ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

BANGALORE INSTITUTE OF TECHNOLOGY

K.R.ROAD, V.V.PURAM, BANGALORE - 560004

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Project On

Audio Podcast Platform using MongoDB

Presented By:

Anagha Kashyap	1BI21IS010
Asha Latha A	1BI21IS017
Keerthika Shetty	1BI21IS041
Karthika G	1BI21IS040

Under the guidance of :
Prof. Padhmanabha J
Assistant Professor,
Dept. of ISE, BIT

CONTENTS

1. Introduction

The Audio Podcast Platform is a modern web application designed to provide a comprehensive and user-friendly environment for podcast enthusiasts. It caters to both podcasters looking to share their content and listeners searching for engaging audio experiences.

Key Features

1. **User Registration and Login**
    1. Enables users to create accounts and log in securely.
2. **Categorized Podcast Browsing**
    1. Podcasts are organized into categories like thrill, health, sports, comedy, and government.
    2. Simplifies the discovery process for listeners by providing relevant content.

3. **User-Generated Content**
    1. Registered users can upload podcasts with details such as title, description, category, and images.
    2. Promotes user-driven content creation, empowering creators.
4. **All-Podcasts Page**
    1. Lists all available podcasts on the platform, offering users a complete catalog.
5. **Profile Management**
    1. Users can view and manage podcasts they’ve uploaded, including adding new ones or updating existing entries.

2. Problem Statement

Problem Statement:

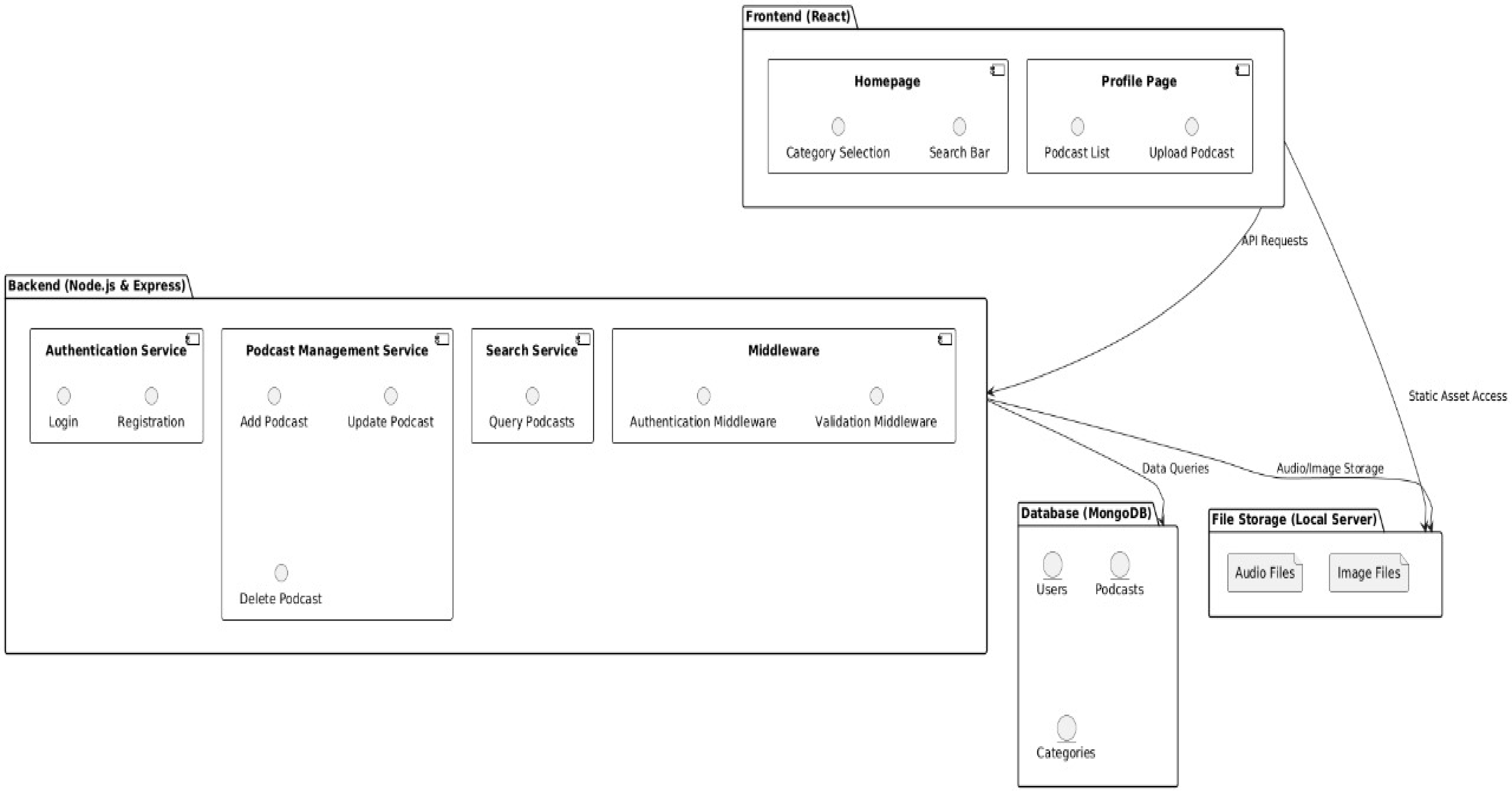
The podcasting industry faces several challenges, including:

- Lack of platforms where users can easily upload and manage their podcasts.
- Difficulty in categorizing and discovering podcasts based on user preferences.
- Limited tools for beginner podcasters to publish content efficiently.
- Inefficient data storage and retrieval in existing platforms, especially with increasing podcast volumes.

  This platform addresses these challenges by introducing a scalable and user-friendly system.

3. System Architecture

Audio Podcast Platform Architecture



4. Tools and Technologies

MongoDB

- A NoSQL database ideal for schema-less and dynamic data.
- Used to store podcast metadata (e.g., title, description, category, file paths) and user details (e.g., name, email).
- Provides high scalability and flexibility in handling growing data volumes.

Mongoose

- Integrated Object Data Modeling (ODM) library to simplify interactions with MongoDB.
- Acts as a bridge between JavaScript code and the database, ensuring consistency in data through schema-based solutions.

