'Trending' sections for personalized music recommendations based on your listening habits!";
}

if (/offline|download/.test(input)) {
  return "For offline listening, you'll need a premium subscription. This lets you download songs to your device.";
}

if (/hi|hello|hey/.test(input)) {
  return "Hello! Ready to explore some great music today?";
}

if (/problem|issue|error|bug/.test(input)) {
  setAwaitingIssueDetail(true);
  return "I'm sorry you're experiencing issues. Our team constantly works to improve Melodify. Could you describe what's happening in detail?";
}

return "Thanks for reaching out! For more specific help, mail us at melodify.support@gmail.com";
};
```

```

useEffect(() => {
  messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
}, [messages]);

return (
  <>
    <Button
      onClick={() => setIsOpen(true)}
      className="fixed bottom-20 right-6 rounded-full w-12 h-12 p-0 bg-purple-600
hover:bg-purple-700 z-50"
    >
      <MessageCircle className="h-5 w-5" />
    </Button>

    <Dialog open={isOpen} onOpenChange={setIsOpen}>
      <DialogContent className="max-w-md h-[80vh] flex flex-col">
        <DialogHeader className="border-b pb-3">
          <h3 className="text-lg font-semibold">Melodify Support</h3>
        </DialogHeader>

        <ScrollArea className="flex-1 p-4 overflow-y-auto scrollbar-thin scrollbar-
thumb-purple-500 scrollbar-track-gray-200">
          <div className="space-y-3">
            {messages.map((message) => (
              <div
                key={message.id}
                className={`flex ${message.sender === 'user' ? 'justify-end' : 'justify-
start`} `}
              >
                <div
                  className={`max-w-[80%] rounded-lg p-3 ${
                    message.sender === 'user'
                      ? 'bg-purple-600 text-white'
                      : 'bg-gray-800 text-white'
                  } `}
                >
                  {message.text}
                </div>
              </div>
            ))}
          <div ref={messagesEndRef} />
        </div>
      </DialogContent>
    </Dialog>
  </>
)

```



