

PODCAST CONTENT MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

NAVEEN S
(Reg. No: 24MCR073)

SURENDER VG
(Reg. No: 24MCR112)

THIRU SELVAM P
(Reg. No: 24MCR118)

*in partial fulfillment of the requirements
for the award of the degree*

of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER APPLICATIONS



KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI, ERODE – 638 060

DECEMBER 2024

**DEPARTMENT OF COMPUTER APPLICATIONS
KONGU ENGINEERING COLLEGE**

(Autonomous)

PERUNDURAI, ERODE – 638 060

DECEMBER 2024

BONAFIDE CERTIFICATE

This is to certify that the project report entitled “**PODCAST CONTENT MANAGEMENT SYSTEM**” is the bonafide record of project work done by **NAVEEN S (Reg. No: 24MCR073), SURENDER V G (Reg. No: 24MCR112), THIRU SELVAM P (Reg. No: 24MCR118)** in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications of Anna University, Chennai during the year 2024-2025.

SUPERVISOR

HEAD OF THE DEPARTMENT

(Signature with seal)

Date:

Submitted for the End semester viva voce examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We affirm that the project report entitled “**PODCAST CONTENT MANAGEMENT SYSTEM**” being submitted in partial fulfilment of the requirements for the award of Master of Computer Applications is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidates.

Date:

NAVEEN S
(Reg. No: 24MCR073)

SURENDER V G
(Reg. No: 24MCR112)

THIRU SELVAM P
(Reg. No: 24MCR118)

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Date:

Name and Signature of the supervisor
(Ms.S.HEMALATHA)

ABSTRACT

In the existing podcast management systems, users face several limitations that hinder their overall experience. These systems often provide only simple login and registration functionality, typically restricted to email and password authentication. Many platforms impose limitations on listening to audio files, restricting access to premium content or full episodes. Additionally, recommendation algorithms are basic, often relying on trending podcasts or broad genres selected by the user, which fail to cater to individual preferences. This leaves listeners struggling to discover new podcasts or relevant episodes, diminishing their engagement and satisfaction.

The proposed Podcast Content Management System aims to address these shortcomings by introducing a more robust and user-centric platform. The system offers options for social media login, such as Google authentication, to streamline user registration and access. Listeners can enjoy unrestricted access to audio content, including free downloads of episodes. The platform will feature real-time highlighting of popular episodes and advanced personalization tools to help users discover content aligned with their interests. Enhanced browsing features and tailored recommendations ensure a seamless and engaging user experience. This project is also intended to serve as an educational foundation, showcasing modern web technologies and methodologies.

The proposed system offers significant advantages over existing solutions. It eliminates barriers to accessing and downloading podcast content, fostering greater user satisfaction. The incorporation of social media login simplifies onboarding, while personalization features improve content discovery and engagement. Real-time updates on popular episodes keep users informed about trending content, creating a dynamic and interactive platform. Overall, this system enhances user experience by combining accessibility, convenience, and personalization, making it a comprehensive solution for podcast enthusiasts.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.**, and other philanthropic trust members of Kongu Vellalar Institute of Technology Trust for having provided with necessary resources to complete this project. We are always grateful to our beloved visionary Principal **Dr.V.BALUSAMY BE(Hons)., MTech., PhD** and thank him for his motivation and moral support.

We convey our gratitude and heartfelt thanks to our Head of the Department **Dr.A.TAMILARASI M.Sc., M.Phil., Ph.D., M.Tech.**, Department of Computer Applications, Kongu Engineering College for her perfect guidance and support that made this work to be completed successfully.

We also like to express our gratitude and sincere thanks to our Project Coordinator and Supervisor **Ms.S.HEMALATHA MCA**, Assistant Professor(Sr.G), Department of Computer Applications, Kongu Engineering college who have motivated us in all aspects for completing the project in scheduled time.

We owe a great deal of gratitude to our parents for helping us to overwhelm in all proceedings. We bow our heart and head with heartfelt thanks to all those who thought us their warm services to succeed and achieve our work.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	iv
	ACKNOWLEDGEMENT	v
	LIST OF TABLES	viii
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 ABOUT THE PROJECT	1
	1.2 EXISTING SYSTEM	1
	1.3 DRAWBACKS OF EXISTING SYSTEM	2
	1.3 PROPOSED SYSTEM	2
	1.4 ADVANTAGES OF THE PROPOSED SYSTEM	2
2	SYSTEM ANALYSIS	3
	2.1 IDENTIFICATION OF NEED	3
	2.2 FEASIBILITY STUDY	3
	2.2.1 Technical Feasibility	3
	2.2.2 Operational Feasibility	4
	2.2.3 Economical Feasibility	4
	2.3 SOFTWARE REQUIREMENT SPECIFICATION	5
	2.3.1 Hardware Requirements	5
	2.3.2 Software Requirements	5
3	SYSTEM DESIGN	9
	3.1 MODULE DESCRIPTION	9
	3.2 DATA FLOW DIAGRAM	12
	3.3 DATABASE DESIGN	14

	3.4 CLASS DIAGRAM	17
	3.5 INPUT DESIGN	18
	3.6 OUTPUT DESIGN	19
4	IMPLEMENTATION	22
	4.1 CODE DESCRIPTION	22
	4.2 STANDARDIZATION OF THE CODING	22
	4.3 ERROR HANDLING	23
5	TESTING AND RESULTS	25
	5.1 TESTING	25
	5.1.1 Unit Testing	25
	5.1.2 Integration Testing	27
	5.1.3 Validation Testing	28
6	CONCLUSION AND FUTURE ENHANCEMENT	29
	6.1 CONCLUSION	29
	6.2 FUTURE ENHANCEMENT	29
	APPENDICES	30
	A. SAMPLE CODING	30
	B. SCREENSHOTS	37
	REFERENCES	43

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
3.1	Users	14
3.2	Podcasts	15
3.3	Episodes	16

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
3.1	Dataflow Diagram Level 0	12
3.2	Dataflow Diagram Level 1	13
3.3	Class Diagram	17
B.1	Login Page	37
B.2	Sign Up page	38
B.3	Home Page	39
B.4	Search Podcast	39
B.5	Upload Podcast	40
B.6	Episode List	41
B.7	Listener Profile	41
B.8	User Data in Data Base	42
B.9	Podcast Data in Data Base	42

LIST OF ABBREVIATIONS

JS	Java Script
DB	Data Base
DFD	Data Flow Diagram
API	Application Programming Interface
UI	User Interface

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

In the digital age, the demand for podcasts has grown exponentially as a convenient and engaging medium for sharing information, telling stories, and discussing diverse topics. The Podcast Content Management System is a user-friendly platform that empowers creators to upload, organize, and manage digital audio content while enabling listeners to stream and download podcasts effortlessly. By offering an intuitive interface and robust features, the PCMS bridges the gap between podcast creators and audiences, making podcast discovery and consumption more engaging and accessible.

This project delivers two core functionalities: efficient content management for creators and dynamic content delivery for listeners. Creators can upload audio files, add metadata such as titles, descriptions, and tags, and organize episodes into curated sections for better discoverability. Listeners benefit from a streamlined browsing, searching, streaming, and downloading experience, tailored to their interests. Features like user interactions, including likes and comments, foster an active and engaged podcasting community.

For creators, the project consolidates fragmented tasks like content management, promotion, and audience interaction into a centralized hub, saving time and enhancing productivity. For listeners, the platform simplifies podcast discovery with categorized listings, advanced search filters, and personalized recommendations, ensuring they quickly find content that suits their preferences.

In conclusion, the Podcast Content Management System revolutionizes podcast hosting, management, and consumption by providing a unified platform for creators and listeners. It enhances content delivery, discoverability, and audience engagement, fostering a stronger and more connected podcast community.

1.2 EXISTING SYSTEM

The existing systems for podcast management and distribution are fragmented, requiring creators to use multiple platforms for hosting, categorizing, and streaming content, which increases operational complexity. These platforms often lack centralized solutions, making it difficult for small-scale podcasters to efficiently manage their episodes and engage with

their audience. For listeners, discoverability and interaction features are limited, and content is rarely categorized effectively, resulting in a less personalized experience. Additionally, media storage solutions often impose file size limits or high costs, while offline access capabilities are typically absent.

1.3 DRAWBACK OF EXISTING SYSTEM

- Simple login and registration functionality, often limited to email and password.
- Most of the websites puts limit to listen audio files.
- Basic recommendations based on trending podcasts or user-selected genres.
- Listeners struggles to find new podcasts or relevant episodes.

1.4 PROPOSED SYSTEM

The proposed Podcast Content Management System (PCMS) aims to provide a centralized, user-friendly platform for both podcast creators and listeners. It will allow creators to easily upload, manage, and organize their episodes with metadata, using **ReactJS** for a responsive front-end and **MongoDB** for efficient storage of metadata. **Firebase** will handle the secure and scalable storage of audio files. Listeners will benefit from a seamless browsing experience, categorized podcast listings, search features, personalized recommendations, and the ability to interact with content through likes, comments, and discussions.

1.5 ADVANTAGES OF THE PROPOSED SYSTEM

- Options for social media login (like Google).
- Allows listeners to listen and download the audios for free.
- Highlight popular episodes in real-time.
- Taking the project as educational basis.

SUMMARY

The existing system for podcast management is fragmented, requiring creators to use multiple tools, which is time-consuming and inefficient. Listeners face challenges in discovering relevant content due to the vast number of podcasts available. The detailed information regarding the system analysis will be presented in the following chapter 2.

CHAPTER 2

SYSTEM ANALYSIS

2.1 IDENTIFICATION OF NEED

The podcast industry is growing rapidly, but creators and listeners face significant challenges. Podcasters often struggle with fragmented tools for managing, promoting, and distributing their content, leading to inefficiencies and missed opportunities. Similarly, listeners find it difficult to discover relevant podcasts amidst the overwhelming volume of content available online. To address these issues, the Podcast Content Management System (PCMS) is proposed. This system provides a centralized, user-friendly platform where creators can seamlessly manage their podcasts and listeners can easily discover and enjoy content tailored to their preferences. By simplifying podcast management and enhancing audience engagement, the PCMS bridges the gap between creators and listeners, fostering a more connected and thriving podcast community.

2.2 FEASIBILITY STUDY

A feasibility study is a complete analysis that is conducted to evaluate the practicality and viability of a proposed project. The prime goal of a feasibility study is to determine whether the project is worth pursuing and if it can be successfully implemented within the defined constraints. Each structure has to be thought of in the developing of the project, as it has to serve the end user in friendly manner. Three considerations involved in feasibility are

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

2.2.1 Technical Feasibility

The Technical Feasibility assess the technical characteristics of the project, including the availability of technology, infrastructure, and expertise required for successful implementation. The proposed system will be platform independent since it is being coded in ReactJS, Nodejs and MongoDB with the help of visual studio code to implement, which have several developing tools for developing the web application.

The cost and benefits analysis may be concluded that computerized system is favorable in today's fast-moving world. The assessment of technical feasibility must be based on online design of the system requirements in terms of input, output, files, programs and procedures.

This web application allows podcast creators to upload and manage unlimited episodes while enabling listeners to discover, stream, and download a vast range of podcasts. The platform is designed to accommodate a large number of creators and users, making it scalable and widely accessible. Additionally, the system ensures an intuitive and user-friendly experience, enhancing usability for both creators and listeners.

2.2.2 Operational Feasibility

The developed podcast content management system is designed to be user-friendly for both content creators (podcast producers) and listeners, allowing them to easily understand and navigate its purpose. Operational feasibility assesses whether the new system will be effectively adopted and used once developed and implemented, or if resistance from users will limit the potential benefits of the application. Additionally, it considers the level of acceptance among users.

The proposed system addresses challenges faced by the current podcast management tools and offers better integration with the daily operations of the organization. Operational feasibility analysis primarily focuses on how the new system will influence the organization's structure and processes. The system has been designed to seamlessly integrate with the existing workflow, ensuring ease of use for all stakeholders. Furthermore, the system allows users to efficiently manage and view podcast content, making it accessible and convenient for listeners.

2.2.3 Economic Feasibility

This application is developed by Reactjs, Nodejs and MongoDB, which is one of the opensource language and it can be used anywhere easily. The economic feasibility is carried out to check the economic impact that the system will have on the organization. Thus, the developed system is well within the budget and this was achieved because most of the technologies used are freely available.

Web-based applications have attracted numerous industries and customers globally, driving many companies to prioritize web-based investments for their business growth. This

application is developed using ReactJS, an open-source JavaScript library widely recognized for building user interfaces and enhancing frontend development, providing an intuitive experience for listeners.

