RAMACHANDRA COLLEGE OF ENGINEERING
AUTONOMOUS

Approved by AICTE, New Delhi
Permanently Affiliated to JNTUK
Recognized by UGC 2(f) & 12(B)
Accreditations
NAAC A+
NBA (EEE, Civil, ME, ECE & CSE)
ISO 9001 : 2015 Certified

BRAIN BUZZ – QUIZ APPLICATION

A PROJECT REPORT SUBMITTED BY:

1. **MAGANTI PRAVEEN SAI** **22ME1A05G5**
2. **CHANDU ANAND SAI VIVEK ,** **22ME5A0512**
3. **GUDAPATI PAVAN SAI** **22ME1A05F0**
4. **KATUMURI THARUN** **22ME1AO5J5**
5. **K.BHASKAR SRI PAVAN RAM** **22ME1A0593**

BACHELOR OF TECHNOLOGY IN
COMPUTER SCIENCE AND ENGINEERING

RAMACHANDRA COLLEGE OF ENGINEERING
AUTONOMOUS

Approved by **AICTE**, New Delhi
Permanently Affiliated to **JNTUK**
Recognized by **UGC 2(f) & 12(B)**
Accreditations
NAAC A+
NBA (EEE, Civil, ME, ECE & CSE)
ISO 9001 : 2015 Certified

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the "MERN Stack Mini Project" was successfully submitted by MAGANTI PRAVEEN SAI(22ME1A05G5),CHANDU ANAND SAI VIVEK (22ME5A0512), GUDAPATI PAVAN SAI (22ME1A05F0), KATUMURI THARUN (22ME1A05J5) and KONERU BHASKAR SRI PAVAN RAM(22ME1A0593) during the 2024-2025 Academic Year. This project was presented in partial fulfillment of the academic requirements for the III B. Tech II Semester of the Bachelor of Technology in Computer Science & Engineering (CSE).

Signatures:

(TRAINING INCHARGE) **(TRAINING COORDINATOR)**

P. BALAKRISHNA REDDY SIR **O. BHAGAVAN SIR**

DECLARATION

We hereby declare that our project titled “Quiz Web Application using MERN Stack” is a genuine and original piece of work completed as part of our academic curriculum. This project has been developed using MongoDB, Express.js, React.js, and Node.js, incorporating key technologies such as HTML, CSS, JavaScript, Axios, and RESTful APIs.

Our objective was to build a dynamic and interactive quiz platform that facilitates both student and admin roles — where students can register, attempt quizzes, track scores, and view leaderboards, and admins can log in, manage quiz content, and view performance analytics. The application also supports authentication, personalized dashboards, and real-time scoring functionality.

The work was carried out with sincerity, effort, and dedication under the guidance and mentorship of our respected trainer Mr. Kushal Sir, whose consistent support and insights played a vital role in shaping the project.
This project is entirely our own creation. No part of it has been copied from any unauthorized source or submitted elsewhere for academic credit. We have ensured originality in coding, design, documentation, and concept, and we have cited references and resources where applicable.

Through this project, we aim to showcase our understanding of full-stack development and contribute to digital learning by providing an intuitive, scalable, and fun quiz experience for users.

ACKNOWLEDGEMENT

We sincerely thank everyone who supported us in completing our project, “Quiz Application.” This project, built using the MERN stack, allowed us to apply our skills to create an interactive and engaging quiz platform. It was an enriching experience that helped us understand the importance of user-centric design and functionality. Our goal was to provide users with an immersive quiz experience while enabling admins to manage quizzes seamlessly.

A special thanks to **Kushal Sir**, our trainer and mentor, for his continuous guidance and encouragement throughout this journey. His insights and feedback were invaluable in shaping our approach and ensuring that the application was both efficient and scalable. His mentorship helped us refine our technical skills and made a significant impact on the overall success of this project.

We would also like to extend our gratitude to our faculty members for their support, as well as to our peers who provided us with valuable feedback. The testing process across different devices and browsers enabled us to ensure the app's reliability and responsiveness.

Finally, we appreciate the resources, tutorials, and documentation that guided us in learning and implementing new technologies. This project reflects our passion for web development and our dedication to creating meaningful, interactive solutions.

INDEX

ABSTRACT

The ability to assess knowledge and skills is an essential part of learning and development. Our project, Quiz Application, is a web-based platform designed to offer an interactive and user-friendly environment for users to take quizzes on various topics while allowing admins to create and manage quizzes effectively. Built using the MERN stack (MongoDB, Express.js, React.js, and Node.js), this application provides real-time quiz attempts, score tracking, and a leaderboard to foster a competitive learning experience.

The core functionality of the project revolves around enabling quiz creation by admins and providing an intuitive interface for users to attempt quizzes. Admins can create and manage different categories of questions, while users can answer questions, view their scores, and participate in the leaderboard. The system is designed for seamless interaction, where each quiz is timed, and users receive immediate feedback on their performance.

To enhance user experience, the application is optimized for responsiveness and accessibility across various devices and browsers. The React-based front-end ensures a smooth and dynamic user experience without needing page reloads. MongoDB is used to store user data, quiz questions, and performance statistics securely.

This project demonstrates our technical expertise in web development and offers a practical solution for interactive learning. It not only showcases our ability to create engaging user interfaces but also reflects our knowledge of back-end systems, data handling, and real-time interactions. Future enhancements could include more detailed performance analytics, AI-powered question generation, and social media sharing features.

Quiz Categories and Features:

The Quiz Application supports multiple categories and allows both users and admins to engage in quizzes based on predefined categories. Features include:

- **Quiz Creation:** Admins can create quizzes with questions, answer options, and timers.
- **User Interaction:** Users can attempt quizzes, receive instant feedback, and see their scores.
- **Leaderboard:** High scorers are displayed on a leaderboard to foster competition.
- **Admin Dashboard:** Admins can manage quizzes, users, and performance metrics.
- **Responsive Design:** The application is optimized for various devices to ensure a smooth experience.