```

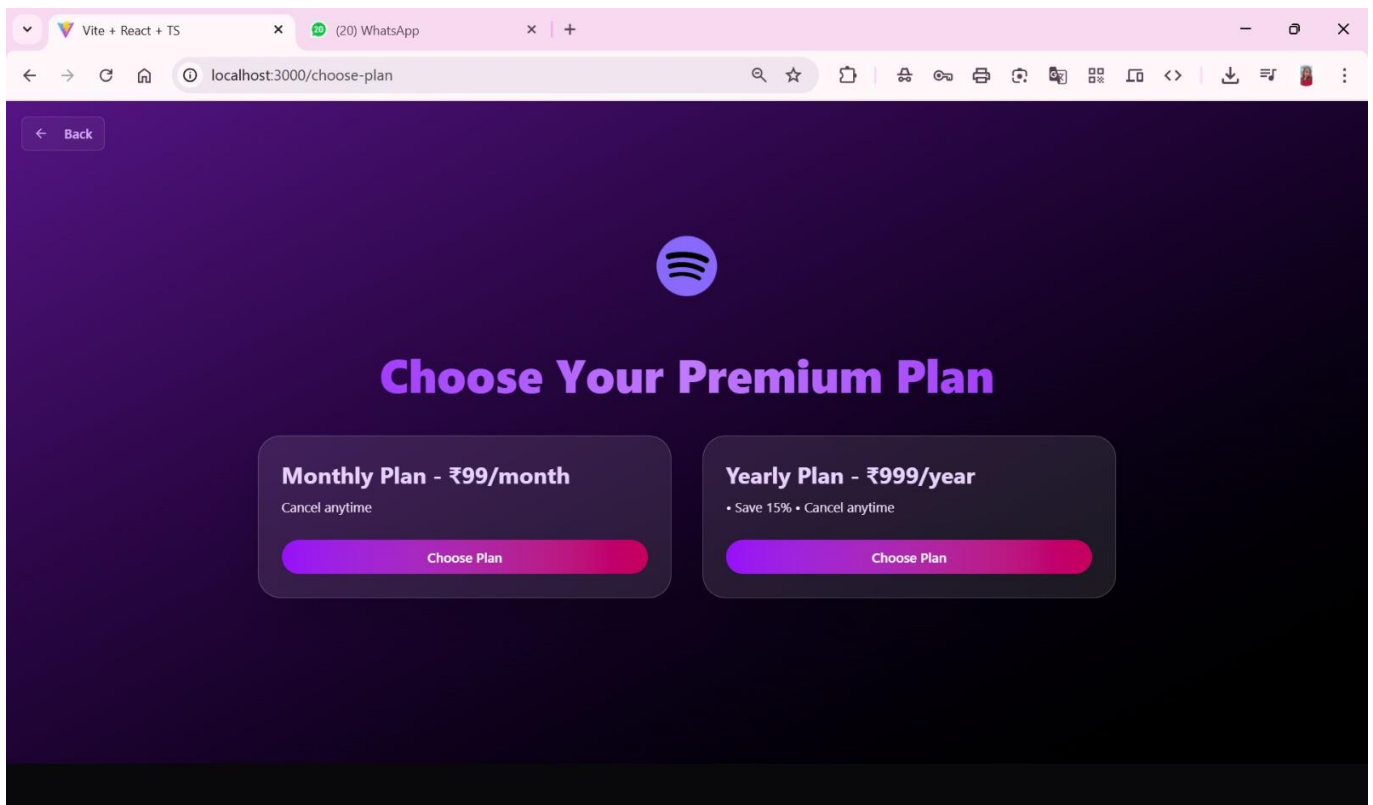
</ScrollArea>

{ /* Suggested Prompts */}
<div className="p-4 border-t border-b overflow-x-auto whitespace-nowrap
flex gap-2">
  {suggestedPrompts.map((prompt, index) => (
    <Button
      key={index}
      variant="outline"
      className="text-sm"
      onClick={() => handleSend(prompt)}
    >
      {prompt}
    </Button>
  ))}
</div>