Backend

- **Node.js**:
  - A JavaScript runtime environment for server-side development.
  - Handles asynchronous processing to improve performance during multiple user requests.
- **Express.js**:
  - A lightweight and flexible Node.js framework for building REST APIs.
  - Simplifies routing and middleware integration for authentication, file handling, and error handling.

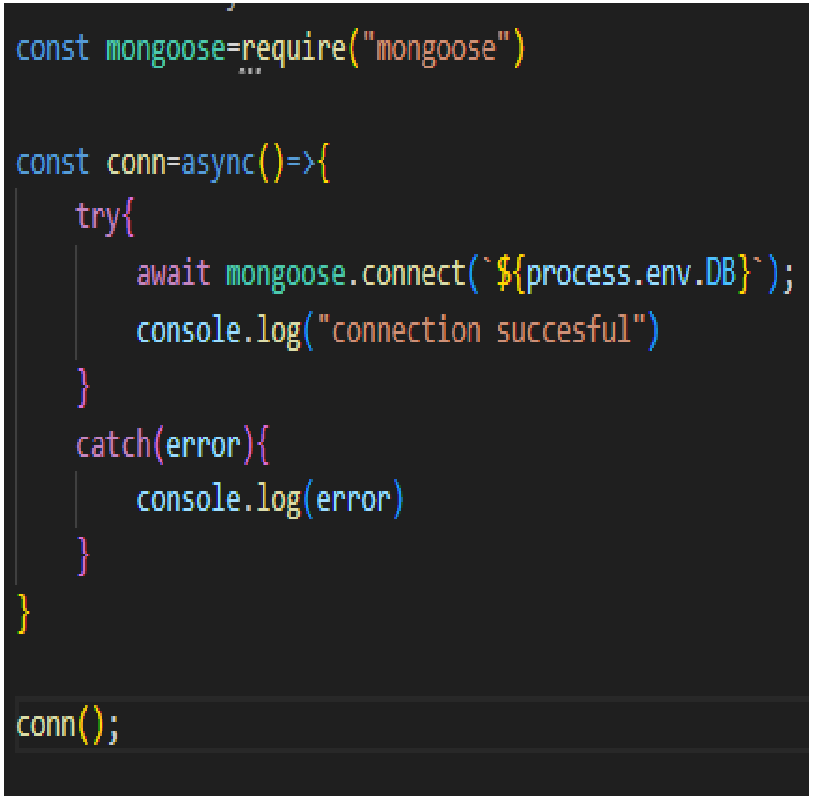
Frontend

- **React.js**:
  - A JavaScript library for building user interfaces.
  - Provides a dynamic and responsive UI, enabling fast rendering and seamless navigation.
  - Component-based architecture allows modular development of pages such as Home, Categories, and Profile.

- **HTML5 and CSS3**:
  - HTML5 is used for structuring the webpage content.
  - CSS3 ensures a visually appealing layout and styling of components.

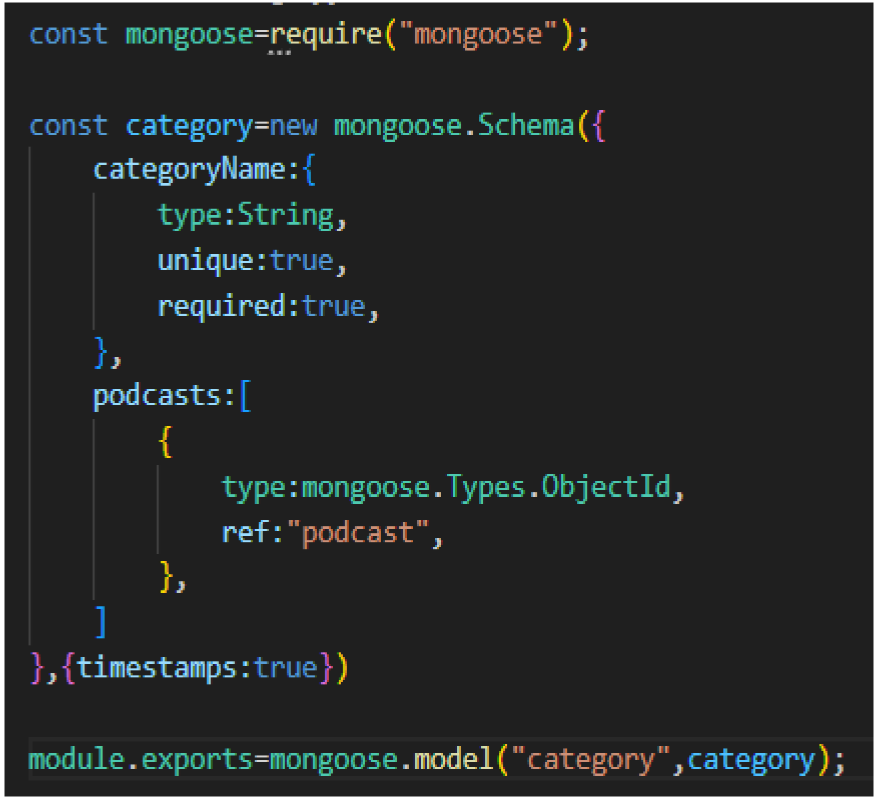
- **JavaScript (ES6)**:
  - Implements interactivity and logic on the client side.
  - Ensures smooth communication with backend APIs.