Results and Insights:

- Users instantly get feedback on their answers with a score summary at the end of each quiz.
- The leaderboard displays the highest scorers to encourage competition.
- The application provides insights on the user's quiz performance, highlighting areas for improvement.

The Quiz Application combines web development skills with interactive features to create an engaging learning tool, aiming to improve knowledge assessment through technology.

Introduction

Background and Motivation

In recent years, the landscape of education and self-study has shifted dramatically toward digital and remote learning modalities. Educators and learners alike seek tools that are both engaging and effective. While classroom-based quizzes provide immediate feedback, they are constrained by scheduling and scalability. Similarly, existing online quiz platforms—though feature-rich—often require steep learning curves or incur subscription fees.

BrainBuzz was conceived to bridge these gaps by delivering a lightweight, accessible quiz application that educators can deploy without licensing hurdles and students can use across devices. By combining a timed quiz mechanism with a seamless administrator interface for content management, BrainBuzz aims to enhance knowledge retention through active recall and immediate feedback.

1.2 Objectives

The project pursues the following key objectives:

1. **User Engagement:** Provide an intuitive, responsive quiz interface with time-bound questions to foster concentration and simulate exam conditions.
2. **Content Flexibility:** Enable administrators to create and update question banks dynamically, supporting diverse quiz categories.
3. **Performance Tracking:** Persist user scores and display leaderboard rankings to motivate continuous improvement and healthy competition.
4. **Security and Reliability:** Implement robust authentication and data validation to protect user information and maintain application integrity.
5. **Ease of Deployment:** Leverage modern web technologies and minimal configuration so that institutions can host the platform with minimal overhead.

1.3 Glossary of Terms

- **Quiz Category:** A thematic grouping of questions (e.g., "Web Development", "General Knowledge").
- **Question Bank:** A repository of questions and associated answers stored in the backend database.
- **Timed Session:** A quiz attempt with a global timer (30 seconds per question by default) that limits the total duration of the session.
- **Leaderboard:** A ranking display of users sorted by cumulative score, showcasing top performers.
- **JWT (JSON Web Token):** A compact, URL-safe means of representing claims between two parties, used for stateless authentication.

Literature Survey

2.1 Commercial Quiz Platforms

Kahoot! and **Quizizz** are two of the most widely adopted commercial quiz platforms in educational settings. Kahoot! emphasizes real-time classroom engagement through live game sessions and leaderboard projections on large screens. Its strengths include intuitive game-based interfaces and cross-device compatibility, but it requires a paid subscription for advanced analytics and large-scale deployments. Quizizz provides asynchronous quiz delivery, detailed reports, and customizable memes for feedback. However, its dependence on external servers can introduce latency in regions with limited connectivity.

Socrative offers quizzes, quick polls, and exit tickets, focusing on simplicity and rapid question deployment. While it integrates seamlessly with learning management systems (LMS) like Canvas and Schoology, its feature set is relatively narrow, lacking robust customization for question types beyond multiple choice and true/false.

2.2 Open-Source Solutions

Moodle's Quiz Module is one of the most comprehensive open-source LMS quiz tools. It supports varied question types (e.g., multiple choice, short answer, calculated questions), detailed feedback, and conditional branching. However, setting up Moodle requires significant server resources, database configuration, and theme customization, presenting a steep learning curve for non-technical administrators.

OpenQuizEngine and **TAO** are other open-source engines that prioritize extensibility. They offer plugin architectures to add new question templates and reporting dashboards, but their nascent ecosystems limit the number of readily available extensions and community support compared to commercial offerings.

2.3 Academic Research on Digital Quizzing

Studies in cognitive psychology highlight the **testing effect**, wherein repeated active recall (quizzing) enhances long-term retention more effectively than passive review (Roediger & Butler, 2011). Timed quizzes, in particular, further improve retention by adding desirable difficulty (Bjork, 1994). Immediate feedback has also been shown to correct misconceptions before they become entrenched (Butler et al., 2007).

Research on **gamification**—applying game mechanics to non-game contexts—demonstrates increased motivation and engagement in learning environments (Deterding et al., 2011). However, excessive gamification can distract from core content; finding the right balance between challenge and fun is critical (Seaborn & Fels, 2015).

2.4 Gap Analysis and BrainBuzz Positioning

While commercial platforms offer polished interfaces and advanced analytics, they often come with subscription costs and limited offline support. Open-source LMS tools provide flexibility but require technical expertise and heavy infrastructure.

BrainBuzz positions itself to fill this gap by:

- **Lightweight Deployment:** A minimal-stack Node.js/MongoDB backend with no heavy LMS overhead.
- **Open Accessibility:** Core features available free of charge; optional self-hosting without vendor

lock-in.

- **Modular Extensibility:** Clear code structure for adding new question types or reporting modules.

By synthesizing active-recall principles, timed sessions, and straightforward administration, BrainBuzz aims to deliver an optimal balance between educational efficacy and operational simplicity.

3)Scope

The Scope section delineates the boundaries of the BrainBuzz Quiz App project by specifying which features, deliverables, and quality attributes are included, as well as constraints and success criteria.

3.1 Functional Scope

This project will implement the following core functional modules:

1. **User Management**:
    - Registration, login, and logout workflows for standard users.
    - Password validation, email uniqueness enforcement, and session persistence via JWT.
    - Profile editing interface (name, email, password updates).
2. **Quiz Engine**:
    - Dynamic retrieval and display of quiz categories.
    - Timed quiz sessions with 30-second per-question countdown and global session timer.
    - Randomized question order and option shuffling to prevent memorization patterns.
    - Immediate feedback after each answer selection (correct/incorrect indication).
3. **Score Handling**:
    - Real-time score calculation and display during and after quiz completion.
    - Persistence of cumulative scores per user in the backend database.
    - Leaderboard API to fetch and display top performers based on total accumulated score.
4. **Administrator Dashboard**:
    - Separate authentication for admins with role-based access control.
    - CRUD operations on quiz categories and questions (create, read, update, delete).
    - Bulk import/export of question banks via JSON.
5. **Contact & Feedback**:
    - Simple contact form for user feedback submission.
    - Logging of contact messages for future review via backend console logs or persistent storage.