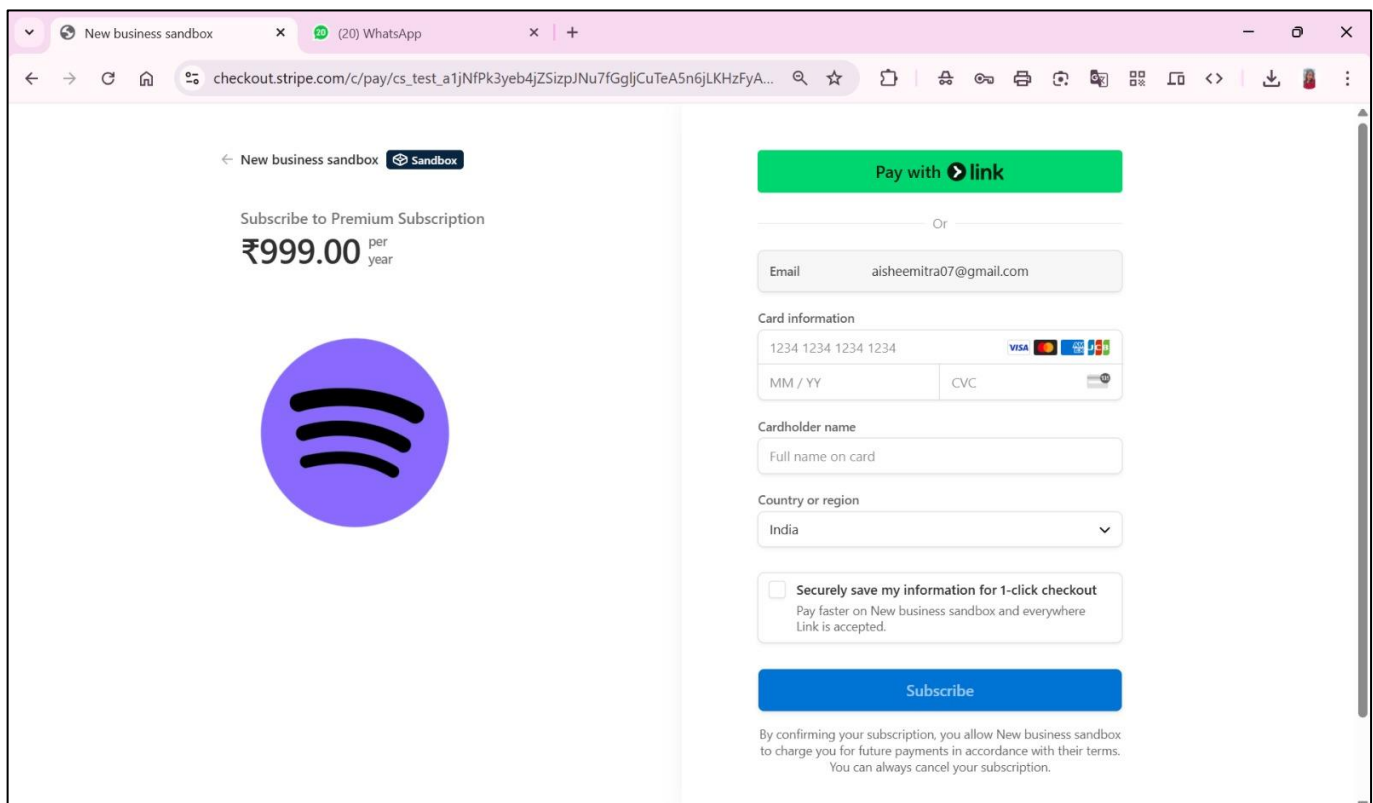
{ /* Chat Input */}
<div className="p-4 border-t flex gap-2">
  <Input
    value={input}
    onChange={(e) => setInput(e.target.value)}
    placeholder="Ask about music, playlists, etc..."
    onKeyDown={(e) => e.key === 'Enter' && handleSend()}
  />
  <Button onClick={() => handleSend()} size="icon">
    <Send className="h-4 w-4" />
  </Button>
</div>
</DialogContent>
</Dialog>
</>
);
};

```

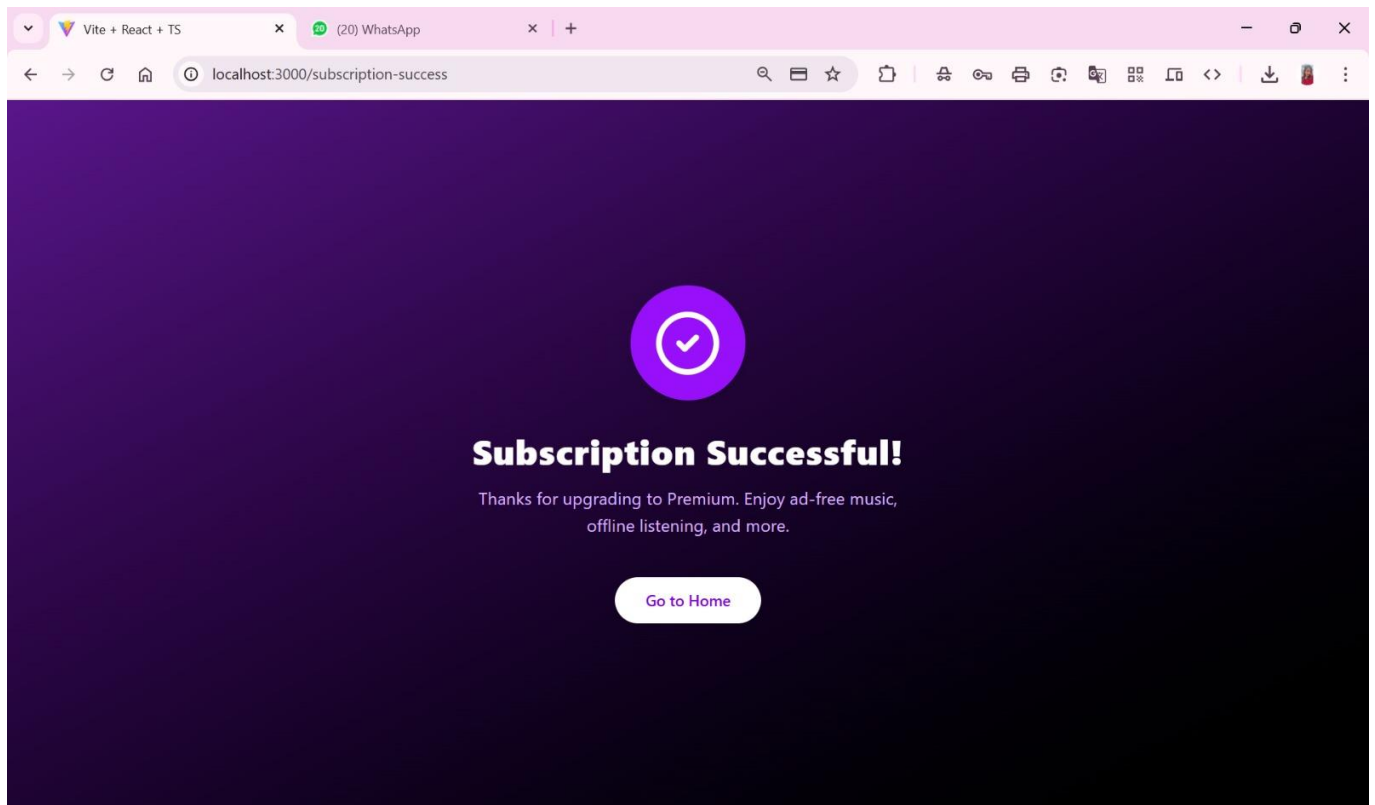
Choose Subscription Plan



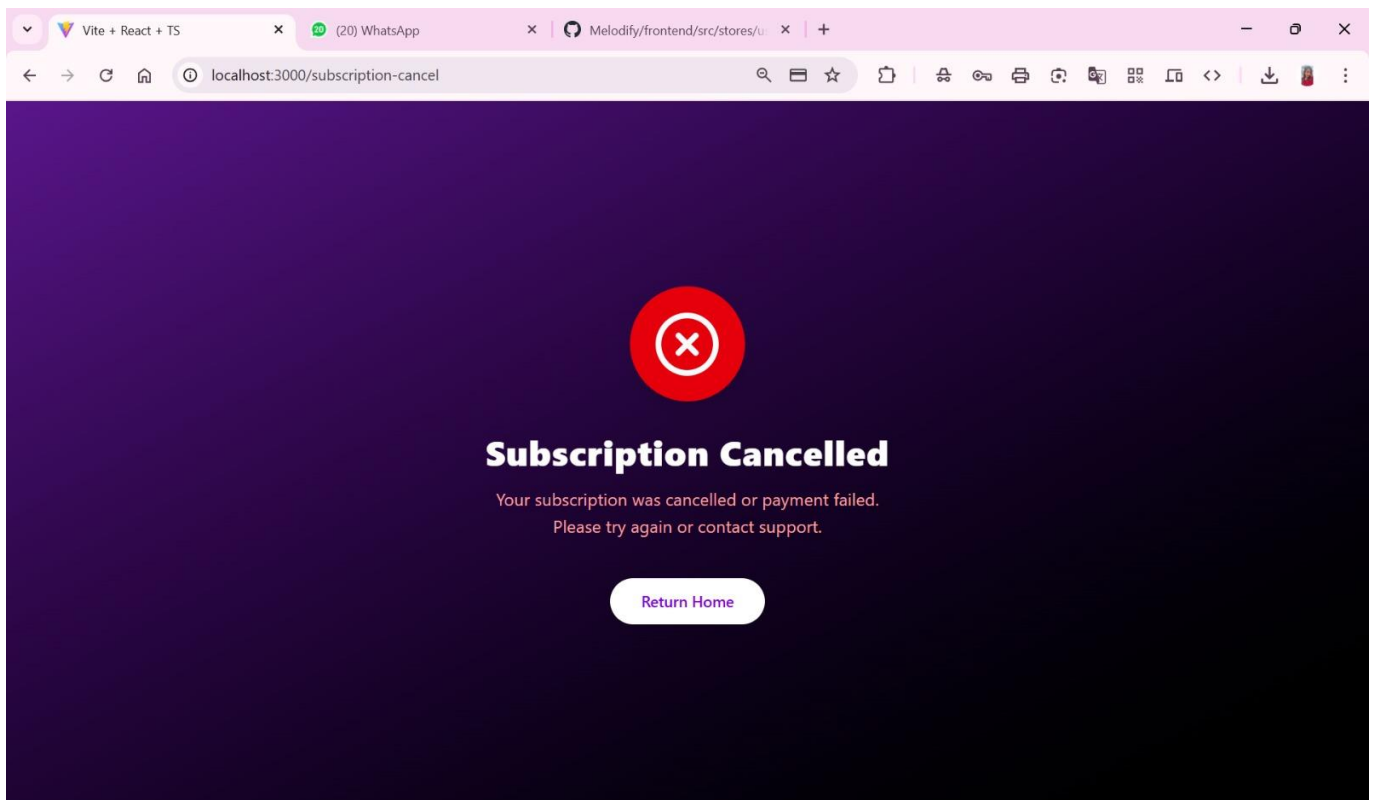
Stripe Payment Gateway



Subscription Payment successful



Subscription cancelled / Payment failed



Database (Mongo DB)

Overview

DATABASE

Clusters

SERVICES

Atlas Search

Stream Processing

Triggers

Migration

Data Federation

SECURITY

Quickstart

Backup

Database Access

Network Access

Advanced

Goto

AISHEE'S ORG - 2025-04-09 > MELODIFY > DATABASES

ClusterO

VERSION 8.0.8

REGION AWS Mumbai (ap-south-1)

OverviewReal TimeMetricsCollectionsAtlas SearchQuery InsightsPerformance AdvisorOnline ArchiveCmd Lin

DATABASES: 1COLLECTIONS: 4

+ Create Database

Search Namespaces

melodify_db

albums

messages

songs

users

LOGICAL DATA SIZE: 7.96KB

STORAGE SIZE: 144KB

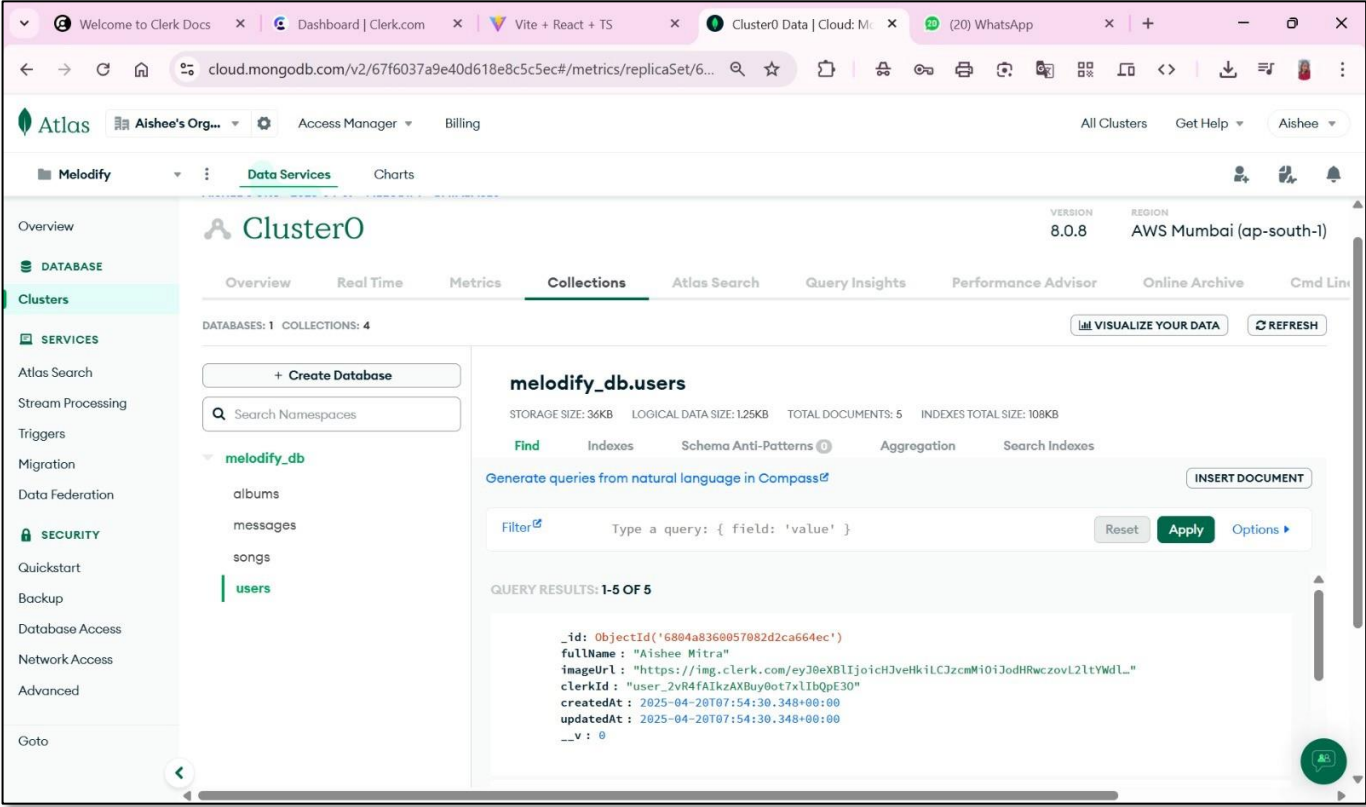
INDEX SIZE: 216KB

TOTAL COLLECTIONS: 4

CREATE COLLECTION

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
albums	5	1.2KB	247B	36KB	1	36KB	36KB
messages	13	2.42KB	191B	36KB	1	36KB	36KB
songs	13	3.09KB	244B	36KB	1	36KB	36KB
users	5	1.25KB	256B	36KB	3	108KB	36KB