2.3 SOFTWARE REQUIREMENT SPECIFICATION

A declaration that describes the capability that a system needs in order to satisfy the user's requirements is known as a system requirement. The system requirements for specific machines, software or business operations are general. Taking it all the way down to the hardware and coding that operates the software. System requirements are the most efficient way to address user needs while lowering implementation costs.

2.3.1 Hardware Requirements

The hardware for the system is selected considering the factors such as CPU processing speed, memory access, peripheral channel access speed, printed speed; seek time& relational data of hard disk and communication speed etc.

Processor	:	INTEL CORE i5
RAM	:	4 GB
Hard disk	:	512 GB
Keyboard	:	108 Key Standard Keyboard
Mouse	:	Scroll Mouse

2.3.2 Software Requirements

The software for the project is selected considering the factors such as working frontend environment, flexibility in the coding language, database knowledge of enhances in backend technology etc.

Scripting Language	:	HTML, CSS, ReactJS
Back End	:	MongoDB
Tools	:	Visual studio code

Front End

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. The primary goal of ReactJS is to create User Interfaces (UI) that accelerate apps. It makes use of virtual DOM (JavaScript object), which enhances the app's performance. Faster than the standard DOM is the JavaScript virtual DOM. ReactJS integrates with different frameworks and can be used on the client and server sides. It makes use of component and data patterns to make larger apps more readable and maintainable. These Components can be nested with other components to allow complex applications to be built of simple building blocks.

ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time. NodeJS and NPM are the platforms need to develop any ReactJS application. React-Bootstrap replaces the Bootstrap JavaScript.

ReactJS primary goal is to create User Interfaces (UI) that increase the speed of apps. It makes use of virtual DOM (JavaScript object), which enhances the app's performance. Faster than the standard DOM is the JavaScript virtual DOM. ReactJS integrates with different frameworks and can be used on the client and server sides. It makes use of component and data patterns to make larger apps more readable and maintainable.

Features of React JS

There are unique features are available on React because that it is widely popular.

- **Virtual DOM:** There is virtual DOM, which is basically a simplified version of the real DOM. Therefore, there is one object in React Virtual DOM for every object that is present in the real DOM. Although it is identical, it is unable to alter the document's design in any way. Virtual DOM manipulation is quick compared to DOM manipulation since no images are generated on the screen.
- **Component:** One of React fundamental building blocks is the component. Or, to put it another way, every application you create with React will be built up of what are known as components. Using components makes creating user interfaces considerably simpler.

- **Performance:** React.js use JSX, which is faster compared to normal JavaScript and HTML. Virtual DOM is a less time taking procedure to update web pages content.

Back End

Node.js is an open-source, cross-platform runtime environment for JavaScript code execution outside of a browser that is based on Chrome's V8 JavaScript engine. You must keep in mind that NodeJS is not a programming language or a framework. It offers a cross-platform runtime environment for event-driven, non-blocking (asynchronous) I/O and extremely scalable server-side JavaScript applications.

The majority of folks are perplexed and realise it's a programming language or framework. Building back-end services like APIs, web applications, and mobile applications frequently uses Node.js. Large corporations like Uber, Netflix, Walmart, etc. use it in their manufacturing.

Features of Node.js

- **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- **Single threaded:** Node.js follows a single threaded model with event looping.
- **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
- **License:** Node.js is released under the MIT license.

MongoDB

MongoDB is a popular open-source NoSQL database management system that is designed to handle large volumes of unstructured or semi-structured data. It is classified as a document- oriented database, and its architecture allows for flexible and scalable data storage and retrieval. MongoDB stores data in flexible, JSON-like BSON (Binary JSON) documents, allowing for a wide variety of data structures within a single database. Unlike traditional relational databases, MongoDB is schema-less, meaning you can insert records without first defining the structure of the document. This flexibility is particularly useful in scenarios where the data structure evolves over time and it uses internal memory (RAM) for

storing the working set of data, resulting in high performance for read and write operations. Additionally, it supports indexing to optimize query performance.

It is designed to scale horizontally, enabling it to handle increased load by adding more servers to a distributed database system. This horizontal scaling approach is often referred to as sharding.

MongoDB provides a powerful query language that supports a wide range of queries, including field, range, and regular expression queries. It also supports aggregation frameworks for more complex data manipulations. It supports indexing on fields, which can significantly improve query performance. Indexes are automatically created for the primary key and can be defined for other fields as needed. MongoDB can automatically partition and distribute data across multiple servers, making it well-suited for large-scale, distributed systems. Sharding enhances horizontal scalability and data distribution.

Features of MongoDB

- **Document Model:** which means that data is stored as documents, and documents are grouped in collections. Developers can focus on the data they need to store and process, rather than worrying about how to split the data across different rigid tables.
- **Sharding:** Sharding is the process of splitting larger datasets across multiple distributed instances, or “shards.”
- **Replication:** When your data only resides in a single server, it is exposed to multiple potential points of failure, such as a server crash, service interruptions, or even good old hardware failure. Any of these events would make accessing your data nearly impossible.
- **Time Series Data:** Time series data is most commonly generated by a device, such as a sensor, that records data over time. The data is stored in a collection of documents, each of which contains a timestamp and a value. MongoDB provides a number of features to help you manage time series data.

SUMMARY

The system requirements are fully explained in detail for each piece of hardware and software that was used to meet the user and system requirements. The above have explained the features of ReactJS which plays main role in the project. The detailed information regarding the system design will be presented in the following chapter 3.

CHAPTER 3

SYSTEM ANALYSIS

3.1 MODULE DESCRIPTION

A module description provides detailed information about the module and its supported components, which is accessible in different manners.

The project contains the following module:

- Dashboard
- Upload Podcast
- Episode List
- Listener Profile

Dashboard:

The dashboard serves as the central hub for both creators and administrators, designed to deliver a seamless and intuitive user experience. It acts as the command center for managing and analyzing all aspects of the podcast content management system. With a focus on accessibility and efficiency, the dashboard provides a clear and comprehensive overview of platform activities, empowering users to make informed decisions and take prompt actions.

For creators, the dashboard offers essential insights into podcast performance, including total plays, unique listeners, and listener retention metrics. These analytics help creators understand audience behavior, identify trends, and optimize content strategies. Recent uploads are prominently displayed, allowing creators to quickly review or update their content. Additionally, the dashboard integrates tools for scheduling new episodes, managing podcast series, and engaging with the audience through comments or feedback mechanisms.

For administrators, the dashboard becomes a powerful tool for oversight and management. Key system metrics, such as the total number of podcasts on the platform, active users, and system health checks, are readily available. Administrators can monitor platform-wide engagement, track user activity, and identify any issues requiring attention. The dashboard also includes administrative controls for managing user roles, reviewing flagged content, and maintaining platform integrity.

Upload Podcast:

The upload podcast feature is a cornerstone of the podcast content management system, designed to streamline the process of adding new episodes to the platform. Tailored specifically for podcast creators, this feature ensures that uploading content is not only simple but also efficient, allowing creators to focus on producing high-quality episodes without being burdened by technical complexities.

Creators can effortlessly upload audio files in various supported formats, ensuring compatibility and flexibility. Alongside the audio upload, the system enables creators to input essential metadata such as episode titles, descriptions, categories, and release dates. These details help enhance discoverability and provide listeners with relevant context for each episode. To ensure a seamless workflow, the upload process is optimized with a user-friendly interface that guides creators through each step, minimizing errors and saving time. Progress indicators and error notifications are integrated to keep users informed throughout the upload. Once an episode is uploaded, creators can preview the metadata, make edits, and schedule publication or release it immediately.

Additionally, the system incorporates robust backend capabilities for secure storage and quick retrieval of audio files. This ensures that episodes are safely stored while remaining easily accessible for future updates or edits. By simplifying the content upload process and integrating powerful management tools, this feature empowers creators to consistently publish episodes with confidence and ease.

Episode List:

The episode list feature is an essential component of the podcast content management system, providing a comprehensive and organized view of all episodes for a specific podcast. This feature is designed to cater to the needs of both creators and listeners, ensuring seamless navigation and efficient content management.

For listeners, the episode list serves as a convenient interface to browse through all available episodes. Episodes are displayed in a clear and structured format, organized by release date, episode number, or custom sorting preferences. Each episode entry includes essential details such as the title, release date, duration, and a brief description, helping users quickly identify content of interest. Listeners can easily stream episodes directly from the platform or download them for offline access, making it a flexible tool for enjoying podcasts on the go. Filters and search options further enhance usability by enabling listeners to find

specific episodes or topics effortlessly.

For creators, the episode list acts as a centralized management tool, allowing them to oversee and update their podcast content in one organized view. Creators can review metadata, monitor episode performance metrics, and make adjustments as needed. Options to edit descriptions, update titles, or change the release order are easily accessible, ensuring that the content remains accurate and engaging. Bulk actions, such as archiving or reordering episodes, streamline content management for series with extensive episode libraries.

By combining a user-friendly design with robust functionality, the episode list feature bridges the gap between creators and listeners, fostering engagement and maintaining an organized podcast experience.

Listener Profile:

The listener profile feature is a cornerstone of the podcast content management system, offering a tailored and engaging experience for every user. This feature empowers listeners to create personalized profiles where they can manage their podcast preferences, track their activity, and interact with the podcast community.

With the listener profile, users can easily keep track of their favorite podcasts and saved episodes, creating a curated library that reflects their unique interests. The profile provides a detailed listening history, allowing users to revisit past episodes and continue listening from where they left off. This enhances convenience and ensures a seamless experience across devices.

Listeners can manage their notification preferences directly from their profile, opting to receive alerts for new episodes, updates from their favorite creators, or community interactions. Personalized recommendations based on listening habits and preferences are prominently displayed, helping users discover new content that aligns with their tastes.

The listener profile also fosters community engagement by enabling interaction with other users. Features like liking episodes, leaving comments, and participating in discussions create a dynamic and interactive environment. This sense of community not only enhances the user experience but also encourages more active participation within the platform.

Designed with user-centricity in mind, the listener profile feature integrates advanced tracking and recommendation algorithms while maintaining a clean, user-friendly interface. By consolidating all these functionalities into one accessible space, the listener profile becomes an essential tool for personalizing and enriching the podcast experience.

3.2 DATAFLOW DIAGRAM

The Data Flow Diagram provides information about the inputs and outputs of each entity and process itself.

LEVEL 0

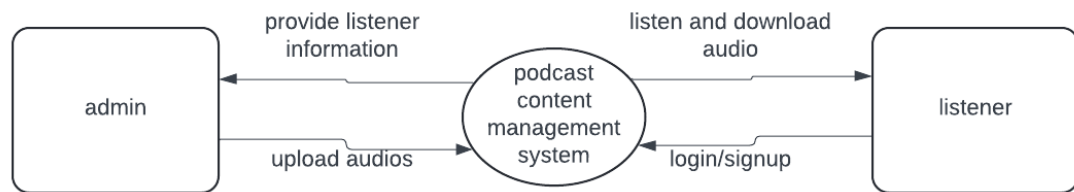


Figure 3.1 Dataflow Diagram Level 0

In Figure 3.1 The Level 0 Data Flow Diagram (DFD) for the podcast content management system provides a high-level overview of the interactions between the system, administrators (admins), and listeners. Admins upload podcast episodes, manage metadata, and access listener-related information like analytics on episode performance, helping them make informed decisions about content strategies.

The system supports content optimization and categorization, ensuring the right content reaches the right audience. Listeners interact with the system by signing up or logging in, enabling secure access to features such as favorite podcasts, saved episodes, and tailored recommendations. Once authenticated, they can browse, stream, or download podcasts. The system tracks listener activity to improve recommendations and engagement while offering social sharing features for enhanced community interaction.

The podcast content management system serves as the central hub, efficiently handling data exchanges, storing content, and managing user profiles, ensuring admins can focus on content creation while listeners enjoy seamless access to their favorite podcasts. It also provides robust security measures to safeguard user data and content.