3.2 Non-functional Scope

The application will adhere to the following quality attributes:

- **Performance:** Quiz questions and categories must load in under 300 ms under typical network conditions (100 kbps upload/download).
- **Security:** All API endpoints protected against common web vulnerabilities (e.g., SQL injection, XSS, CSRF). Passwords stored using bcrypt hashing (future implementation).
- **Usability:** Responsive UI compatible with desktop and mobile viewports, following WCAG AA accessibility guidelines.

- **Maintainability:** Modular code organization with clear separation of concerns (MVC on backend, component-based frontend).
- **Scalability:** Support for horizontal scaling of the backend via stateless API design.

3.3 Constraints and Assumptions

- **Timeline:** Four-month development cycle with bi-weekly sprints.
- **Budget:** Open-source libraries only; no paid third-party services.
- **Technical:** Hosting on standard Node.js-compatible cloud provider; MongoDB Atlas free tier.
- **Assumptions:** Users have modern browsers; stable internet connectivity; admin users have basic technical proficiency.

3.4 Success Criteria

The project will be deemed successful if:

- 95% of unit and integration tests pass across backend and frontend.
- 90% user satisfaction rating in post-release survey (ease of use, performance).
- Achievement of 300 ms average API response time for quiz-related endpoints.
- Zero critical security vulnerabilities identified in penetration testing.

4. System Requirements

The system requirements define the baseline hardware, software, and network conditions under which the BrainBuzz Quiz App will operate optimally. These are categorized into functional, environmental, and security considerations.

4.1 Hardware Requirements:

Client-Side (End-User Devices)

Component	Minimum Specification	Recommended Specification
Processor	1.2 GHz dual-core	2.0 GHz quad-core
RAM	1 GB	4 GB
Storage	100 MB free browser cache	SSD for faster access
Display	1024×768 resolution	1366×768 or higher

Server-Side (Deployment Machine)

Component	Minimum Specification	Recommended Specification
Processor	2 GHz dual-core	2.5 GHz quad-core or higher
RAM	2 GB	4–8 GB
Disk Storage	10 GB (project + MongoDB)	20 GB SSD
Network	1 Mbps bandwidth	10 Mbps for smoother response

Note: The application can be hosted on cloud platforms like Heroku, Vercel, or self-managed VPS services such as DigitalOcean or AWS EC2 instances.

4.2 Software Requirements

Backend Environment

- **Node.js**: v14 or later
- **npm/yarn**: npm v6+ or yarn v1.22+
- **MongoDB**: v4.4+ (local or MongoDB Atlas cloud database)
- **Express**: v4.18+
- **Mongoose**: for schema-based interaction with MongoDB

Frontend Environment

- **React.js**: v17 or later
- **React Router DOM**: v6
- **Axios**: HTTP request handler
- **Session Storage**: for JWT token caching
- **CSS Modules or Tailwind CSS (optional)**: for scoped styling

Developer Tools

- **VS Code** or JetBrains WebStorm
- **Postman/Insomnia** for API testing
- **Git + GitHub** for version control
- **Dotenv** for managing environment variables

4.3 Network Requirements

Parameter	Minimum Requirement	Description
Bandwidth	1 Mbps download/upload	Ensures quiz content loads without lag
Latency	<200 ms	Reduces delay in quiz transitions
Availability	99% uptime recommended	Especially critical for assessment periods
Port Access	Port 8000 (backend), 3000 (frontend)	Can be changed depending on deployment

4.4 Security Requirements

- **Authentication**: JWT-based stateless session tokens
- **Input Validation**: Server-side validation on all form fields
- **CORS Policy**: Restricted cross-origin access to allow only whitelisted domains
- **Data Protection**: Password hashing (bcrypt to be implemented)
- **HTTPS**: Mandatory on production with SSL certificates
- **Environment Variables**: All secrets stored securely in .env and never hardcoded

5) Methodology

The development of the Quiz Application followed a structured and systematic approach to ensure the project met its objectives of being interactive, scalable, and user-friendly. Below is a detailed breakdown of the methodology used, covering the entire lifecycle from planning to deployment.

1. Project Planning and Requirement Analysis

- **Objective Definition**: The first step involved clearly defining the goals of the Quiz Application. Key objectives included creating an engaging platform for users to take quizzes, providing real-time feedback, and enabling administrators to manage content efficiently.
- **Stakeholder Identification**: We identified two primary user groups:
    - **Users**: Individuals taking quizzes, tracking scores, and competing on the leaderboard.
    - **Administrators**: Responsible for creating and managing quiz content and monitoring user performance.
- **Functional Requirements**:
    - User registration and login with secure authentication.
    - Quiz category selection and question rendering.
    - Real-time scoring and leaderboard updates.
    - Admin dashboard for quiz management.
- **Non-Functional Requirements**:
    - Responsive design for accessibility across devices.
    - Fast performance with minimal load times.
    - Secure data handling and user privacy.

2. System Design

- **Architecture Design**: A client-server architecture was chosen, with the frontend built using React.js for dynamic user interactions and the backend powered by Node.js and MongoDB for data management.
- **Database Design**: MongoDB was selected for its flexibility in handling quiz data. Key collections included:
    - **Users**: Storing user credentials and scores.
    - **Quizzes**: Storing quiz categories and questions.
    - **Scores**: Tracking user performance.
- **UI/UX Design**: Wireframes and mockups were created to ensure an intuitive user experience. The design prioritized simplicity, with clear navigation between quiz categories, questions, and results.
- **Authentication Design**: JWT (JSON Web Tokens) was implemented for secure user and admin authentication, ensuring only authorized access to sensitive features.