5. Implementation



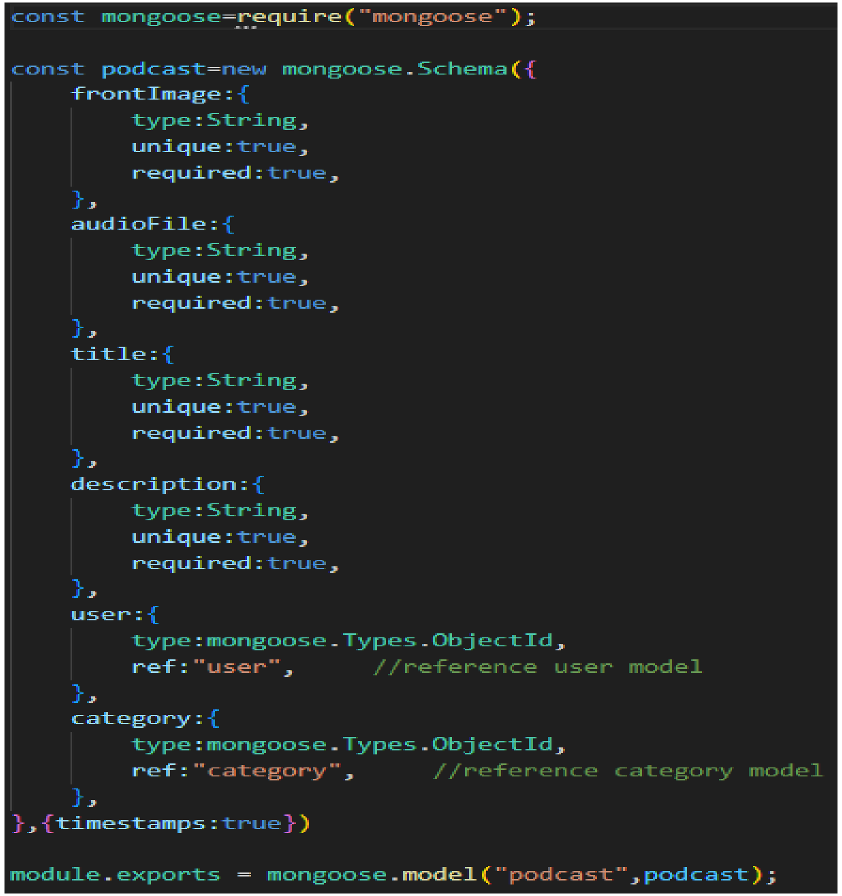
```
const mongoose=require("mongoose")

const conn=async()=>{
    try{
        await mongoose.connect(`${process.env.DB}`);
        console.log("connection succesful")
    }
    catch(error){
        console.log(error)
    }
}

conn();
```

MongoDB connection



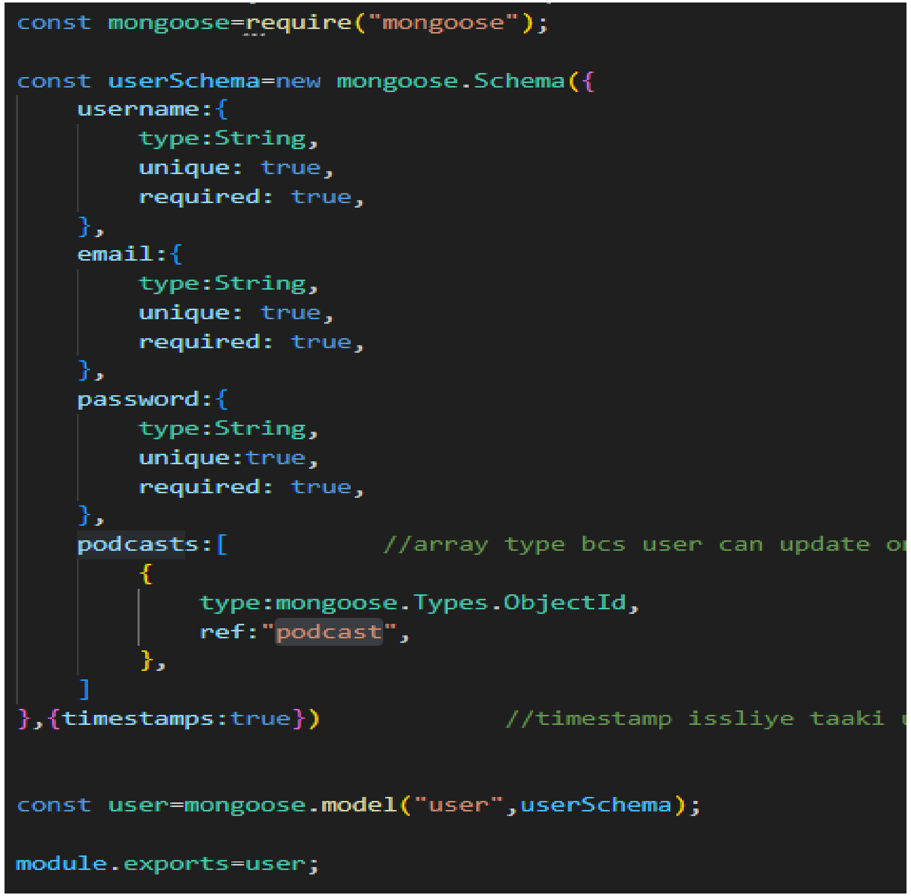
```
const mongoose=require("mongoose");

const category=new mongoose.Schema({
    categoryName:{
        type:String,
        unique:true,
        required:true,
    },
    podcasts:[
        {
            type:mongoose.Types.ObjectId,
            ref:"podcast",
        },
    ]
},{timestamps:true})

module.exports=mongoose.model("category",category);
```