LEVEL 1

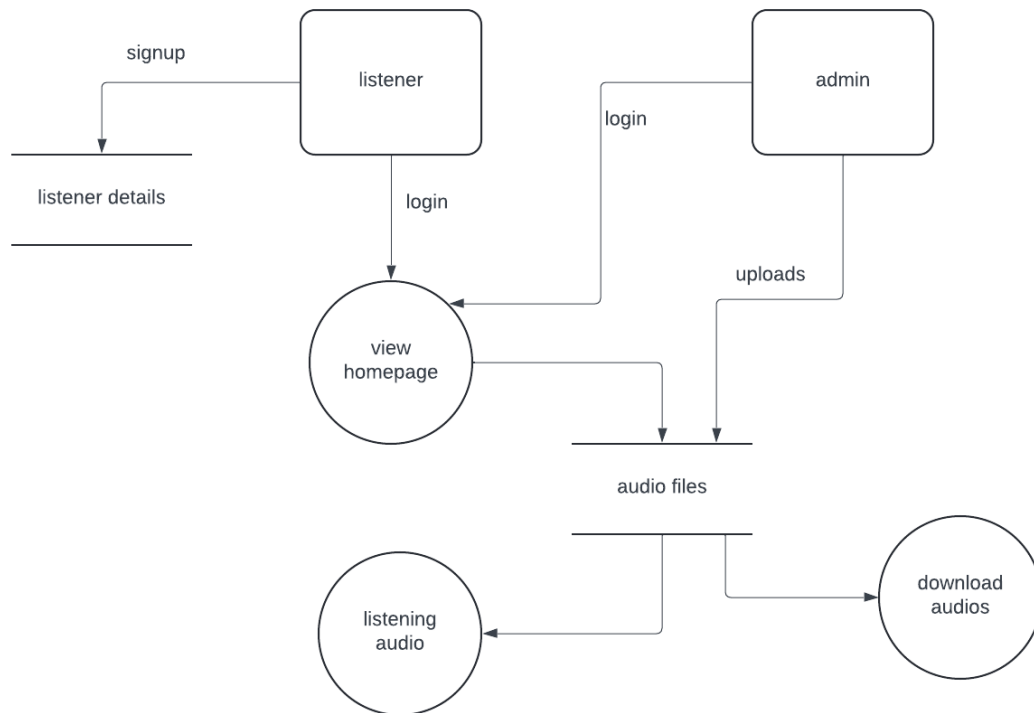


Figure 3.2 Dataflow Diagram Level 1

In Figure 3.2 The podcast content management system operates as follows: New users sign up and provide necessary details, while existing users log in to their accounts. Upon successful authentication, users are presented with the homepage, showcasing a curated list of available podcasts. Users can explore the content, selecting specific podcasts or audio files to begin listening. On the administrative side, authorized users can access the admin panel to manage the platform. This includes uploading new audio files, organizing content, and overseeing user accounts.

Uploaded audio files are securely stored and organized within the system. Users have the option to download audio files for offline listening, subject to any applicable restrictions or permissions. The Level 1 Data Flow Diagram (DFD) expands on the system's core processes, showing data flow between users, the admin panel, content storage, and recommendation systems. It further details interactions like metadata management, analytics tracking, and content categorization. This helps refine content delivery and enhance user experience through personalization and targeted recommendations.

3.3.DATABASE DESIGN

Table 3.1 Users

Column Name	Data Type
_id	int
name	string
email	string
password	string
img	string
googlesignin	boolean
podcasts	array
favorites	array
createdAt	date
updatedAt	date

In Table 3.1 represents the User Collection in the podcast website database, storing key details about listeners and their activities. The _id column serves as a unique identifier for each user, ensuring each record is distinct and easily retrievable. The name, email, and password columns store basic user information securely, with passwords encrypted for protection to prevent unauthorized access. The googlesignin column tracks whether users registered via Google, facilitating seamless sign-in options and simplifying user management. The podcasts and favorites columns store arrays for uploaded podcasts and liked episodes, ensuring personalized content management and improving user experience by enabling easy access to preferred content.

The createdAt and updatedAt columns maintain timestamps for user account creation and updates, allowing the system to track user activity and manage data synchronization efficiently. Additionally, the database structure supports future scalability by easily integrating new user attributes or features, such as subscription plans or listening history, to further enhance the platform's capabilities. The use of indices on key columns like email and _id helps improve query performance and data retrieval speed, ensuring the platform can scale efficiently as the number of users grows. This design ensures efficient user management and enhanced functionality for the platform, offering a secure, personalized, and scalable experience for all users.

Table 3.2 Podcast

Column Name	Data Type
_id	int
name	string
description	string
thumbnail	string
creator	int
tags	array
type	string
category	string
views	int
episodes	array
createdAt	date
updatedAt	date

In Table 3.2 represents the Podcast Collection in the podcast website database, storing essential details about each podcast. The _id column serves as a unique identifier for every podcast entry, ensuring data integrity and easy retrieval. The name, description, and thumbnail columns provide information and visuals for the podcast, contributing to a rich user experience by displaying key details and engaging imagery. The creator column links the podcast to its user ID, ensuring traceability and proper attribution of content to the creator. The tags, type, and category columns classify the podcast for better discoverability, enabling users to search for podcasts based on relevant keywords and genres.

The views column tracks the number of times a podcast has been accessed, providing valuable data for content analysis and decisions. The episodes column stores an array of associated episode IDs, allowing the system to link individual episodes to the corresponding podcast, ensuring seamless navigation between episodes. Additionally, the podcast collection can be extended to include metadata such as language, duration, and release frequency, further improving categorization and user preferences. The database design also supports integration with analytics tools, offering insights into podcast performance and listener engagement. This structured approach ensures scalability, security, and efficient management of podcast data.

Table 3.3 Episodes

Column Name	Data Type
_id	int
name	String
description	string
thumbnail	string
creator	int
type	string
duration	string
file	string
createdAt	date
updatedAt	date

In Table 3.3 represents the Episode Collection in the podcast website database, storing detailed information about individual podcast episodes. The _id column acts as a unique identifier for each episode, ensuring data consistency and easy retrieval. The name and description columns provide the title and a brief overview of the episode's content, offering listeners an informative preview before they start listening. The thumbnail column stores an image associated with the episode for visual representation, enhancing the platform's user interface and making content more visually appealing. The creator column links the episode to its creator's user ID, ensuring proper attribution and traceability of content. The type and duration columns specify the format and length of the episode, offering users context on what to expect, such as whether it's an interview, narrative, or Q&A format, and how long the episode lasts. The file column stores the path to the audio file, making it easy for the system to retrieve and play the episode.

The createdAt and updatedAt columns track timestamps for episode creation and updates, aiding in content management and providing insights into the episode's timeline. Additionally, the episode table can include fields for metadata such as language, release date, and rating, allowing for advanced search and filtering options. It can also support integration with analytics tools to track listener behavior, engagement, and episode popularity. This comprehensive design ensures that episodes are well-organized and efficiently managed within the system, contributing to a seamless user experience.

3.4 CLASS DIAGRAM

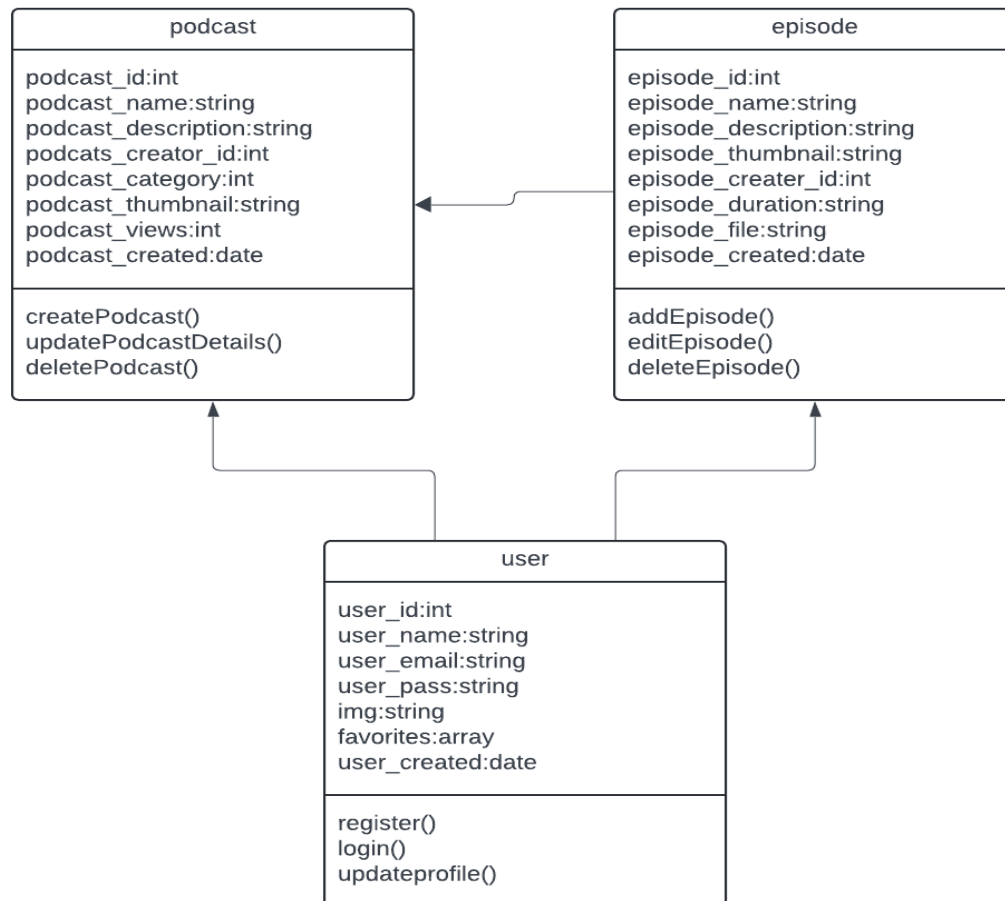


Figure 3.3 Class Diagram

Figure 3.3 represents the class diagram illustrates the design of a Podcast Content Management System with three main classes: **podcast**, **episode**, and **user**. The **podcast** class manages attributes like `podcast_id`, `podcast_name`, `podcast_description`, and includes methods for creating, updating, and deleting podcasts. The **episode** class represents individual episodes with attributes such as `episode_id`, `episode_name`, `episode_description`, and `episode_file`, along with methods for adding, editing, and deleting episodes. The **user** class stores user information like `user_id`, `user_name`, `user_email`, and `favorites`, with methods for registration, login, and profile updates. Relationships indicate that a **podcast** can have multiple **episodes**, and **users** interact with both **podcasts** and **episodes**. This structure ensures efficient content management and user interaction.

3.5 INPUT DESIGN

Input design is the process of converting user-originated inputs to a computer understandable format. Input design is one of the most expensive phase of the operation of computerized system and is often the major problem of a system. A large number of problems with a system can usually be tracked back to fault input design and method.

Every moment of input design should be analyzed and designed with utmost care. The decisions made during the input design are the project gives the low time consumption to make sensitive application made simple. Thus, the developed system is well within the budget. This was achieved because most of the technologies used are freely available. Only the customized product had to be purchased. In the project, the forms are designed with easy-to-use option. The coding is being done such that proper validation are made to get the prefect input. No error inputs are accepted.

Login form

This a user login form, the user can login with their registered mail id and password to access the application. After completing the registration, the user should give their correct register details else the user cannot able to login.