3. Development Process

- **Agile Methodology**: The project followed an Agile approach with iterative sprints. Each sprint focused on delivering specific features, such as user authentication, quiz rendering, or admin controls.

- **Frontend Development**:
    - **React.js Components**: Reusable components like Quiz, Question, and Score were developed to handle different parts of the quiz-taking process.
    - **State Management**: React's useState and useEffect hooks managed local state and side effects, ensuring real-time updates without page reloads.
    - **API Integration**: Axios was used to make API calls to the backend for fetching quizzes and submitting scores.
- **Backend Development**:
    - **Node.js and Express**: RESTful APIs were created for user authentication, quiz management, and score tracking.
    - **MongoDB Integration**: Mongoose ORM was used to interact with the MongoDB database, ensuring efficient data operations.
    - **Authentication**: JWT tokens were generated upon login and verified for protected routes.
- **Version Control**: Git was used for version control, with regular commits and branches for feature development.

4. Testing

- **Unit Testing**: Individual components and functions were tested to ensure correctness. For example, the BMI calculation logic was unit-tested with various inputs.
- **Integration Testing**: End-to-end tests verified that frontend and backend components worked seamlessly together, such as submitting a quiz and updating the leaderboard.
- **User Acceptance Testing (UAT)**: A group of beta users tested the application, providing feedback on usability and identifying bugs.
- **Performance Testing**: Load times and responsiveness were tested across different devices and browsers to ensure optimal performance.

5. Deployment

- **Hosting**: The frontend was deployed on a platform like Netlify or Vercel, while the backend was hosted on Heroku or a similar service.
- **Database Hosting**: MongoDB Atlas was used for cloud-based database management, ensuring scalability and reliability.

6. Maintenance and Future Enhancements

- **Bug Fixes and Updates**: Post-deployment, the application was monitored for issues, with regular updates to fix bugs and improve performance.
- **Feature Expansion**: Plans for future enhancements include adding multiplayer quiz modes, detailed analytics, and integration with learning management systems.

.7) Code Implementation

Backend

- **Models:** Admin, UserLogin, Quiz schemas defined in Mongoose.
- **Routes:** Modular Express routers for authentication (/api/auth), quizzes (/api/quiz), scoring (/api/save-score), leaderboard (/api/leaderboard), and contact (/api/contact).
- **Security:** JWT tokens for session management; environment variables via dotenv; CORS enabled for cross-origin requests.
- **Server:** server.js bootstraps Express, connects to MongoDB Atlas, and mounts route modules.

CODE:

models/Admin.js:

```
const mongoose = require('mongoose');
const adminSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});
module.exports = mongoose.model('Admin', adminSchema);
```

models/Quiz.js:

```
const mongoose = require("mongoose");
const quizSchema = new mongoose.Schema({
  category: String,
  questions: [
    {
      text: String,
      options: [{ text: String, isCorrect: Boolean }],
    },
  ],
});
module.exports = mongoose.model("Quiz", quizSchema);
```

models/UserLogin.js:

```
const mongoose = require("mongoose");
const userLoginSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  score: { type: Number, default: 0 },
});
module.exports = mongoose.model("UserLogin", userLoginSchema);
```

routes/adminlogin.js:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const router = express.Router();
const Admin = require("../models/Admin");
const JWT_SECRET = process.env.ADMIN_JWT_SECRET || "secret";
router.post("/register", async (req, res) => {
  const { username, email, password } = req.body;
  try {
    const existingAdmin = await Admin.findOne({ $or: [{ email }, { username }] });
    if (existingAdmin) {
      return res.status(400).json({ message: "Admin already exists" });
    }
    const newAdmin = new Admin({ username, email, password });
    await newAdmin.save();
    res.status(201).json({ message: "Admin registered successfully" });
```

```
  } catch (err) {
    res.status(500).json({ message: "Server error" });
  }
});
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    const admin = await Admin.findOne({ email });
    if (!admin || admin.password !== password) {
      return res.status(401).json({ message: "Invalid credentials" });
    }
    const token = jwt.sign({ id: admin._id, role: "admin", email: admin.email }, JWT_SECRET, {
      expiresIn: "2h",
    });
    res.json({ token, email: admin.email, username: admin.username });
  } catch (err) {
    res.status(500).json({ message: "Server error" });
  }
});
module.exports = router;
```

routes/contact.js:

```
const express = require("express");
const router = express.Router();
router.post("/", (req, res) => {
  const { name, email, message } = req.body;
  res.json({ message: "Thank you for contacting us. We will get back to you soon." });
});
module.exports = router;
```

routes/leaderboard.js:

```
const express = require("express");
const UserLogin = require("../models/UserLogin");
const router = express.Router();
router.get("/", async (req, res) => {
  try {
    const leaderboard = await UserLogin.find()
      .sort({ score: -1 })
      .select("name score -_id");
    res.json(leaderboard);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
module.exports = router;
```

routes/quiz.js:

```
const express = require("express");
const Quiz = require("../models/Quiz");
const router = express.Router();
router.get("/categories", async (req, res) => {
  try {
    const categories = await Quiz.distinct("category");
    res.json(categories);
  } catch (err) {
    res.status(500).json({ message: "Server error" });
  }
});
router.get("/:category", async (req, res) => {
  try {
    const categoryParam = req.params.category.trim();
    const quizzes = await Quiz.find({
      category: { $regex: new RegExp(`^${categoryParam}$`, "i") },
    });
    if (!quizzes.length) {
      return res.status(404).json({ message: "No quizzes found" });
    }
    const questions = quizzes.flatMap((quiz) => quiz.questions);
```