Category Model



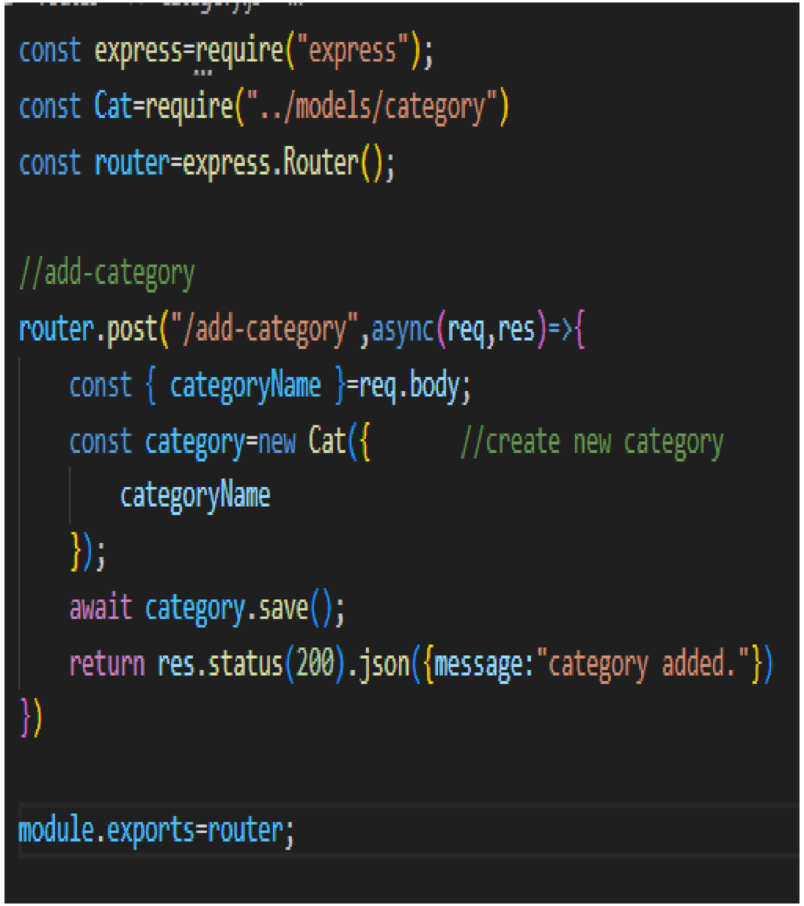
```
const mongoose=require("mongoose");

const podcast=new mongoose.Schema({
    frontImage:{
        type:String,
        unique:true,
        required:true,
    },
    audioFile:{
        type:String,
        unique:true,
        required:true,
    },
    title:{
        type:String,
        unique:true,
        required:true,
    },
    description:{
        type:String,
        unique:true,
        required:true,
    },
    user:{
        type:mongoose.Types.ObjectId,
        ref:"user",      //reference user model
    },
    category:{
        type:mongoose.Types.ObjectId,
        ref:"category",      //reference category model
    },
},{timestamps:true})

module.exports = mongoose.model("podcast",podcast);
```

Podcast Model



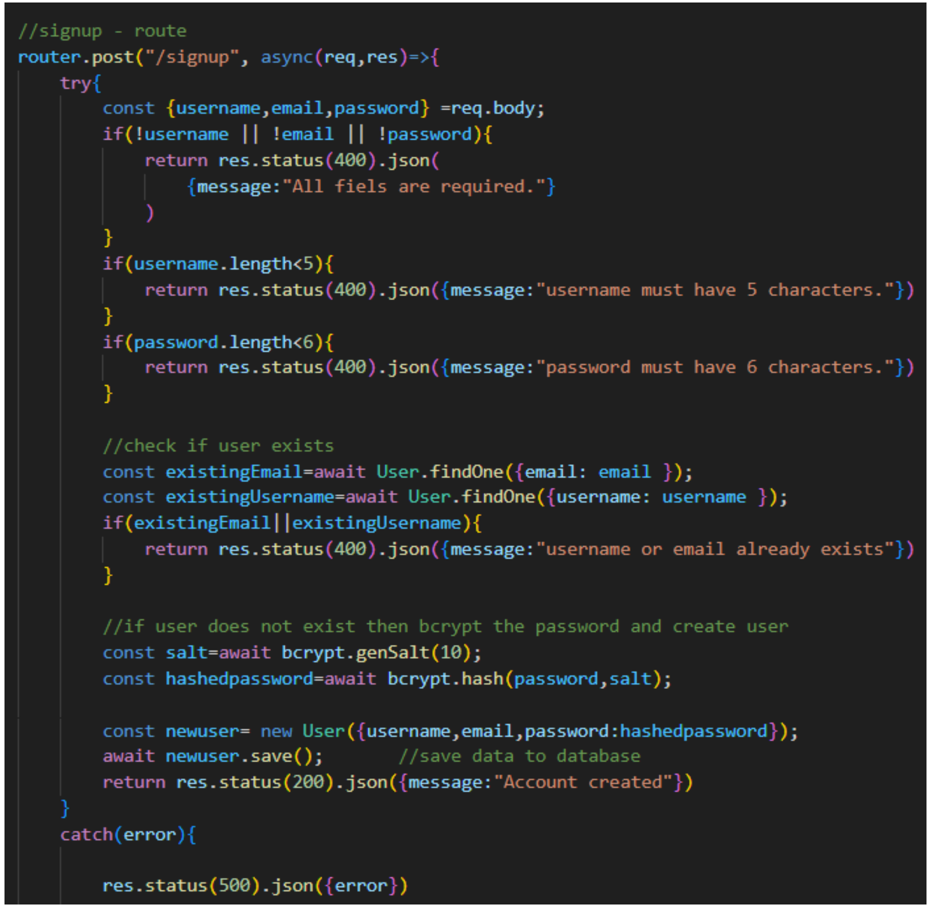
```
const mongoose=require("mongoose");

const userSchema=new mongoose.Schema({
    username:{
        type:String,
        unique: true,
        required: true,
    },
    email:{
        type:String,
        unique: true,
        required: true,
    },
    password:{
        type:String,
        unique:true,
        required: true,
    },
    podcasts:[          //array type bcs user can update o
        {
            type:mongoose.Types.ObjectId,
            ref:"podcast",
        },
    ]
},{timestamps:true})          //timestamp issliye taaki

const user=mongoose.model("user",userSchema);

module.exports=user;
```