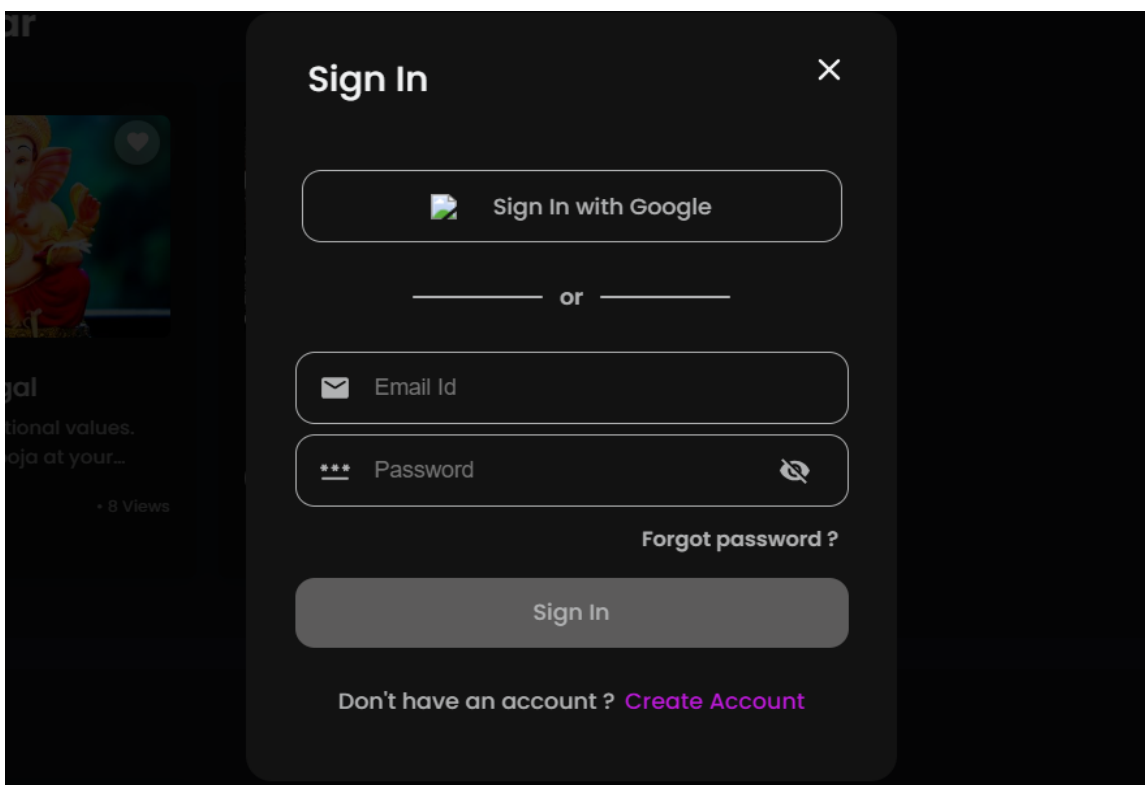


Figure 3.1 User Login page

The Figure 3.1 is user login form. Login form has mail id and password. The username should be in proper email format. If the fields are not filled, form will not be submitted. Password must contain at least 8 digit or characters. If the form is submitted successfully the page redirected towards home page.

Search Podcast:

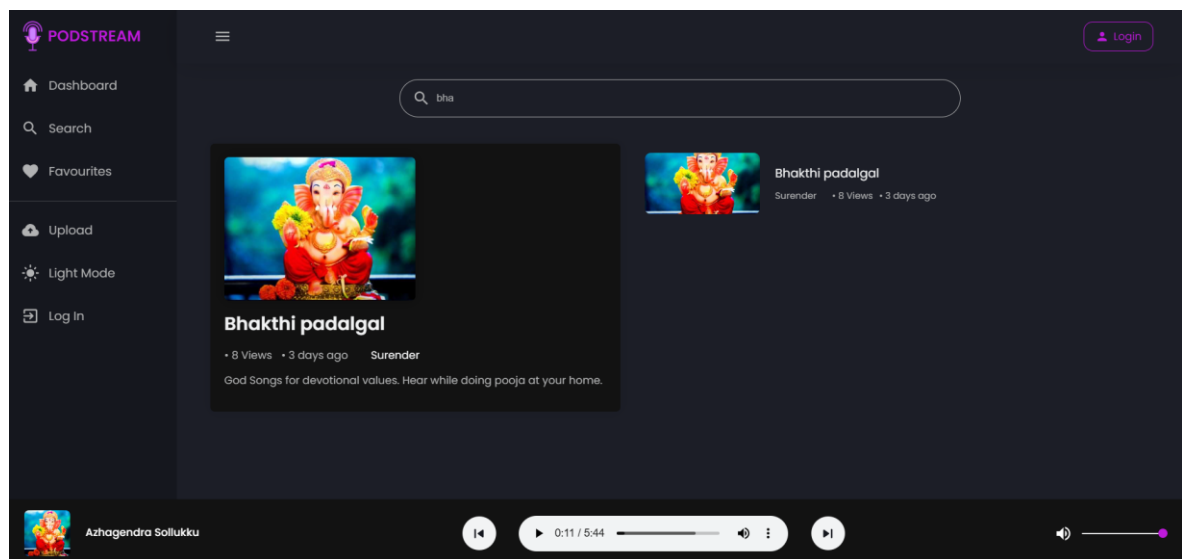


Figure 3.2 Search Podcast

The Figure 3.2 The Search Module in a Podcast Content Management System allows users to find podcasts quickly by entering keywords in a search bar. It retrieves relevant results based on titles, descriptions, or creators and displays them with thumbnails, titles, views, and upload dates. Users can refine searches using filters or sorting options like popularity or relevance. The module ensures efficient navigation with fast and accurate retrieval of content. This feature enhances user experience by simplifying podcast discovery and boosting platform engagement.

3.6 OUTPUT DESIGN

Output design generally refers to the results and information that are generated by the system for many end-users; it should be understandable with the enhanced format. Computer output is most important direct source of information to the user. Output design deals with form design. Efficient output design should improve the interfacing with user. The term output applies to any information produced by an information system in terms of data displayed. When analyst design system output, they identify the specific output that is needed

to meet the requirements of end user.

Previewing the output reports by the user is extremely important because the user is the ultimate judge of the quality of the output and in turn, the success of the system. When designing output, system analysis accomplishes more things like, to determine what applications, website or documents are blocked or allowed. The output is designed in such way that is attractive, convenient and informative.

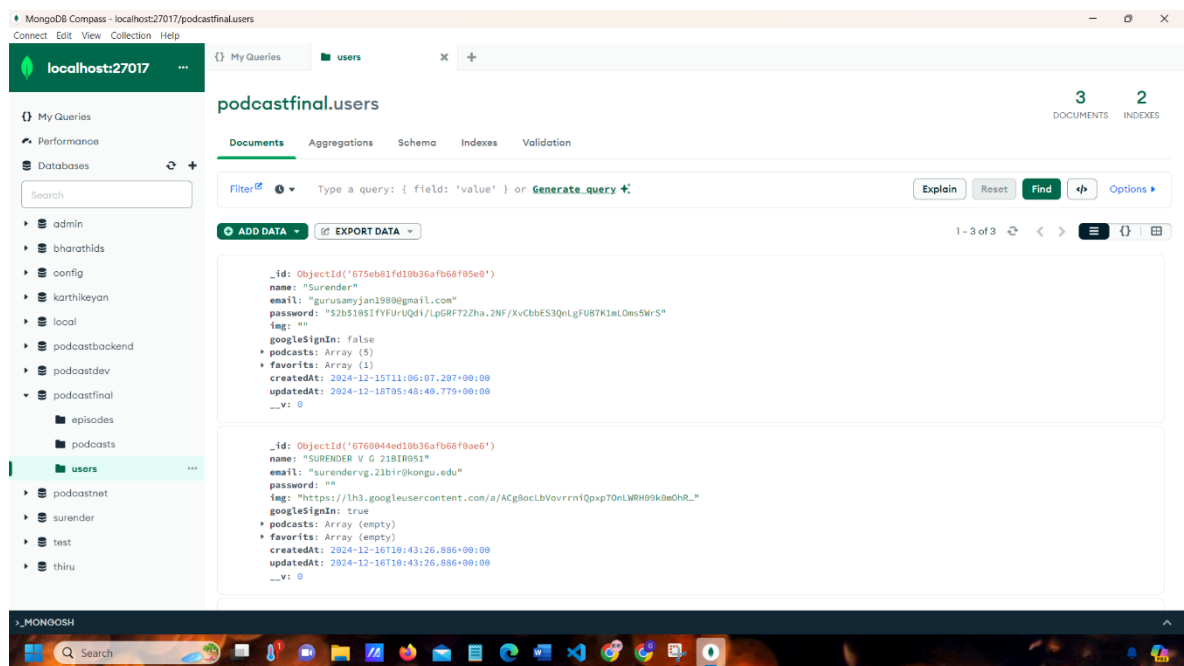


Figure 3.3 Storing in MongoDB

Figure 3.3 shows the information related to login and listener details, stored in MongoDB as documents within collections. In MongoDB, data is represented as flexible, schema-less documents, typically in BSON (Binary JSON) format, which allows for easy storage and retrieval of user-related information. Each user document includes key fields such as the user's name, email, password (encrypted), profile image, and preferences, which can be easily updated or queried. The documents are organized into collections, with each collection storing data of a similar type, such as user details or login history, ensuring efficient management and access. MongoDB's document-based structure enables quick retrieval of user data and seamless updates, offering high scalability and flexibility to accommodate changes in the user model, such as adding new attributes (e.g., subscription plans or user activity). This approach also ensures that listener details can be personalized and indexed for fast access, improving the overall user experience.

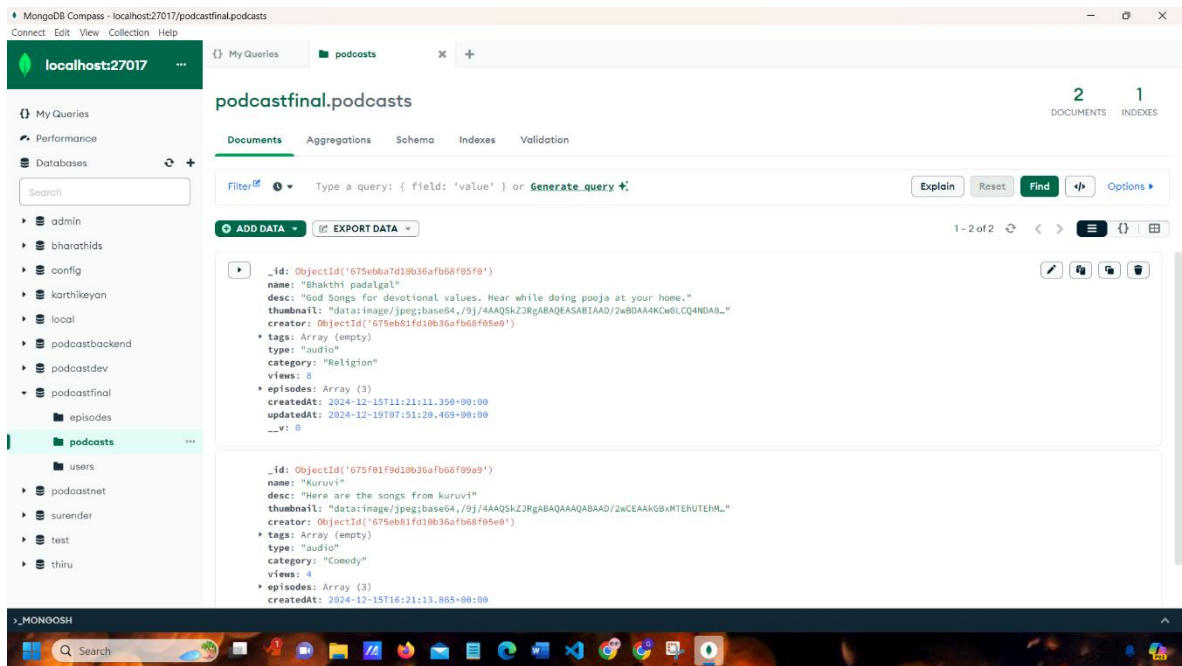


Figure 3.4 Podcast Details Stored In MongoDB

Figure 3.4 shows the information stored in MongoDB about uploaded podcasts, which are stored as documents within a podcast collection. Each podcast document includes essential details such as the podcast name, a cover image, an audio file link, and a description, allowing for efficient management and retrieval of content. The flexible schema of MongoDB allows the addition of other metadata, such as podcast categories, tags, creator information, and listener ratings, which can be used for enhanced search and filtering capabilities. The search module leverages these stored details to fetch and display relevant podcasts based on user queries, utilizing MongoDB's indexing and querying features for fast and accurate results. This structure ensures that podcast data is easily accessible and updatable, supporting the dynamic needs of the platform and improving the user experience by delivering personalized and targeted content. Additionally, the use of MongoDB's document-based storage allows for seamless handling of large volumes of podcast data, ensuring scalability as the platform grows.

SUMMARY

This chapter contains a detailed description of the modules. Login Page, how the search module works, how the user data is stored in the database are all mentioned in the module description. The Dataflow diagrams depicts the flow of the application from the user entry to the exit. It flows through all the processes.

CHAPTER 4

IMPLEMENTATION

4.1 CODE DESCRIPTION

Code description can be used to summarize code or to explain the programmer's intent. Good comments don't repeat the code or explain it. They clarify its intent. Comments are sometimes processed in various ways to generate documentation external to the source code itself by document generator or used for integration with systems and other kinds of external programming tools. I have chosen reactJS as front end and mongoDB as back end.