```
    res.json(questions);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
router.post("/add-quiz", async (req, res) => {
  try {
    const { category, questions } = req.body;
    if (!category || !Array.isArray(questions) || questions.length === 0) {
      return res.status(400).json({ message: "Invalid quiz data" });
    }
    const formattedQuestions = questions.map(q => ({
      text: q.question,
      options: q.options.map(opt => ({
        text: opt,
        isCorrect: opt.trim().toLowerCase() === q.correctAnswer.trim().toLowerCase()
      }))
    }));
    let existingQuiz = await Quiz.findOne({ category });
    if (existingQuiz) {
      existingQuiz.questions.push(...formattedQuestions);
      await existingQuiz.save();
    } else {
      await Quiz.create({ category, questions: formattedQuestions });
    }
    res.status(201).json({ message: "Quiz saved successfully" });
  } catch (err) {
    res.status(500).json({ message: "Server error" });
  }
});
module.exports = router;
```

routes/score.js:

```
const express = require("express");
const UserLogin = require("../models/UserLogin");
const router = express.Router();
router.post("/", async (req, res) => {
  try {
    const { email, score } = req.body;
    const user = await UserLogin.findOne({ email });
    if (!user) return res.status(404).json({ message: "User not found" });
    user.score = (user.score || 0) + score;
    await user.save();
    res.json({ message: "Score updated", score: user.score });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
module.exports = router;
```

routes/userlogin.js:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const UserLogin = require("../models/UserLogin");
const router = express.Router();
const JWT_SECRET = process.env.JWT_SECRET;
router.post("/register", async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const exists = await UserLogin.findOne({ email });
    if (exists) return res.status(400).json({ message: "User already exists" });
    const newUser = new UserLogin({ name, email, password });
    await newUser.save();
    res.status(201).json({ message: "User registered successfully!" });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await UserLogin.findOne({ email });
    if (!user || user.password !== password)
      return res.status(400).json({ message: "Invalid credentials" });
    const token = jwt.sign(
      { userId: user._id, name: user.name, email: user.email },
      JWT_SECRET,
      { expiresIn: "1h" }
    );
    res.json({ token, name: user.name, email: user.email });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
module.exports = router;
```

.env:

```
MONGO_URI=mongodb+srv://sai:sai123@quizapp.g2t9neo.mongodb.net/?retryWrites=true&w=majority&appName=quizapp
JWT_SECRET=your_jwt_secret_key
PORT=8000
```

server.js:

```
require('dotenv').config();
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const AuthRoutes = require("./routes/userlogin");
const AdminAuthRoutes = require("./routes/adminlogin");
const QuizRoutes = require("./routes/quiz");
const ScoreRoutes = require("./routes/score");
const LeaderboardRoutes = require("./routes/leaderboard");
const ContactRoutes = require("./routes/contact");
const app = express();
app.use(cors());
app.use(express.json());
app.use("/api/auth", AuthRoutes);
app.use("/api/auth/admin", AdminAuthRoutes);
app.use("/api/quiz", QuizRoutes);
app.use("/api/save-score", ScoreRoutes);
app.use("/api/leaderboard", LeaderboardRoutes);
app.use("/api/contact", ContactRoutes);
mongoose
  .connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("MongoDB Atlas connected"))
  .catch((err) => console.log("MongoDB connection error:", err));
app.get("/", (req, res) => {
  res.send("Backend is running");
});
const PORT = process.env.PORT || 8000;
app.listen(PORT, () =>
  console.log(`Server is running on port ${PORT}`)
);
```

Frontend

- **Router:** React Router v6 defines public (/, /admin) and protected (/home, /quiz/:category) routes.
- **State Management:** Local state hooks and sessionStorage to persist tokens and user info.
- **Components:** Reusable UI elements (AuthForm, Navbar, Quiz, CategoryCard, Score).
- **Pages:** Feature pages for Home, Quiz, Profile, Leaderboard, Contact, AdminDashboard, and AdminCreateQuiz.

- **Styling:** CSS modules following BEM conventions for maintainability.

Components:

// AdminAuthForm.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import '../css/AdminAuthForm.css';
const AdminAuthForm = () => {
  const [adminData, setAdminData] = useState({ username: '', email: '', password: '' });
  const [isRegister, setIsRegister] = useState(false);
  const [output, setOutput] = useState('');
  const navigate = useNavigate();
  const handleSubmit = async e => {
    e.preventDefault();
    const endpoint = isRegister ? 'register' : 'login';
    try {
      const res = await axios.post(`http://localhost:8000/api/auth/admin/${endpoint}`, adminData);
      if (!isRegister) {
        sessionStorage.setItem('adminToken', res.data.token);
        setOutput('Admin login successful');
        setTimeout(() => window.location.reload(), 500);
      } else {
        setOutput('Admin registered successfully. You can now login.');
        setIsRegister(false);
      }
    } catch (err) {
      setOutput(err.response?.data?.message || 'Operation failed');
    }
  };
  return (
    <div className="admin-auth-container">
      <form onSubmit={handleSubmit} className="admin-auth-form">
        <h2>{isRegister ? 'Admin Register' : 'Admin Login'}</h2>
        {isRegister && (
          <input type="text" placeholder="Username" value={adminData.username}
    onChange={e => setAdminData({ ...adminData, username: e.target.value })}
            required
          />
        )}
        <input
          type="email"
          placeholder="Email"
          value={adminData.email}
          onChange={e => setAdminData({ ...adminData, email: e.target.value })}
          required
        />
        <input
          type="password"
          placeholder="Password"
          value={adminData.password}
          onChange={e => setAdminData({ ...adminData, password: e.target.value })}
          required
        />
        <button type="submit">{isRegister ? 'Register as Admin' : 'Login as Admin'}</button>
        <p onClick={() => setIsRegister(!isRegister)} className="admin-toggle-link">
          {isRegister ? 'Already have an account? Login' : 'No account? Register as Admin'}
        </p>
        {output && <p className="admin-output">{output}</p>}
      </form>
    </div>
  );
```