User Model



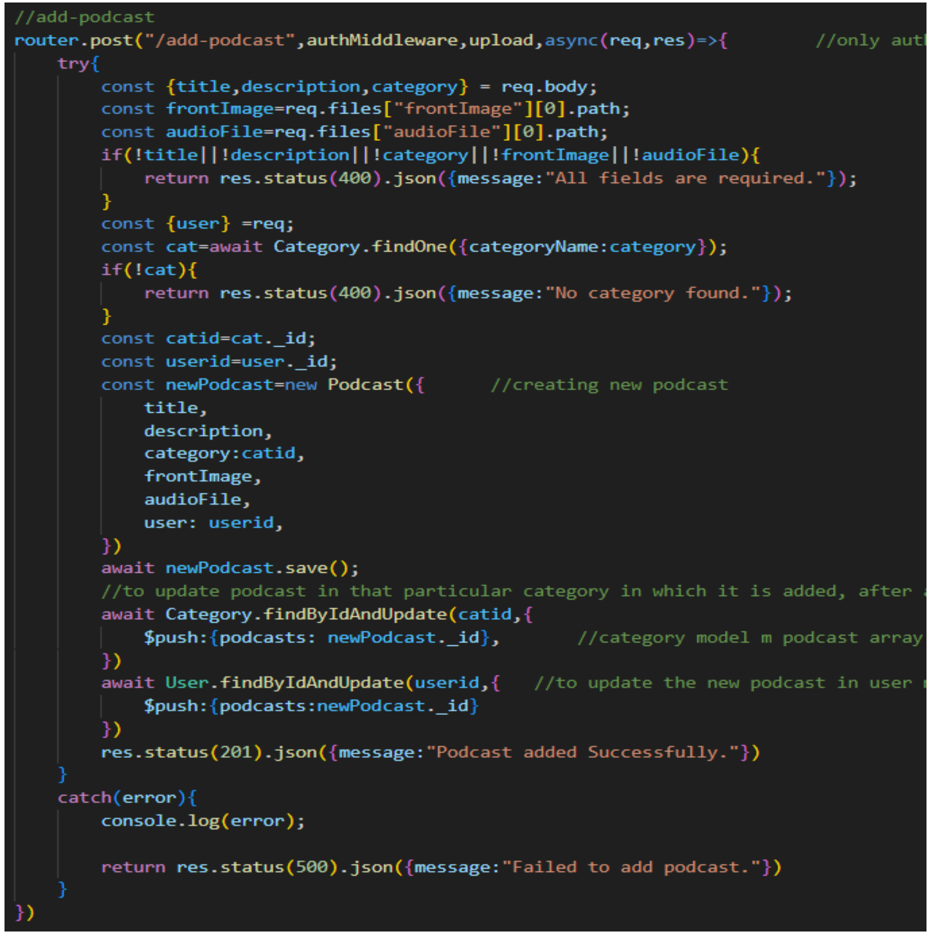
```
const express=require("express");
const Cat=require("../models/category")
const router=express.Router();

//add-category
router.post("/add-category",async(req,res)=>{
    const { categoryName }=req.body;
    const category=new Cat({       //create new category
        categoryName
    });
    await category.save();
    return res.status(200).json({message:"category added."})
})

module.exports=router;
```

Add Category



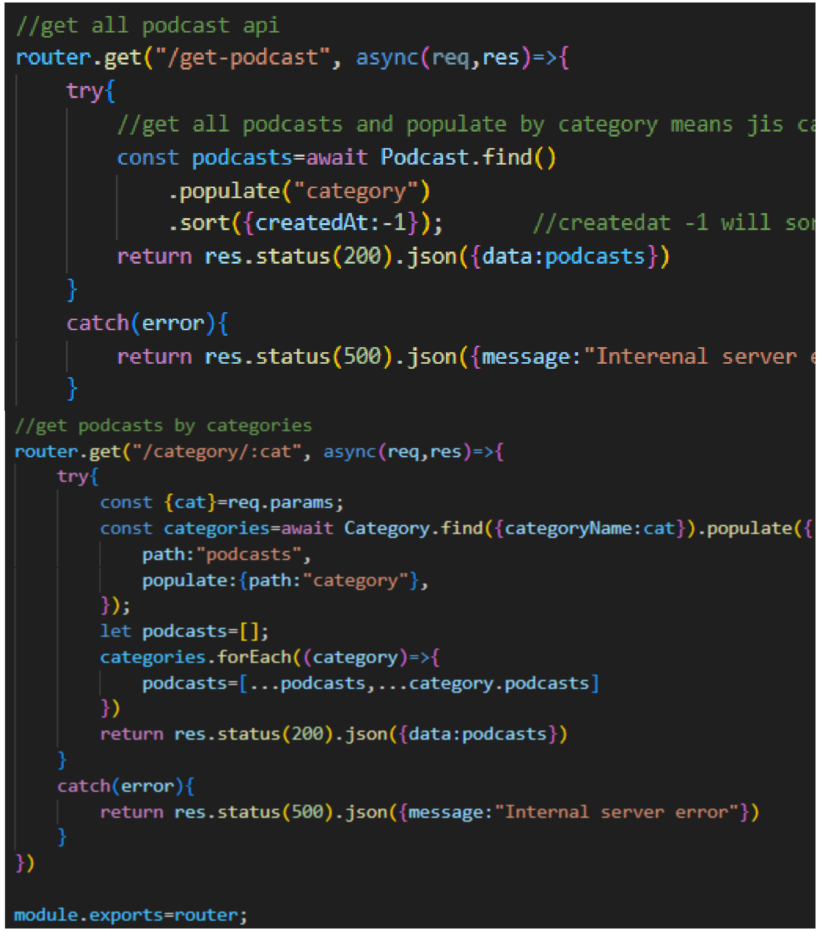
```
//signup - route
router.post("/signup", async(req,res)=>{
    try{
        const {username,email,password} =req.body;
        if(!username || !email || !password){
            return res.status(400).json(
                {message:"All fiels are required."}
            )
        }
        if(username.length<5){
            return res.status(400).json({message:"username must have 5 characters."})
        }
        if(password.length<6){
            return res.status(400).json({message:"password must have 6 characters."})
        }

        //check if user exists
        const existingEmail=await User.findOne({email: email });
        const existingUsername=await User.findOne({username: username });
        if(existingEmail||existingUsername){
            return res.status(400).json({message:"username or email already exists"})
        }

        //if user does not exist then bcrypt the password and create user
        const salt=await bcrypt.genSalt(10);
        const hashedpassword=await bcrypt.hash(password,salt);

        const newuser= new User({username,email,password:hashedpassword});
        await newuser.save();       //save data to database
        return res.status(200).json({message:"Account created"})
    }
    catch(error){

        res.status(500).json({error})
```