4.2 STANDARDIZATION OF THE CODING

Coding standards define a programming style. A coding standard does not usually concern itself with wrong or right in a more abstract sense. It is simply a set of rules and guidelines for the formatting of source code. The other common type of coding standard is the one used in or between development teams. Professional code performs a job in such a way that is easy to maintain and debug. All the coding standards are followed while creating this project. Coding standards become easier, the earlier you start. It is better to do a neat job than cleaning up after all is done. Every coder will have a unique pattern than he adheres to such a style might include the conventions he uses to name variables and functions and how he comments his work. When the said pattern and style is standardized, it pays off the effort well in the long.

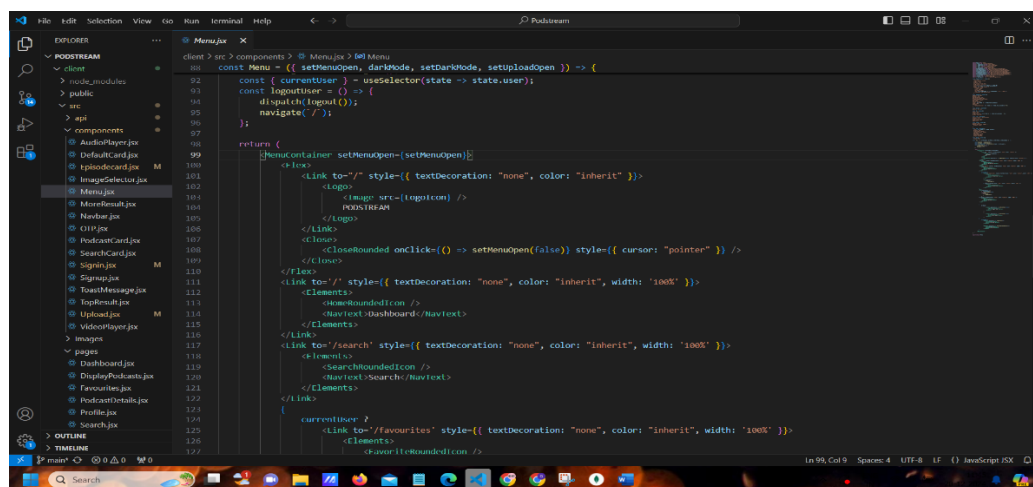


Figure 4.1 Standardization of the coding

In Figure 4.1 Elements like MenuContainer, Logo, Elements, NavText, etc., follow clear and descriptive naming conventions that make it easy to understand their purpose.

Reusable Components: The use of modular components (like Link, Image, and Close) enhances reusability and keeps the code DRY (Don't Repeat Yourself), allowing easy updates or changes.

Separation of Concerns: The code separates concerns by keeping layout (Flex, Logo, Elements) and behavior (setMenuOpen, onClick) distinct, which promotes clarity and easier maintenance.

4.3 ERROR HANDLING

Exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. In many cases there are many corner cases which must be checked during an execution but “if-else” can only handle the defined conditions. In if-else, conditions are manually generated based on the task.

An error is a serious problem than an application doesn't usually get pass without incident. Errors cause an application to crash, and ideally send an error message offering some suggestion to resolve the problem and return to a normal operating state, there is no way to deal with errors “live” or in production the only solution is to detect them via error monitoring and bug tracking and dispatch a developer or two to sort out the code.

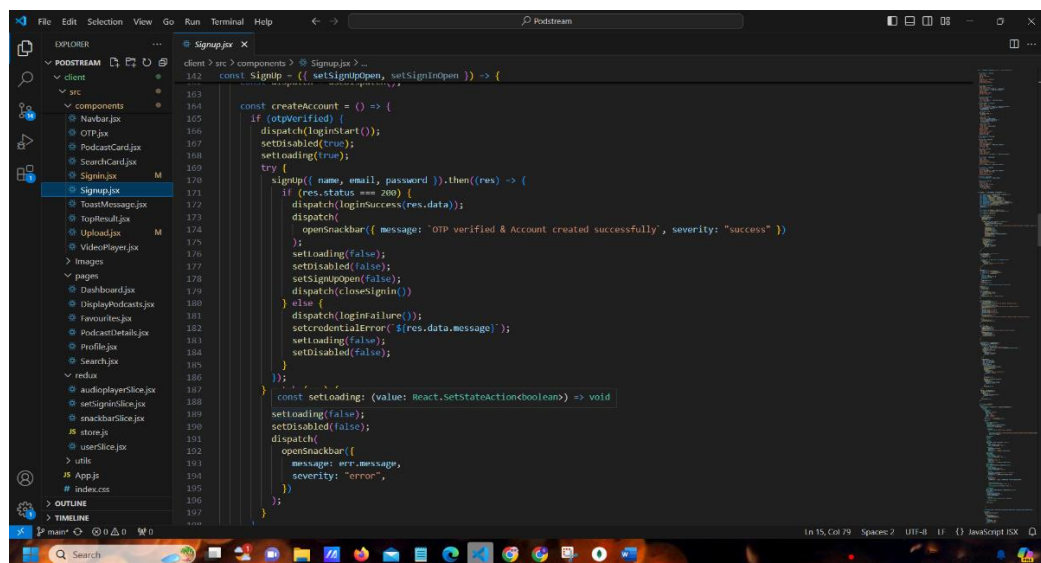


Figure 4.2 Error Handling

In Figure 4.2 implements error handling for OTP verification and account creation by first checking if `otpVerified` is true before proceeding. It dispatches `loginStart()` to signal the start of the signup process and disables the button to prevent multiple submissions. A try-catch block is used to handle any potential errors during the `signUp` process.

If the signup is successful (status 200), it dispatches `loginSuccess()` and shows a success message in a snackbar. If the signup fails (non-200 status), it dispatches `loginFailure()` and sets the error message. In case of an error in the try block, the catch section handles the issue by dispatching a failure action and displaying the error message in a snackbar. After each action, the loading and disabled states are reset to allow further interactions..

USER INTERFACE DESIGN

Input design is the process of converting user originated inputs to a computer understandable format. Input design is one of the most expensive phases of the operation of computerized system and is often the major problem of a system. A large number of problems with a system can usually be tracked back to fault input design and method. Every moment of input design should be analyzed and designed with utmost care. To provide cost effective method of input.

To achieve the highest possible level of accuracy. To ensure that the input is understood by the user. System analysis decide the following input design details like, what data to input, what medium to use, how the data should be arranged or coded, data items and transactions needing validations to detect errors and at last the dialogue to guide user in providing input. Input data of a system may not be necessarily is raw data captured in the system from scratch. These can also be the output of another system or subsystem.

SUMMARY

In this chapter shows explained that the purpose of a code description is to summarise the code or to clarify the programmer's intent. Good comments don't repeat or explain the code. A programming style is defined by coding standards. A coding standard isn't usually concerned with what's proper or bad in a broader sense. Exception handling is a method or process for dealing with and executing anomalous statements in code. Next Chapter 5 is shown about the Testing.

CHAPTER 5

TESTING

5.1 TESTING

Software testing serves as the final assessment of specifications, designs, and coding and is a crucial component of software quality assurance. The system is tested throughout the testing phase utilizing diverse test data. The preparation of test data is essential to the system testing process. The system under study is tested after the test data preparation. Once the source code is complete, relevant data structures should be documented. The finished project must go through testing and validation, when errors are explicitly targeted and attempted to be found.

The project developer is always in charge of testing each of the program's separate units, or modules. Developers frequently also perform integration testing, which is the testing phase that results in the creation of the complete program structure.

This project has undergone the following testing procedures to ensure its correctness

- Unit testing
- Integration Testing
- Validation Testing

5.1.1 Unit Testing

A testing strategy known as unit and integration testing has been used to check that the system behaves as expected. The testing strategy was based on the functionality and the requirements of the system. In unit checking out, we have to check the applications making up the device.

This enables, to stumble on mistakes in coding and common sense which might be contained with the module alone. The checking out became completed at some stage in programming level itself.

Test Case 1

Module : User Login

Input : Username and Password

Event : Button click

Output : Logged in successfully

Test 1

Module : User Login

Username : gurusamyjan1980@gmail.com

Password : Surender@123

Output : Logged in successfully

Event : Button click

Analysis : Username and Password has been verified

Test 2

Module : User Login

Username : gurusamyjan1980@gmail.com

Output : Enter the password

Event : Button click

Analysis : Username and Password has been checked and error shown

5.1.2 Integration Testing

Integration testing is done to test itself if the individual modules work together as one single unit. In integration testing, the individual modules that are to be integrated are available for testing. Thus, the manual test data that used to test the interfaces replaced by that which in generated automatically from the various modules. It can be used for testing how the module would actually interact with the proposed system. The modules are integrated and tested to reveal the problem interfaces.

Test Case 1

Module : User Login
 Input : Username and Password
 Output : Redirect Home page

Test 1

Module : User Login
 Username : gurusamyjan1980@gmail.com
 Password : Surender@123
 Output : Redirected to Home page
 Analysis : Username and Password has been verified

Test 2

Module : User Login
 Username : gurusamyjan198@gmail.com
 Password : Surender@123
 Output : Enter the correct username
 Analysis : Username and Password has been checked and error shown

5.1.3 Validation Testing

Verification and validation checking out are critical tests, which might be achieved earlier than the product has been surpassed over to the customer. This make sure, that the software program checking out lifestyles cycle begins off evolved early. The intention of each verification and validation is to make certain that the product is made in step with the necessities of the customer and does certainly fulfil the supposed purpose.

Test case 1

Module	: User Registration
Input	: Password
Output	: Password must be at least 8 characters, must contain at least one uppercase and special characters

Test 1

Module	: User Registration
Password	: Surender@123
Output	: Registered successfully
Analysis	: The given password is 8 characters and contains one uppercase and special characters. So, it is validated.

Test 2

Module	: User Registration
Password	: surender123
Output	: Password must contain at least one uppercase and special characters
Analysis	: The password with 8 characters and at least one uppercase and special characters can be validated.

SUMMARY

The preceding chapter discusses the many types of testing, including unit testing, integration testing and system testing. During the system evaluation following the completion of the source code, it is documented as associated data structures. The final project must go through testing and validation, which includes both subtle and overt attempts to find problems.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The project work entitled “**PODCAST CONTENT MANAGEMENT**” so far been tested and documented completely. The Podcast Content Management System (PCMS) provides an efficient, centralized platform for managing and distributing podcast content. By integrating technologies like ReactJS, MongoDB, and Firebase, the system ensures seamless functionality for both creators and listeners. It simplifies podcast uploads, categorization, and streaming, while offering interactive features like search, recommendations, and user engagement through comments and likes. The platform addresses the limitations of existing systems by offering a streamlined, user-friendly experience, making podcast management and discovery accessible and enjoyable for all listeners.

6.2 FUTURE ENHANCEMENT:

- **Live Streaming Support**

Live streaming support in a podcast management system allows creators to broadcast real-time audio or video content to their audience, enhancing engagement and interaction. It typically includes features like chat integration, live notifications, and viewer analytics. This functionality enables podcasters to offer dynamic, real-time experiences while maintaining seamless content delivery and audience participation.

- **Mobile App Development**

Mobile app development in a podcast management system enables users to access, manage, and listen to podcasts on-the-go, offering a seamless experience across devices. It includes features like episode downloading, push notifications, and personalized recommendations to enhance user engagement. The app also supports offline listening, making it convenient for users to enjoy content without a constant internet connection.