Users-Table: -



Testing: -

Testing of the Melodify application was carried out thoroughly by team member **Rajdeep Kangsha Banik**. A combination of Unit Testing, Black Box Testing, and White Box Testing was used to verify the correctness, usability, and reliability of each component in the system. The table below outlines the key test scenarios, their classifications, expected results, and outcomes.

Test Case ID	Feature Tested	Testing Type	Test Scenario	Expected Result	Status
TC01	Music Playback	Black Box Testing	Play/pause a track from the interface	Music starts/stops as expected	✓ Passed
TC02	Volume Control	Black Box Testing	Adjust the volume slider	Volume changes in real time	✓ Passed
TC03	Navigation Controls	Black Box Testing	Use next/previous buttons while playing music	Navigates to the correct track	✓ Passed
TC04	Admin Dashboard	White Box Testing	Create a new album or song	Item is stored and displayed correctly	✓ Passed
TC05	Real-Time Chat	Black & White Box Testing	Send messages between two users	Messages are received instantly	✓ Passed
TC06	User Presence	White Box Testing	Log in/out from different devices	Status updates in real time	✓ Passed
TC07	Authentication (Clerk)	Black Box Testing	Perform login, logout, and sign-up	Auth flow works smoothly and securely	✓ Passed
TC08	Media Upload (Cloudinary)	White Box Testing	Upload album artwork	Image is stored and rendered correctly	✓ Passed
TC09	Analytics	White Box Testing	Visit analytics dashboard after user activity	Data displays updated metrics	✓ Passed