Register



```
//add-podcast
router.post("/add-podcast",authMiddleware,upload,async(req,res)=>{        //only aut
    try{
        const {title,description,category} = req.body;
        const frontImage=req.files["frontImage"][0].path;
        const audioFile=req.files["audioFile"][0].path;
        if(!title||!description||!category||!frontImage||!audioFile){
            return res.status(400).json({message:"All fields are required."});
        }
        const {user} =req;
        const cat=await Category.findOne({categoryName:category});
        if(!cat){
            return res.status(400).json({message:"No category found."});
        }
        const catid=cat._id;
        const userid=user._id;
        const newPodcast=new Podcast({      //creating new podcast
            title,
            description,
            category:catid,
            frontImage,
            audioFile,
            user: userid,
        })
        await newPodcast.save();
        //to update podcast in that particular category in which it is added, after
        await Category.findByIdAndUpdate(catid,{
            $push:{podcasts: newPodcast._id},       //category model m podcast array
        })
        await User.findByIdAndUpdate(userid,{   //to update the new podcast in user
            $push:{podcasts:newPodcast._id}
        })
        res.status(201).json({message:"Podcast added Successfully."})
    }
    catch(error){
        console.log(error);

        return res.status(500).json({message:"Failed to add podcast."})
    }
})
```

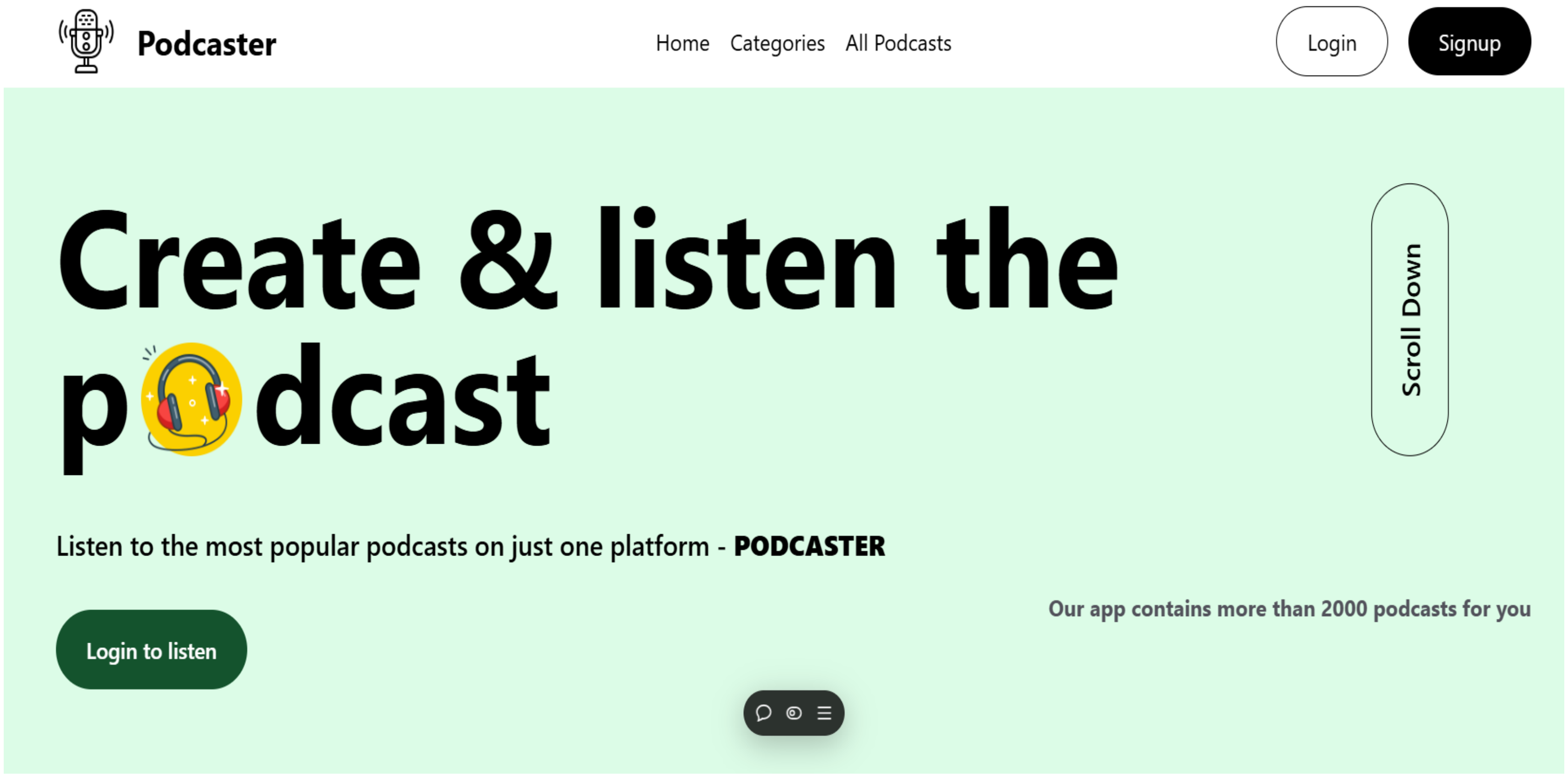
Add Podcast



```
//get all podcast api
router.get("/get-podcast", async(req,res)=>{
    try{
        //get all podcasts and populate by category means jis ca
        const podcasts=await Podcast.find()
            .populate("category")
            .sort({createdAt:-1});        //createdat -1 will so
        return res.status(200).json({data:podcasts})
    }
    catch(error){
        return res.status(500).json({message:"Interenal server
    }
//get podcasts by categories
router.get("/category/:cat", async(req,res)=>{
    try{
        const {cat}=req.params;
        const categories=await Category.find({categoryName:cat}).populate({
            path:"podcasts",
            populate:{path:"category"},
        });
        let podcasts=[];
        categories.forEach((category)=>{
            podcasts=[...podcasts,...category.podcasts]
        })
        return res.status(200).json({data:podcasts})
    }
    catch(error){
        return res.status(500).json({message:"Internal server error"})
    }
})

module.exports=router;
```