```
};
export default AdminAuthForm;

// AdminRegisterForm.jsx
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
const AdminRegisterForm = () => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({ username: "", password: "" });
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await fetch("http://localhost:8000/api/auth/admin/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(formData),
      });
      const data = await res.json();
      if (res.ok) {
        alert("Admin registered! You can now login.");
        navigate("/admin");
      } else {
        alert(data.message || "Registration failed");
      }
    } catch (err) {
      console.error(err);
      alert("Something went wrong.");
    }
  };
  return (
    <div className="min-h-screen flex flex-col items-center justify-center bg-gray-100 p-4">
      <div className="bg-white p-6 rounded-xl shadow-md w-full max-w-md">
        <h2 className="text-2xl font-bold mb-4 text-center">Admin Register</h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <input
            type="text"
            name="username"
            value={formData.username}
            onChange={handleChange}
            placeholder="Username"
            className="w-full px-4 py-2 border rounded-lg"
            required
          />
          <input
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
            placeholder="Password"
            className="w-full px-4 py-2 border rounded-lg"
            required
          />
          <button
            type="submit"
            className="w-full bg-green-500 text-white py-2 rounded-lg hover:bg-green-600 transition"
          >
            Register
          </button>
        </form>
        <div className="text-center mt-4">
          <p className="text-sm text-gray-500">Already an admin?</p>
```

```
          <button
            onClick={() => navigate("/admin")}
            className="mt-2 text-blue-500 hover:underline"
          >
            Go to Login
          </button>
        </div>
      </div>
    </div>
  );
};
export default AdminRegisterForm;
```

// category.jsx

```
import React from "react";
import "../css/category.css";
const Categorycard = ({ category, icon, onClick }) => (
  <div className="category-card" onClick={() => onClick(category)}>
    <img src={icon} alt={category} className="category-icon" />
    <p>{category}</p>
  </div>
);
export default Categorycard;
```

// Layout.jsx