APPENDICES

A.SAMPLE CODING

app.js

```
import { ThemeProvider } from "styled-components";
import { useState, useEffect } from "react";
import { darkTheme, lightTheme } from '../utils/Themes.js'
import Signup from '../src/components/Signup.jsx';
import Signin from '../src/components/Signin.jsx';
import OTP from '../src/components/OTP.jsx'
import Navbar from '../src/components/Navbar.jsx';
import Menu from '../src/components/Menu.jsx';
import Dashboard from '../src/pages/Dashboard.jsx'
import ToastMessage from '../components/ToastMessage.jsx';
import Search from '../src/pages/Search.jsx';
import Favourites from '../src/pages/Favourites.jsx';
import Profile from '../src/pages/Profile.jsx';
import Upload from '../src/components/Upload.jsx';
import DisplayPodcasts from '../src/pages/DisplayPodcasts.jsx';
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import { useDispatch, useSelector } from "react-redux";
import styled from 'styled-components';
import AudioPlayer from "../components/AudioPlayer.jsx";
import VideoPlayer from "../components/VideoPlayer.jsx";
import PodcastDetails from "../pages/PodcastDetails.jsx";
import { closeSignin } from "../redux/setSigninSlice.jsx";

const Frame = styled.div`
  display: flex;
  flex-direction: column;
  flex: 3;
`;

const Podstream = styled.div`
  display: flex;
  flex-direction: row;
  width: 100%;
  height: 100vh;
  background: ${({ theme }) => theme.bgLight};
  overflow-y: hidden;
  overflow-x: hidden;
`;

function App() {

  const [darkMode, setDarkMode] = useState(true);
  const { open, message, severity } = useSelector((state) => state.snackbar);
```



```

const { openplayer,type, episode, podid, currenttime,index } = useSelector((state) =>
state.audioplayer);
const { opensi } = useSelector((state) => state.signin);
const [SignUpOpen, setSignUpOpen] = useState(false);
const [SignInOpen, setSignInOpen] = useState(false);
const [menuOpen, setMenuOpen] = useState(true);
const [uploadOpen, setUploadOpen] = useState(false);

const { currentUser } = useSelector(state => state.user);
const dispatch = useDispatch()
//set the menuOpen state to false if the screen size is less than 768px
useEffect(() => {
  const resize = () => {
    if (window.innerWidth < 1110) {
      setMenuOpen(false);
    } else {
      setMenuOpen(true);
    }
  }
  resize();
  window.addEventListener("resize", resize);
  return () => window.removeEventListener("resize", resize);
}, []);

useEffect(()=>{
  dispatch(
    closeSignin()
  )
},[])

return (

  <ThemeProvider theme={darkMode ? darkTheme : lightTheme}>

    <BrowserRouter>
      {opensi && <Signin setSignInOpen={setSignInOpen}
setSignUpOpen={setSignUpOpen} />}
      {SignUpOpen && <Signup setSignInOpen={setSignInOpen}
setSignUpOpen={setSignUpOpen} />}
      {uploadOpen && <Upload setUploadOpen={setUploadOpen} />}
      {openplayer && type === 'video' && <VideoPlayer episode={episode}
podid={podid} currenttime={currenttime} index={index}/>}
      {openplayer && type === 'audio' && <AudioPlayer episode={episode}
podid={podid} currenttime={currenttime} index={index}/>}
    <Podstream>
      {menuOpen && <Menu setMenuOpen={setMenuOpen} darkMode={darkMode}
setDarkMode={setDarkMode} setUploadOpen={setUploadOpen}
setSignInOpen={setSignInOpen}/>}
    <Frame>

```

```

        <Navbar menuOpen={menuOpen} setMenuOpen={setMenuOpen}
setSignInOpen={setSignInOpen} setSignUpOpen={setSignUpOpen} />
        <Routes>
            <Route path="/" exact element={<Dashboard setSignInOpen={setSignInOpen}/>} />
        />
        <Route path="/search" exact element={<Search />} />
        <Route path="/favourites" exact element={<Favourites />} />
        <Route path="/profile" exact element={<Profile />} />
        <Route path="/podcast/:id" exact element={<PodcastDetails />} />
        <Route path="/showpodcasts/:type" exact element={<DisplayPodcasts/>} />

        </Routes>
    </Frame>

    {open && <ToastMessage open={open} message={message} severity={severity}
/>}
    </Podstream>

    </BrowserRouter>

    </ThemeProvider>

    );
}

export default App;

```

Menu.js

```

import React from 'react'
import styled from 'styled-components'
import { useDispatch } from "react-redux";
import { useNavigate } from 'react-router-dom';
import { logout } from "../redux/userSlice";
import { Link } from 'react-router-dom'
import { useSelector } from 'react-redux';
import HomeRoundedIcon from '@mui/icons-material/HomeRounded';
import SearchRoundedIcon from '@mui/icons-material/SearchRounded';
import FavoriteRoundedIcon from '@mui/icons-material/FavoriteRounded';
import BackupRoundedIcon from '@mui/icons-material/BackupRounded';
import LightModeRoundedIcon from '@mui/icons-material/LightModeRounded';
import DarkModeRoundedIcon from '@mui/icons-material/DarkModeRounded';
import ExitToAppRoundedIcon from '@mui/icons-material/ExitToAppRounded';
import CloseRounded from '@mui/icons-material/CloseRounded';
import LogoIcon from '../Images/Logo.png'
import { openSignin } from '../redux/setSigninSlice';

const MenuContainer = styled.div`
    flex: 0.5;

```

```

flex-direction: column;
height: 100vh;
display: flex;
box-sizing: border-box;
align-items: flex-start;
background-color: ${({ theme }) => theme.bg};
color: ${({ theme }) => theme.text_primary};
@media (max-width: 1100px) {
  position: fixed;
  z-index: 1000;
  width: 100%;
  max-width: 250px;
  left: ${({ setMenuOpen }) => (setMenuOpen ? "0" : "-100%")};
  transition: 0.3s ease-in-out;
}
`;
const Elements = styled.div`
padding: 4px 16px;
display: flex;
flex-direction: row;
box-sizing: border-box;
justify-content: flex-start;
align-items: center;
gap: 12px;
cursor: pointer;
color: ${({ theme }) => theme.text_secondary};
width: 100%;
&:hover{
  background-color: ${({ theme }) => theme.text_secondary + 50};
}
`;
const NavText = styled.div`
padding: 12px 0px;
`;
const HR = styled.div`
width: 100%;
height: 1px;
background-color: ${({ theme }) => theme.text_secondary + 50};
margin: 10px 0px;
`;
const Flex = styled.div`
justify-content: space-between;
display: flex;
align-items: center;
padding: 0px 16px;
width: 86%;
`;
const Close = styled.div`
display: none;

```

```

@media (max-width: 1100px) {
  display: block;

}
`;
const Logo = styled.div`
  color: ${({ theme }) => theme.primary};
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 6px;
  font-weight: bold;
  font-size: 20px;
  margin: 16px 0px;
`;
const Image = styled.img`
  height: 40px;
`;
const Menu = ({ setMenuOpen, darkMode, setDarkMode, setUploadOpen }) => {

  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { currentUser } = useSelector(state => state.user);
  const logoutUser = () => {
    dispatch(logout());
    navigate(`/`);
  };

  return (

    <MenuContainer setMenuOpen={setMenuOpen}>
      <Flex>

        <Link to="/" style={{ textDecoration: "none", color: "inherit" }}>
          <Logo>
            <Image src={LogoIcon} />

            PODSTREAM
          </Logo>

        </Link>
        <Close>
          <CloseRounded onClick={() => setMenuOpen(false)} style={{ cursor:
"pointer" }} />
        </Close>

      </Flex>

      <Link to="/" style={{ textDecoration: "none", color: "inherit", width: '100%' }}>

```

```

        <Elements>
            <HomeRoundedIcon />
            <NavText>Dashboard</NavText>
        </Elements>
    </Link>
    <Link to='/search' style={{ textDecoration: "none", color: "inherit", width: '100%'
}}>
        <Elements>
            <SearchRoundedIcon />
            <NavText>Search</NavText>
        </Elements>
    </Link>
    {
        currentUser ?
        <Link to='/favourites' style={{ textDecoration: "none", color: "inherit", width:
'100%' }}>
            <Elements>
                <FavoriteRoundedIcon />
                <NavText>Favourites</NavText>
            </Elements>
        </Link >
        :
        <Link onClick={() =>
            dispatch(

                openSignin()
            )
        } style={{ textDecoration: "none", color: "inherit", width: '100%' }}>
            <Elements>
                <FavoriteRoundedIcon />

                <NavText>Favourites</NavText>
            </Elements>

        </Link >
    }

    <HR />
    <Link onClick={() => {
        if (currentUser) {
            setUploadOpen(true)
        }
    else {
        dispatch(
            openSignin()
        )
    }
    }

```

```

    }
  } style={{ textDecoration: "none", color: "inherit", width: '100%' }}>
    <Elements>
      <BackupRoundedIcon />

      <NavText>Upload</NavText>

    </Elements>

  </Link>

  {
    darkMode ?
    <>
      <Elements onClick={() => setDarkMode(false)}>
        <LightModeRoundedIcon />
        <NavText>Light Mode</NavText>
      </Elements>
    </>
    :
    <>
      <Elements onClick={() => setDarkMode(true)}>
        <DarkModeRoundedIcon />
        <NavText>Dark Mode</NavText>
      </Elements>
    </>
  }
  {
    currentUser ?
    <Elements onClick={() => logoutUser()}>

    <ExitToAppRoundedIcon />

    <NavText>Log Out</NavText>
    </Elements>

    :
    <Elements onClick={() => dispatch(openSignin())}>
      <ExitToAppRoundedIcon />

      <NavText>Log In</NavText>
    </Elements>
  }

  </MenuContainer >
)
}

export default Menu

```

B. SCREENSHOTS

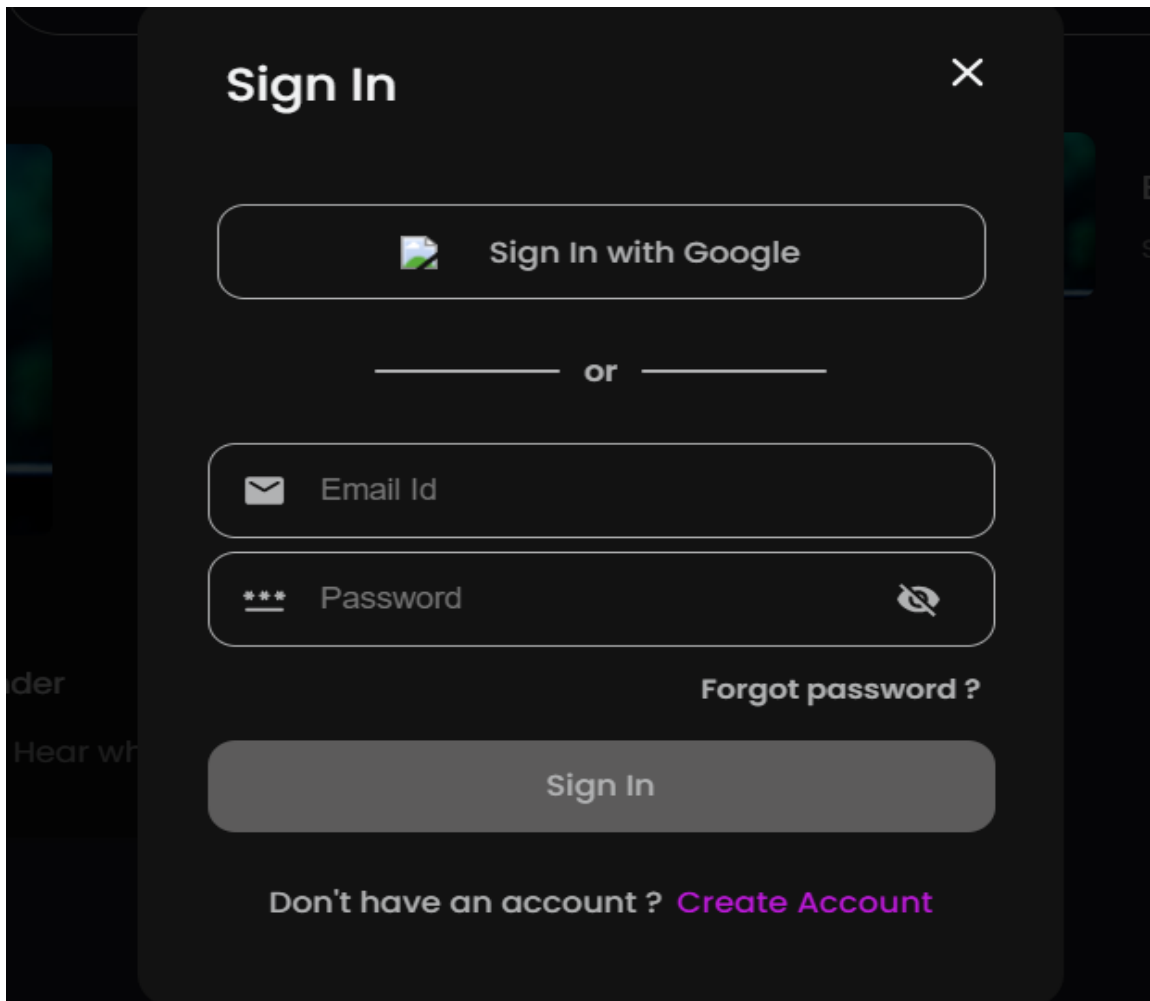


Figure B.1. Login Page

In Figure B.1 The login page of the podcast management system allows listeners to access their personalized content by using their registered email ID and password. Upon entering the correct credentials, they are authenticated and granted access to a tailored experience, including favorite podcasts, recommended episodes, and user preferences. The page includes clear input fields for the email ID and password, with an option for users to reset their password if forgotten. Additionally, a "Remember Me" checkbox is provided, enabling users to stay logged in for future sessions, enhancing convenience. If authentication fails, an error message is displayed, guiding users to correct their login details. The login page also includes links for new users to sign up, providing easy navigation to the registration process. Overall, the design focuses on simplicity and security, ensuring a smooth login experience for the listener.