Get all podcast and get podcast by category

6. Snapshots



Podcaster

Home Categories All Podcasts

Login Signup

Create & listen the podcast

Scroll Down

Listen to the most popular podcasts on just one platform - **PODCASTER**

Our app contains more than 2000 podcasts for you

Login to listen

Fig1: Home Page

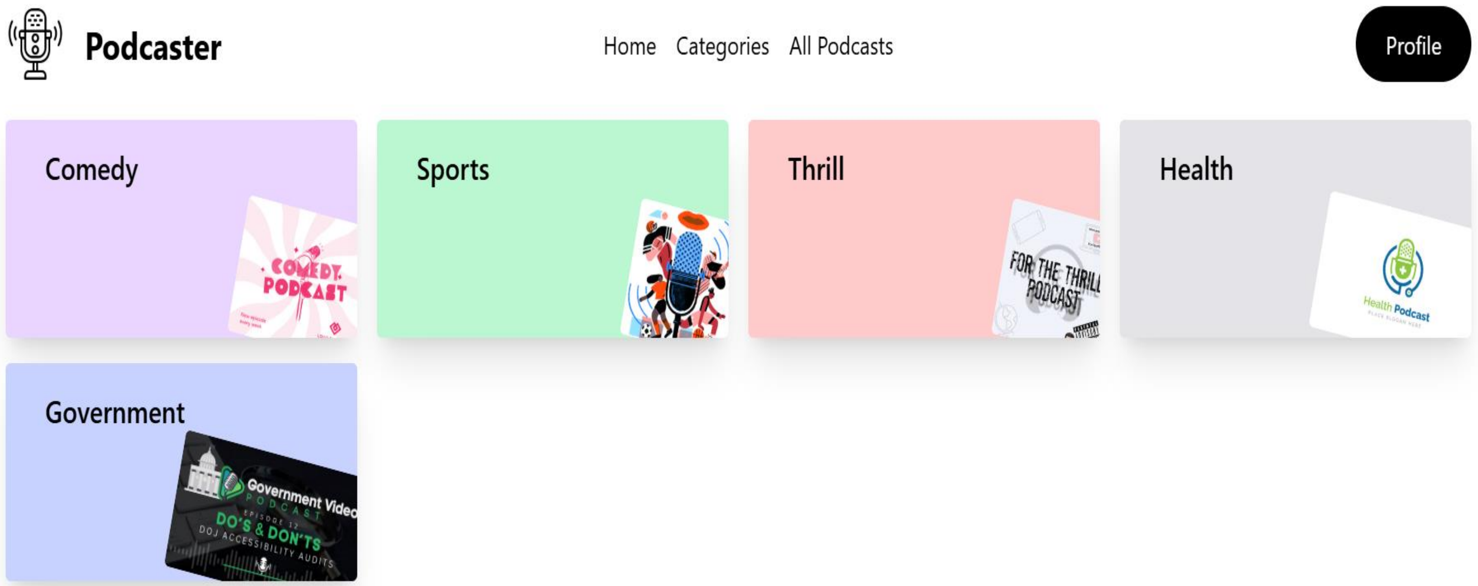
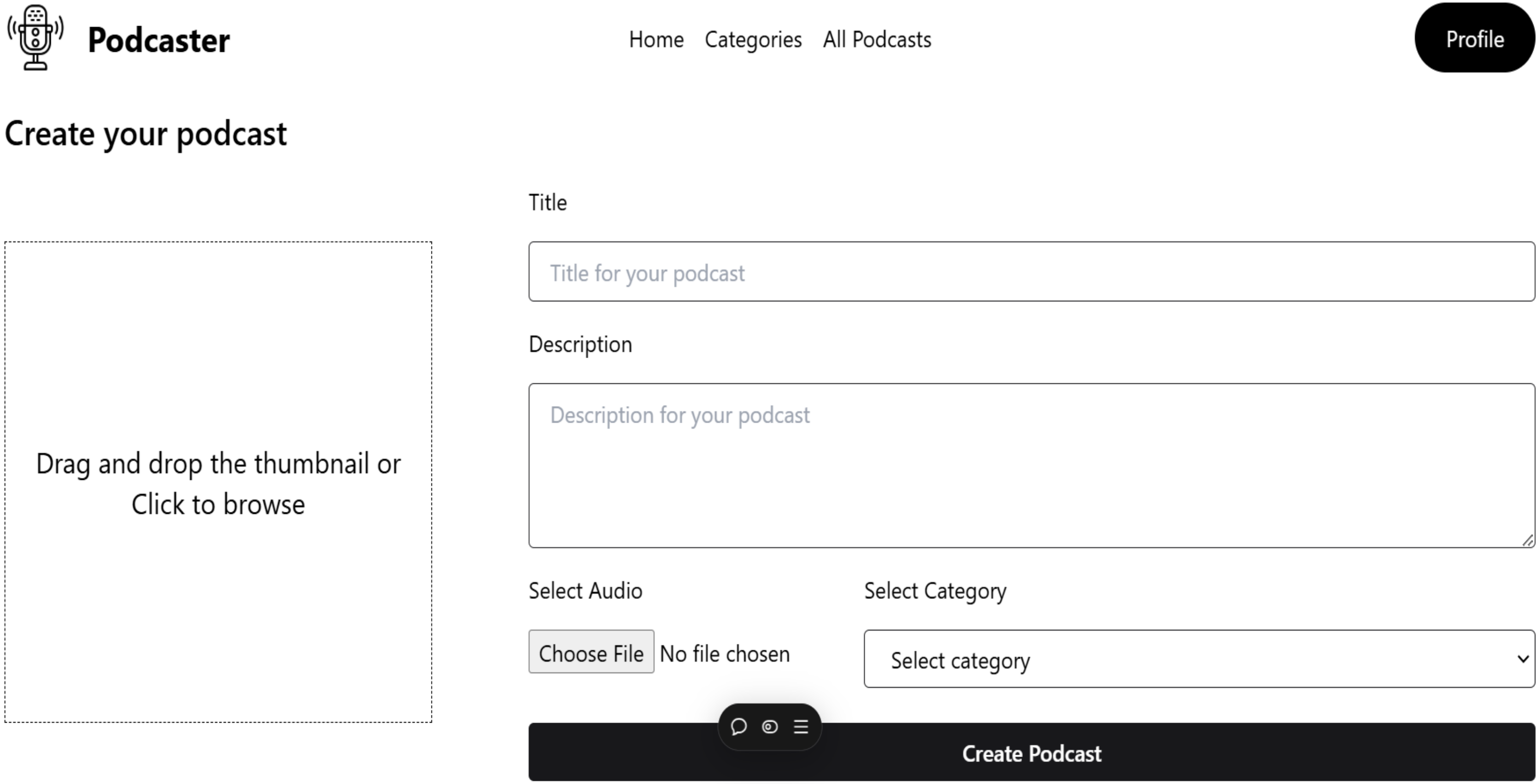