```
import React, { useEffect, useState } from "react";
import Navbar from "./Navbar";
import { Outlet, useNavigate } from "react-router-dom";
const Layout = () => {
  const navigate = useNavigate();
  const [auth, setAuth] = useState(!!sessionStorage.getItem("token"));
  useEffect(() => {
    const checkAuth = () => setAuth(!!sessionStorage.getItem("token"));
    window.addEventListener("storage", checkAuth);
    return () => window.removeEventListener("storage", checkAuth);
  }, []);
  useEffect(() => {
    if (!auth) navigate("/", { replace: true });
  }, [auth, navigate]);
  return (
    <div>
      <Navbar />
      {auth && <Outlet />}
    </div>
  );
};
export default Layout;
```

// loginpage.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "../css/loginform.css";
import axios from "axios";
const AuthForm = ({ setAuth }) => {
  const [isActive, setIsActive] = useState(false);
  const navigate = useNavigate();
  const [loginData, setLoginData] = useState({ email: "", password: "" });
  const [loginOutput, setLoginOutput] = useState("");
  const [registerData, setRegisterData] = useState({ name: "", email: "", password: "" });
  const [registerOutput, setRegisterOutput] = useState("");
  const handleToggle = () => setIsActive(!isActive);
  const handleLogin = async e => {
    e.preventDefault();
    try {
```

```
    const res = await axios.post("http://localhost:8000/api/auth/login", loginData, {
      headers: { "Content-Type": "application/json" }
    });
    sessionStorage.setItem("token", res.data.token);
    sessionStorage.setItem("username", res.data.name);
    sessionStorage.setItem("email", res.data.email);
    setLoginOutput("Login Successful");
    setTimeout(() => {
      setAuth(true);
      navigate("/home", { replace: true });
    }, 1000);
  } catch (err) {
    setLoginOutput(err.response?.data?.message || "Invalid Credentials");
  }
};
const handleRegister = async e => {
  e.preventDefault();
  if (registerData.password.length < 8) {
    setRegisterOutput("Password length must be at least 8 characters");
    return;
  }
  try {
    const res = await axios.post("http://localhost:8000/api/auth/register", registerData, {
      headers: { "Content-Type": "application/json" }
    });
    if (res.status === 201) {
      setRegisterOutput("Registration Successful");
      setTimeout(() => setIsActive(false), 2000);
    }
  } catch (err) {
    setRegisterOutput(err.response?.data?.message || "Signup failed. Try again!");
  }
};
return (
  <div className={`container ${isActive ? "active" : ""}`}>
    <div className="form-box login">
      <form onSubmit={handleLogin}>
        <h1>Login</h1>
        <div className="input-box">
          <input
            type="email"
            placeholder="Email"
            value={loginData.email}
            onChange={e => setLoginData({ ...loginData, email: e.target.value })}
            required
          />
          <i className="bx bxs-user"></i>
        </div>
        <div className="input-box">
          <input
            type="password"
            placeholder="Password"
            value={loginData.password}
            onChange={e => setLoginData({ ...loginData, password: e.target.value })}
            required
          />
          <i className="bx bxs-lock-alt"></i>
        </div>
        <div className="forgot-link"><a href="#">Forgot password?</a></div>
        <button type="submit" className="btn">Login</button>
        <p id="loutput">{loginOutput}</p>
        <p className="text-sm text-gray-500">Are you an Admin?</p>
        <button
          onClick={() => navigate("/admin")}
```

```
            className="mt-2 btn admin-btn"
          >
            Go to Admin Login
          </button>
        </form>
      </div>
      <div className="form-box register">
        <form onSubmit={handleRegister}>
          <h1>Registration</h1>
          <div className="input-box">
            <input
              type="text"
              placeholder="Username"
              value={registerData.name}
              onChange={e => setRegisterData({ ...registerData, name: e.target.value })}
              required
            />
            <i className="bx bxs-user"></i>
          </div>
          <div className="input-box">
            <input
              type="email"
              placeholder="Email"
              value={registerData.email}
              onChange={e => setRegisterData({ ...registerData, email: e.target.value })}
              required
            />
            <i className="bx bxs-envelope"></i>
          </div>
          <div className="input-box">
            <input
              type="password"
              placeholder="Password"
              value={registerData.password}
              onChange={e => setRegisterData({ ...registerData, password: e.target.value })}
              required
            />
            <i className="bx bxs-lock-alt"></i>
          </div>
          <button type="submit" className="btn">Register</button>
          <p id="routput">{registerOutput}</p>
        </form>
      </div>
      <div className="toggle-box">
        <div className="toggle-panel toggle-left">
          <h1>Hello, Welcome!</h1>
          <p>Don't have an Account?</p>
          <button className="btn register-btn" onClick={handleToggle}>Register</button>
        </div>
        <div className="toggle-panel toggle-right">
          <h1>Welcome Back!</h1>
          <p>Already have an Account?</p>
          <button className="btn login-btn" onClick={handleToggle}>Login</button>
        </div>
      </div>
    </div>
  );
};
export default AuthForm;

// Navbar.jsx
import React from "react";
import "../css/navbar.css";
import { useNavigate, Link } from "react-router-dom";
```

```
import brainbuzzLogo from "../assets/brainbuzzlogo.png";
const Navbar = () => {
  const navigate = useNavigate();
  const handleLogOut = () => {
    sessionStorage.clear();
    navigate("/", { replace: true });
    window.location.reload();
  };
  return (
    <nav>
      <div className="nav-container">
        <div className="Logo">
          <img src={brainbuzzLogo} alt="BrainBuzz Logo" className="logo-img" />
        </div>
        <ul>
          <li className="items"><Link to="/home">Home</Link></li>
          <li className="items"><Link to="/profile">Profile</Link></li>
          <li className="items"><Link to="/leaderboard">Leaderboard</Link></li>
          <li className="items"><Link to="/contact">Contact</Link></li>
        </ul>
        <button onClick={handleLogOut}>Logout</button>
      </div>
    </nav>
  );
};
export default Navbar;

// Quiz.jsx
import React from "react";
import "../css/quiz.css";
const Quiz = ({ category, question, timeLeft, onAnswer }) => (
  <div className="quiz-container">
    <div className="quiz-header">
      <h2 className="quiz-category">{category}</h2>
      <div className="quiz-timer"><span>{timeLeft}s</span></div>
    </div>
    <div className="quiz-question-box">
      <p>{question?.text}</p>
    </div>
    <div className="quiz-options">
      {question?.options.map((opt, idx) => (
        <button
          key={idx}
          onClick={() => onAnswer(opt.isCorrect)}
          className="quiz-option-btn"
        >
          {opt.text}
        </button>
      ))}
    </div>
  </div>
);
export default Quiz;

// Score.jsx
import React from "react";
import "../css/quiz.css";
const Quiz = ({ category, question, timeLeft, onAnswer }) => (
  <div className="quiz-container">
    <div className="quiz-header">
      <h2 className="quiz-category">{category}</h2>
      <div className="quiz-timer"><span>{timeLeft}s</span></div>
    </div>
    <div className="quiz-question-box">
```

```
      <p>{question?.text}</p>
    </div>
    <div className="quiz-options">
      {question?.options.map((opt, idx) => (
        <button
          key={idx}
          onClick={() => onAnswer(opt.isCorrect)}
          className="quiz-option-btn"
        >
          {opt.text}
        </button>
      ))}
    </div>
  </div>
);
export default Quiz;

// AdminCreateQuiz.jsx
import React, { useState } from 'react';
import '../css/createquiz.css';
const AdminCreateQuiz = () => {
  const [category, setCategory] = useState('');
  const [questions, setQuestions] = useState([
    {
      question: '',
      options: ['', '', '', ''],
      correctAnswer: '',
    },
  ]);
  const handleQuestionChange = (index, field, value) => {
    const updatedQuestions = [...questions];
    if (field === 'question' || field === 'correctAnswer') {
      updatedQuestions[index][field] = value;
    } else {
      updatedQuestions[index].options[field] = value;
    }
    setQuestions(updatedQuestions);
  };
  const addQuestion = () => {
    setQuestions([
      ...questions,
      {
        question: '',
        options: ['', '', '', ''],
        correctAnswer: '',
      },
    ]);
  };
  const removeQuestion = (index) => {
    const updatedQuestions = questions.filter((_, i) => i !== index);
    setQuestions(updatedQuestions);
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    const quizData = {
      category,
      questions,
    };
    try {
      const response = await fetch('http://localhost:8000/api/quiz/add-quiz', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(quizData),
      });
```

```
    const data = await response.json();
    if (response.ok) {
      alert('✅ Quiz added successfully!');
      setCategory('');
      setQuestions([
        {
          question: '',
          options: ['', '', '', ''],
          correctAnswer: '',
        },
      ]);
    } else {
      alert(data.message || '❌ Error adding quiz.');
    }
  } catch (error) {
    console.error('❌ Error adding quiz:', error);
    alert('Error adding quiz. Please check console for details.');
  }
};
return (
  <div className="quiz-container">
    <h2 className="quiz-heading">Create New Quiz</h2>
    <form className="quiz-form" onSubmit={handleSubmit}>
      <div className="form-group">
        <label>Category Name</label>
        <input
          type="text"
          value={category}
          onChange={(e) => setCategory(e.target.value)}
          required
          placeholder="Enter quiz category"
        />
      </div>
      {questions.map((q, idx) => (
        <div className="question-block" key={idx}>
          <h3>Question {idx + 1}</h3>
          <input
            type="text"
            placeholder="Enter question"
            value={q.question}
            onChange={(e) => handleQuestionChange(idx, 'question', e.target.value)}
            required
          />
          <div className="options-container">
            {q.options.map((opt, i) => (
              <input
                key={i}
                type="text"
                placeholder={`Option ${i + 1}`}
                value={opt}
                onChange={(e) => handleQuestionChange(idx, i, e.target.value)}
                required
              />
            ))}
          </div>
          <input
            type="text"
            placeholder="Correct answer"
            value={q.correctAnswer}
            onChange={(e) => handleQuestionChange(idx, 'correctAnswer', e.target.value)}
            required
          />
          {questions.length > 1 && (
            <button
```

```
            type="button"
            className="remove-btn"
            onClick={() => removeQuestion(idx)}
          >
            🗑 Remove Question
          </button>
        )}
      </div>
    ))}
    <button type="button" className="add-btn" onClick={addQuestion}>
      ➕ Add New Question
    </button>
    <button type="submit" className="submit-btn">
      ✅ Submit Quiz
    </button>
  </form>
 </div>
);
};
export default AdminCreateQuiz;
```