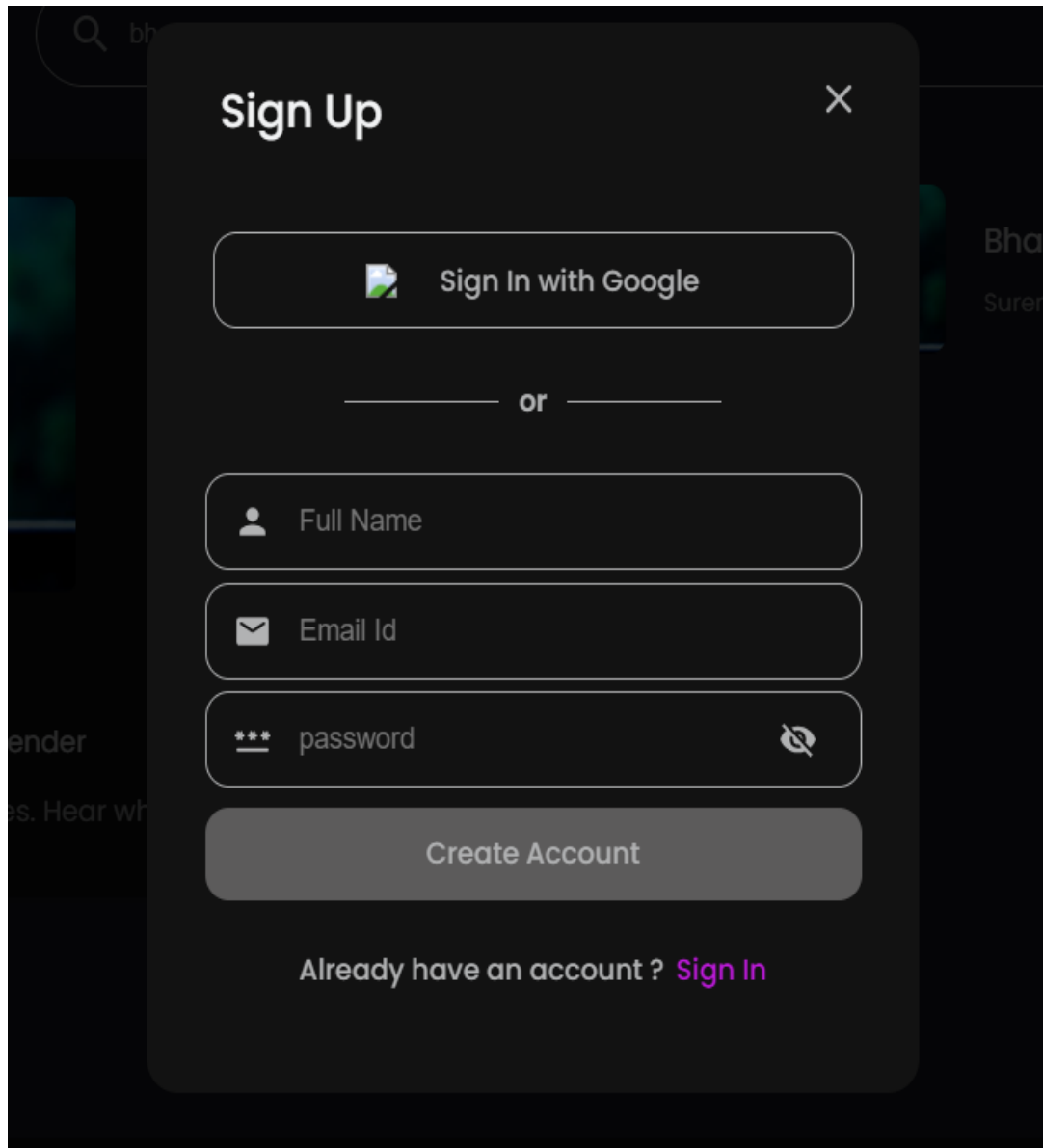


Figure B.2. Sign Up Page

In Figure B.2 shows the sign-up page which is mandatory part. Entering the details of the users that will be stored into the database base. The Sign-Up Page in a podcast website enables new users to create accounts by providing essential details like name, email, and password. It features a simple, user-friendly design with input validation to ensure accurate data entry. Additional options like profile image upload or interest selection may personalize the user experience. Secure password encryption safeguards user information during account creation. This page serves as the first step for users to access features like podcast streaming, favorites, and personalized recommendations.

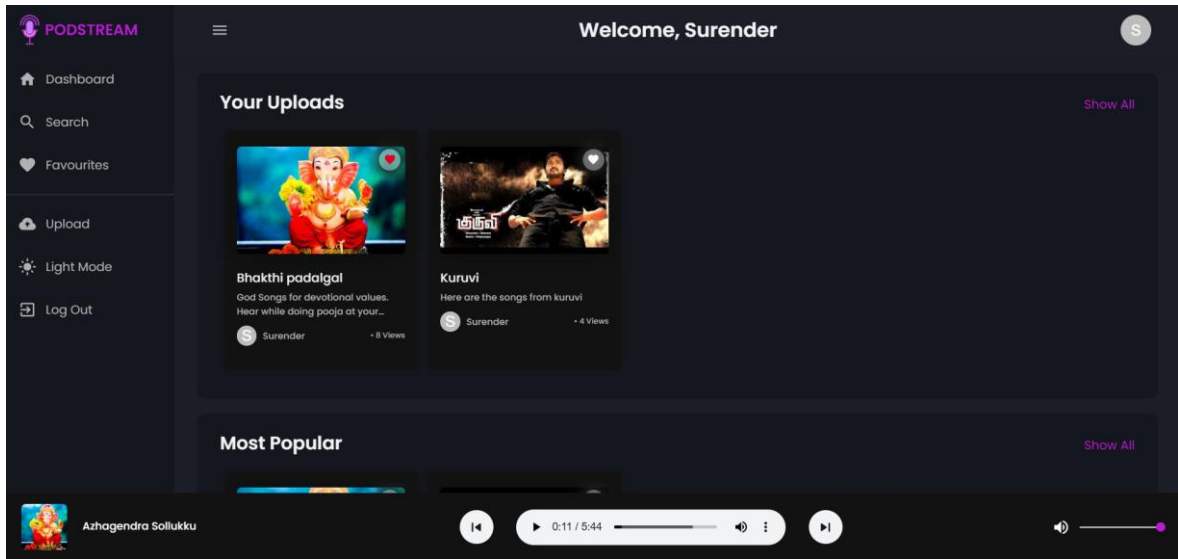


Figure B.3. Home Page

In Figure B.3 represents the Home Page of a podcast website serves as the central hub, showcasing featured podcasts, trending categories, and personalized recommendations. It offers a clean layout with easy navigation to explore content, search for podcasts, and access user profiles. This page ensures an engaging and user-friendly experience for both listeners and creators.

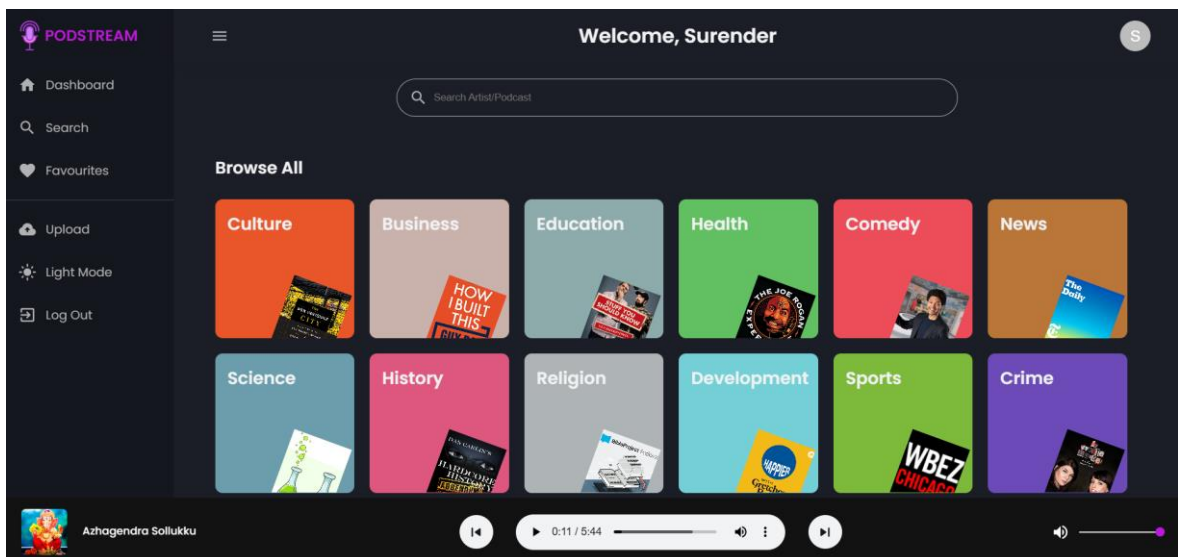
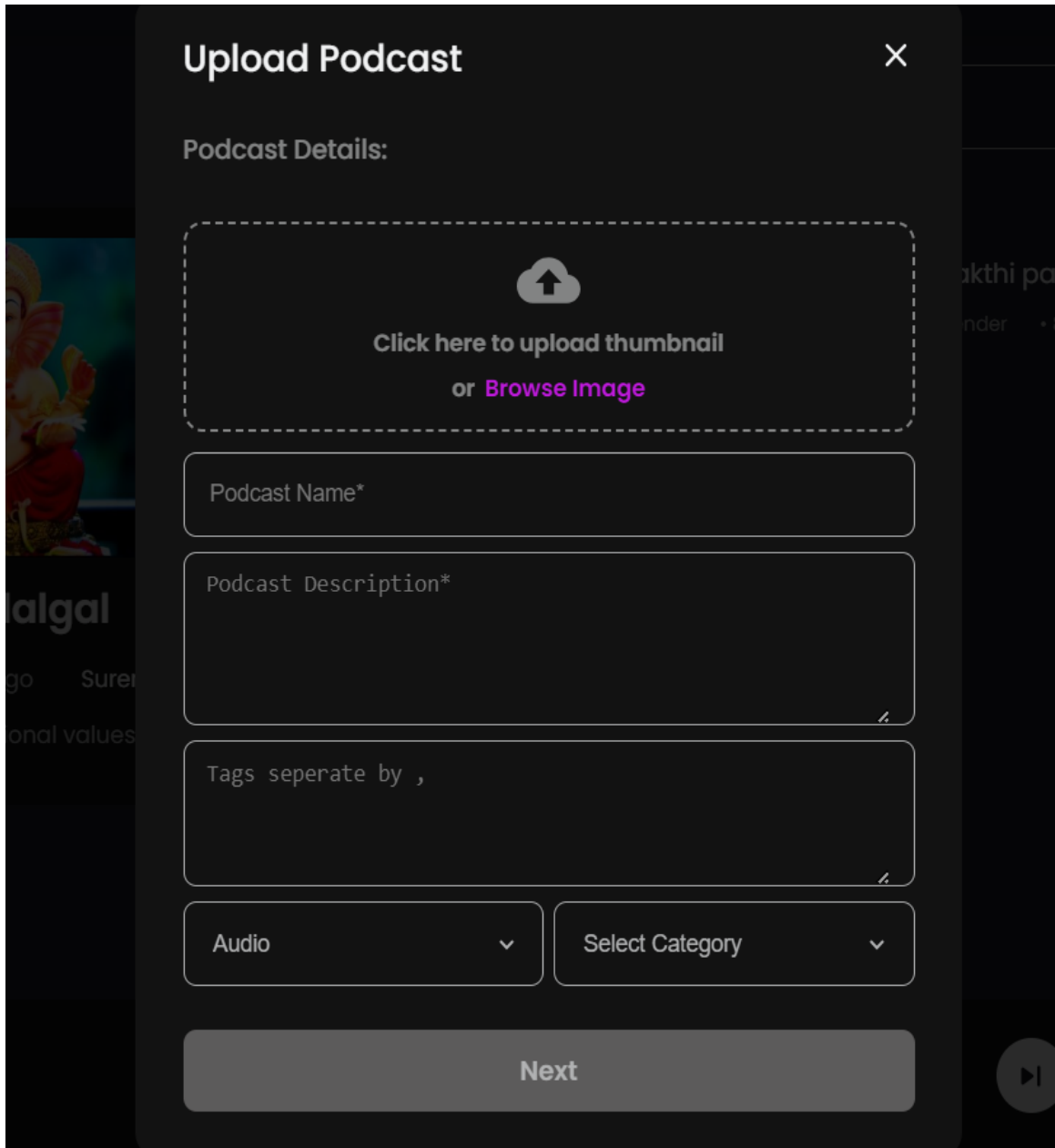



Figure B.4. Search Podcast

In Figure B.4 represents the Search Module in a podcast website allows users to find podcasts by entering keywords such as titles, descriptions, or creator names. It displays results with thumbnails, names, and other details, enabling quick and efficient navigation. This feature enhances content discovery, making it easier for users to explore their interests.