TC10	Error Handling	Black Box Testing	Try accessing unauthorized routes	Appropriate error or redirection is shown	✓ Passed
TC11	UI Responsiveness	Black Box Testing	Load the app on various screen sizes	UI adapts responsively to each device	✓ Passed
TC12	Voice search	Black Box Testing	Speak a song name into the search bar	Matching song/artist appears in results	✓ Passed
TC13	Customer Support Chatbot	Black Box Testing	Ask a help-related question	Bot responds accurately and promptly	✓ Passed
TC14	Stripe Subscription Payment	Black Box Testing	Purchase premium plan via Stripe	Payment succeeds, and premium access is granted	✓ Passed

Future Scope and Enhancements: -

While Melodify currently offers a robust set of features comparable to modern music streaming platforms, there are several opportunities to expand its functionality and improve user experience in future iterations. These enhancements can further solidify its potential as a scalable, production-ready application.

Mobile Application Development

To increase accessibility and user reach, a mobile version of Melodify can be developed using **React Native** or **Flutter**, enabling seamless music streaming on Android and iOS devices.

Playlist and Favourites System

Adding the ability for users to create playlists, save favourite tracks, and personalize their listening experience as premium features that would significantly boost user engagement and retention.

Music Recommendations

Integration with AI or third-party APIs (e.g., Spotify API or Last.fm) could enable personalized music recommendations based on user behaviour, history, and preferences.

Enhanced User Profiles

User profile customization with avatars, bios, themes, and follower systems could foster a more social, community-driven platform.

Mobile Push Notifications

Adding push notifications for real-time updates, new releases, or social interactions would improve user re-engagement on both web and mobile platforms.

Multi-language Support

Introducing internationalization (i18n) and multi-language support would make Melodify accessible to a broader global audience.

Advanced Admin Analytics

Extending the analytics dashboard with real-time charts, user trends, and system performance metrics could help admins better manage the platform and content.

Conclusion: -

The development of **Melodify** demonstrates the successful implementation of a full-stack web application using modern technologies and best practices. By leveraging the **MERN stack**, along with tools like **Clerk**, **Socket.io**, **Cloudinary**, **Stripe**, **Web Speech API**, and **shadcn/ui**, the project delivers a **feature-rich and interactive music streaming experience**.

From music playback and real-time chat to admin content management, analytics, **search with voice input**, and a **customer support chatbot**, Melodify replicates and extends core functionalities of platforms like Spotify in a scalable, modular architecture. The integration of **Stripe as a secure and seamless payment gateway** further enables premium features, showcasing the system's readiness for real-world monetization.