Fig2: Categories Page



Podcaster

Home Categories All Podcasts

Profile

Create your podcast

Drag and drop the thumbnail or Click to browse

Title

Title for your podcast

Description

Description for your podcast

Select Audio

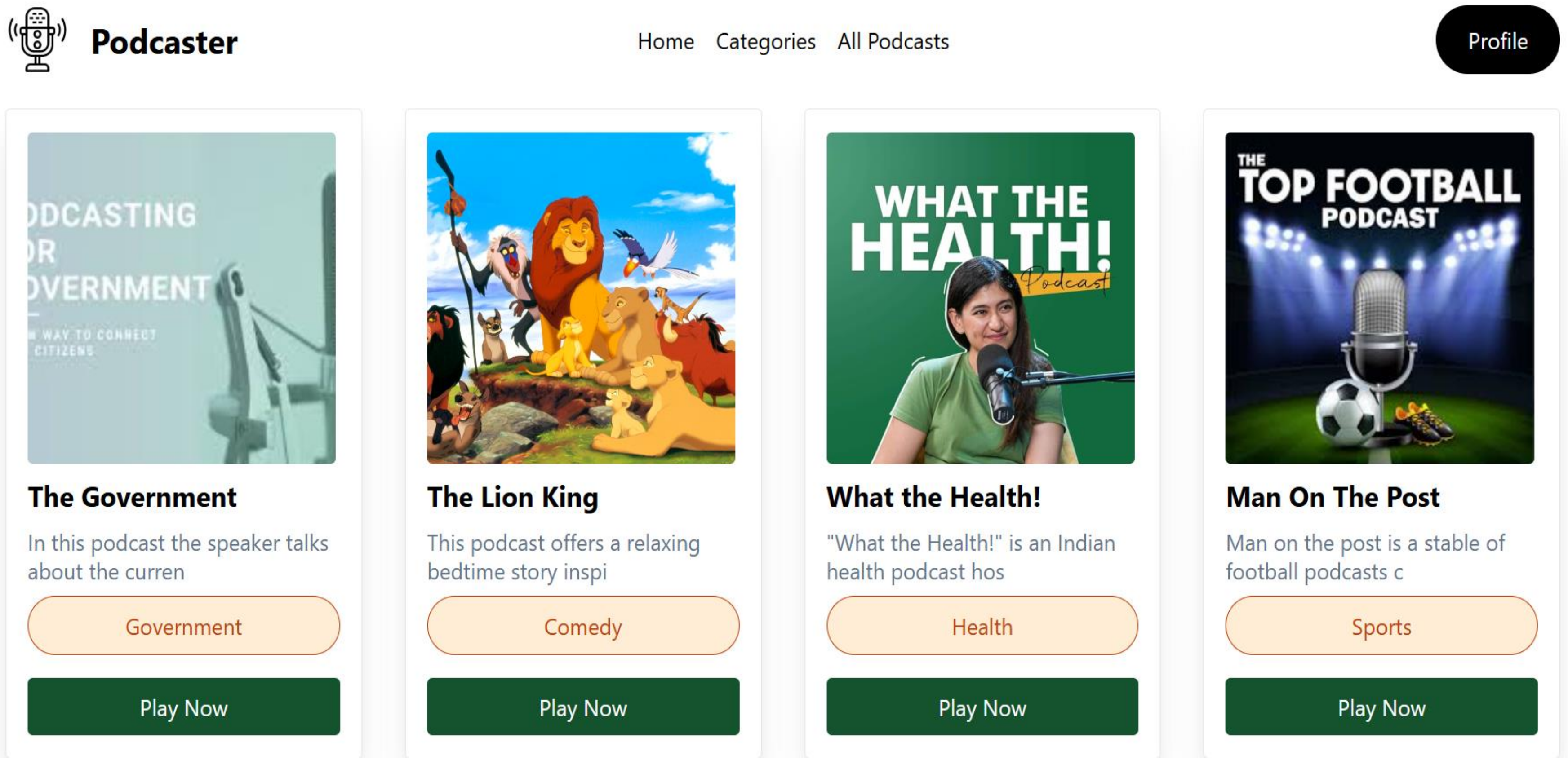
Choose File No file chosen

Select Category

Select category

Create Podcast

Fig3: Add Podcast



The Government

In this podcast the speaker talks about the curren

The Lion King

This podcast offers a relaxing bedtime story inspi

What the Health!

"What the Health!" is an Indian health podcast hos

Man On The Post

Man on the post is a stable of football podcasts c

Fig4: All Podcast Page

7. Importance of MongoDB

MongoDB is a cornerstone of the platform due to its following features:

- **Schema-Less Design**:
  - Allows flexibility in storing diverse podcast data formats (e.g., title, description, category, audio file paths).
  - Facilitates quick updates without schema migration, making it ideal for dynamic data.

- **Scalability**:
  - Efficiently handles increasing user data and podcast metadata as the platform grows.

- **High Performance**:
  - Offers fast read/write operations, ensuring smooth user experience even during heavy traffic.

Querying and Indexing

- MongoDB’s querying capabilities are used for efficient message retrieval, such as fetching messages by timestamp or filtering by sender/receiver in real-time communication.

Session Management

- MongoDB can store session data for managing user authentication (e.g., login/logout functionality). Libraries like connect-mongo are used to manage sessions efficiently.

References

[1] https://www.mongodb.com/

[2] https://www.w3schools.com/mongodb/

[3] https://nodejs.org/en

[4] https://expressjs.com/

[5] https://ejs.co/

Thank You