Upload Podcast ×

Podcast Details:


Click here to upload thumbnail
or [Browse Image](#)

Podcast Name*

Podcast Description*

Tags separate by ,

Audio ▼ Select Category ▼

Next

Figure B.5. Upload Podcast

In Figure B.5 represents the Upload Podcast Module in a podcast website allows creators to share their content by submitting essential details. Users can upload a cover image to visually represent the podcast and audio files for streaming. They provide a description to highlight the podcast's purpose, select a category and specify a name for easy identification. The module ensures a seamless upload process with user-friendly forms and validations. This feature empowers creators to distribute their podcasts efficiently and reach their target audience.

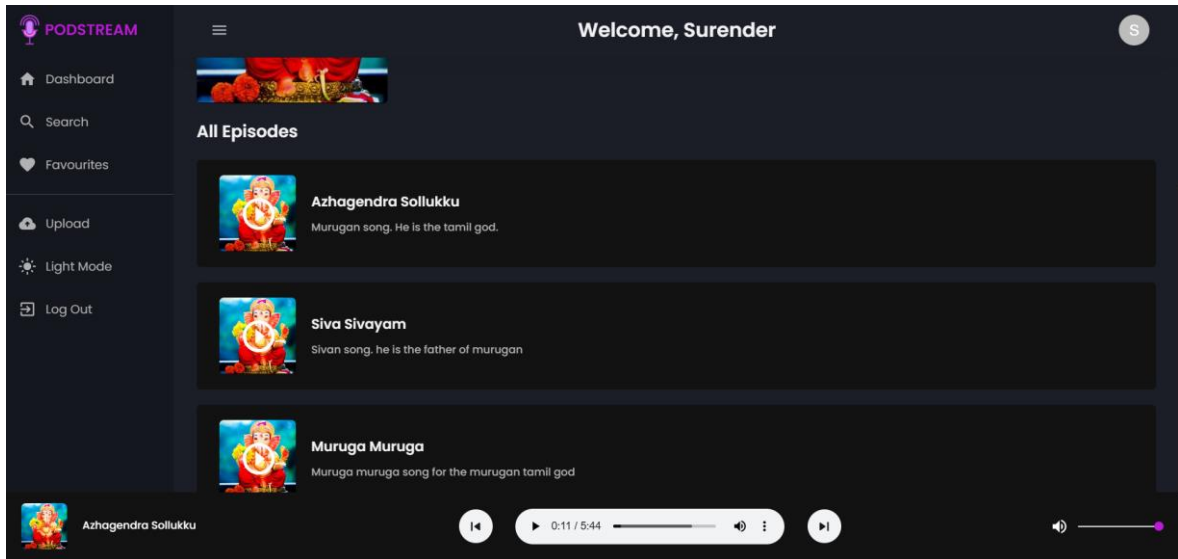


Figure B.6. Episode List

In Figure B.6 represents the Episode List in a podcast website displays all episodes of a selected podcast in an organized layout. Each episode includes details such as the title, description, duration, and release date, along with playback options. This feature allows users to browse and access episodes easily, enhancing their listening experience.

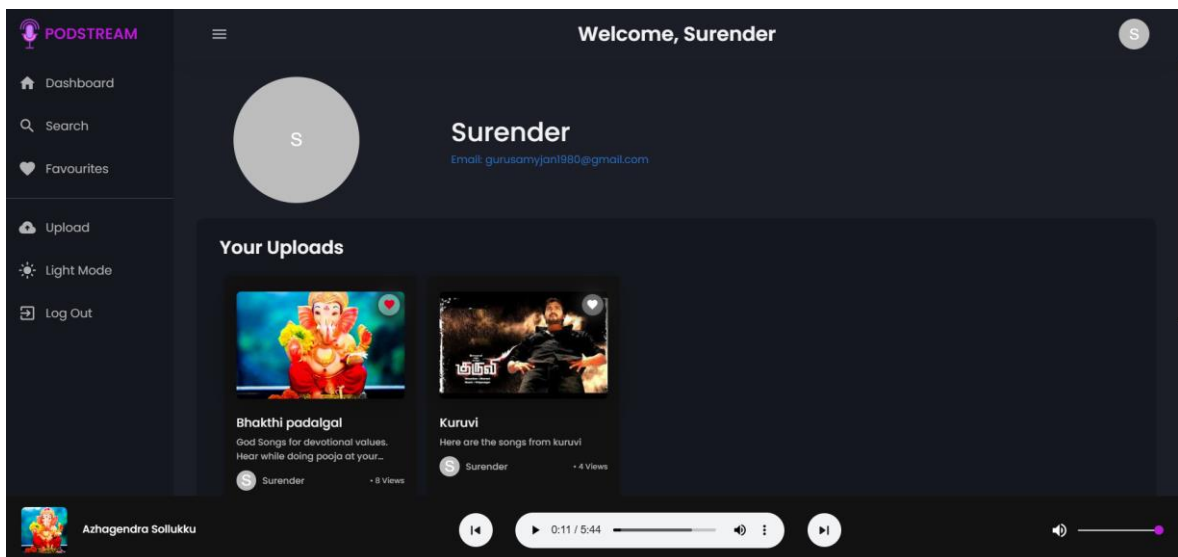


Figure B.7. Listener profile

In Figure B.7 represents the Listener Profile Page displays user details, listening history, and favourite podcasts, offering a personalized experience. It allows users to manage their account, preferences, and explore recommended content.

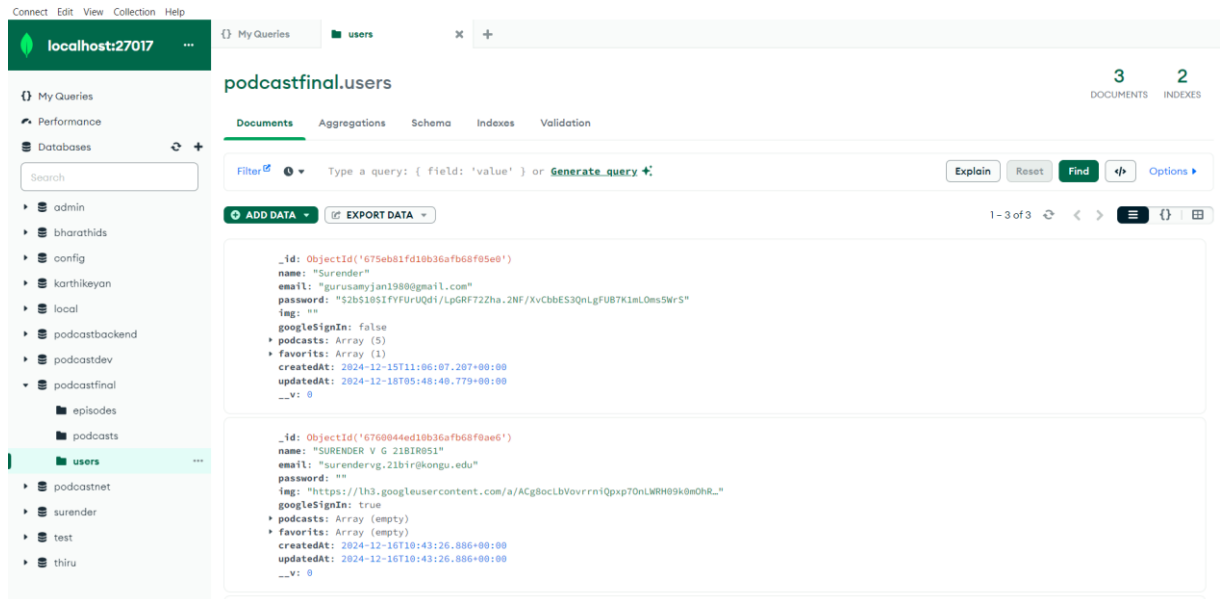


Figure B.8. User Data in Data Base

In Figure B.8 represents the MongoDB, user data for login and signup functionality is typically organized within a users collection. Each document in this collection represents an individual user and contains various fields to store user information securely.

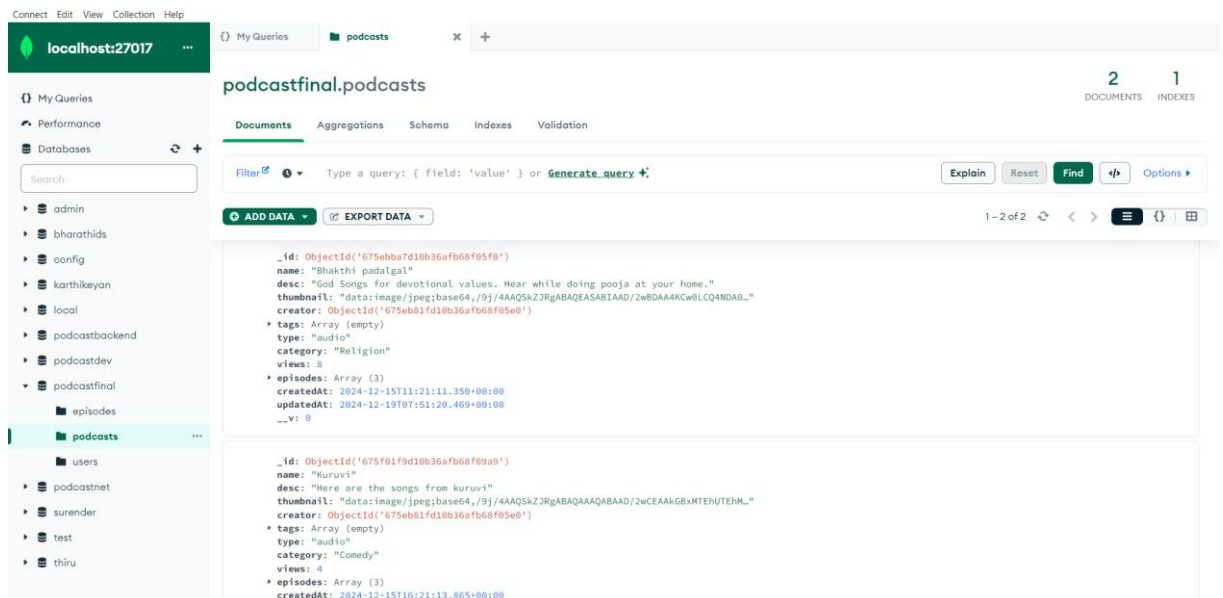


Figure B.9. Podcast Data in Data Base

In Figure B.9 represents the MongoDB, product data for the uploaded product functionality is typically organized within a products collection. Each document in this collection represents the information for the farm product and store the information securely.

REFERENCES

- [1] Ryan Foster, "MERN Stack Development: A Practical Guide", 1st Edition, 2022.
- [2] Christopher Rodriguez, "The Art of NoSQL: MongoDB in Action", 3rd Edition, 2017.
- [3] Victoria Simmons, "Web Application Security Handbook", 2nd Edition, 2019.
- [4] Olivia Foster, "React Design Patterns: Best Practices for Scalable Code", 1st Edition, 2020.
- [5] Ethan Knight, "Node.js for Beginners: From Novice to Ninja", 3rd Edition, 2016.
- [6] Gabriella Bennett, "MongoDB in Practice: Real-World Examples", 4th Edition, 2019.
- [7] <https://www.geeksforgeeks.org/reactjs/>
- [8] <https://www.mongodb.com/>
- [9] <https://www.geeksforgeeks.org/nodej/>
- [10] www.simplilearn.com, 'Learn Reactjs with MongoDB'
- [11] www.skillshare.com/
- [12] www.stackoverflow.co
- [13] www.w3school.com, 'Learn Reactjs'
- [14] <https://www.mongodb.com/languages/mern-stack-tutorial>