This project not only highlights the technical feasibility of building a real-time, media-rich application but also emphasizes the importance of **user experience**, **responsiveness**, **accessibility**, and **system design**. While the current version provides a strong foundation, there is ample scope for future enhancements such as **playlist support**, **AI-driven recommendation engines**, and **mobile application development**.

Overall, **Melodify** stands as a testament to the effective use of full-stack development for creating **dynamic, scalable, and engaging web applications**.

Bibliography: -

- 📌 React.js Documentation - <https://reactjs.org/>
- 📌 Vite Documentation - <https://vitejs.dev/>
- 📌 Node.js Documentation - <https://nodejs.org/>
- 📌 Express.js Documentation - <https://expressjs.com/>
- 📌 MongoDB Atlas Documentation - <https://www.mongodb.com/cloud/atlas>
- 📌 Clerk Authentication Docs - <https://clerk.dev/docs>
- 📌 Socket.io Documentation - <https://socket.io/docs>
- 📌 Cloudinary API Documentation - <https://cloudinary.com/documentation>
- 📌 shadcn/ui Components - <https://ui.shadcn.com/>
- 📌 Stripe Documentation - <https://stripe.com/docs>
- 📌 Web Speech API (for voice search) - https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
- 📌 GitHub - For version control and collaboration
- 📌 Stack Overflow - <https://stackoverflow.com/> (for issue resolution and community support).