AdmnDashboard.jsx:

```
import React, { useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "../css/AdminDashboard.css";
const AdminDashboard = () => {
 const navigate = useNavigate();
 const adminName = sessionStorage.getItem("adminUsername") || "Admin";
 useEffect(() => {
  if (!sessionStorage.getItem("adminToken")) {
   navigate("/admin", { replace: true });
  }
 }, [navigate]);
 const handleLogout = () => {
  sessionStorage.removeItem("adminToken");
  sessionStorage.removeItem("adminUsername");
  navigate("/", { replace: true });
  window.location.reload();
 };
 return (
  <div className="admin-dashboard-container">
   <header className="admin-header">
    <h1 className="admin-title">Admin Dashboard</h1>
    <button className="logout-button" onClick={handleLogout}>
     Logout
    </button>
   </header>
   <h2 className="admin-welcome">welcome {adminName}!</h2>
   <div className="admin-actions">
    <button
     className="admin-button"
     onClick={() => navigate("/admin/create-quiz")}
    >
     Add Questions
    </button>
    <button
     className="admin-button"
     onClick={() => navigate("/leaderboard")}
    >
     LeaderBoard
    </button>
   </div>
  </div>
 );
```

```
};
export default AdminDashboard;
```

Contact.jsx:

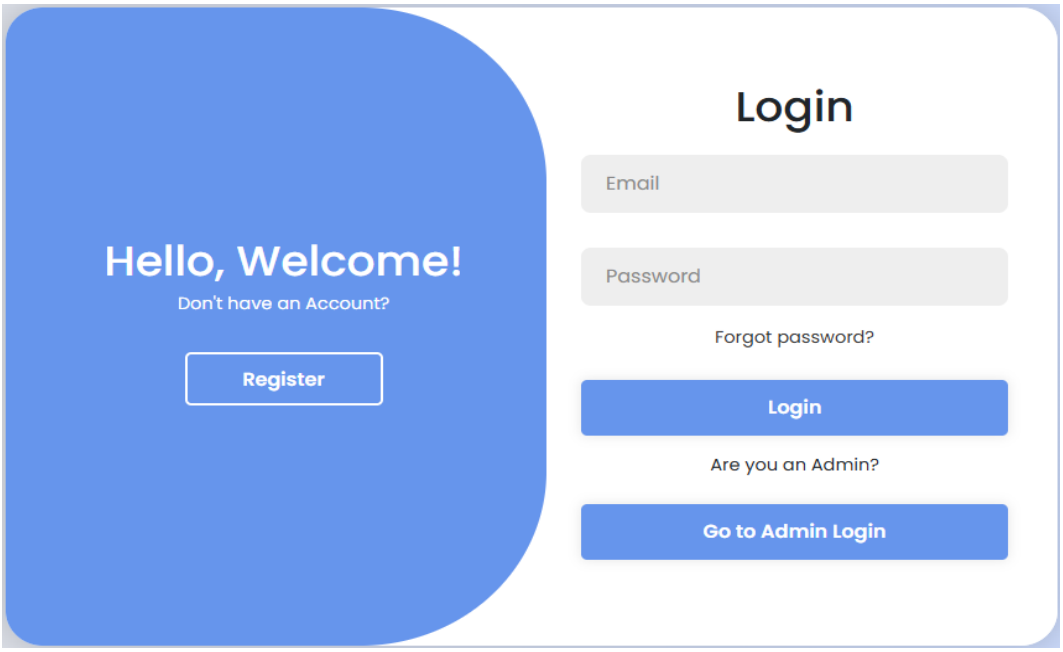
```
import React, { useState } from "react";
import axios from "axios";
import "../css/contact.css";
const Contact = () => {
  const [formData, setFormData] = useState({ name: "", email: "", message: "" });
  const [responseMsg, setResponseMsg] = useState("");
  const handleChange = e => setFormData({...formData, [e.target.name]: e.target.value});
  const handleSubmit = e => {
    e.preventDefault();
    axios.post("http://localhost:8000/api/contact", formData)
      .then(res => {
        setResponseMsg(res.data.message);
        setFormData({ name: "", email: "", message: "" });
      })
      .catch(() => setResponseMsg("There was an error sending your message."));
  };
  return (
    <div className="contact-container">
      <h1>Contact</h1>
      <form onSubmit={handleSubmit} className="contact-form">
        <label>Name:<input name="name" value={formData.name} onChange={handleChange} required /></label>
        <label>Email:<input name="email" type="email" value={formData.email} onChange={handleChange} required /></label>
        <label>Message:<textarea name="message" value={formData.message} onChange={handleChange} required/></label>
        <button type="submit" className="btn">Send Message</button>
      </form>
      {responseMsg && <p className="response-message">{responseMsg}</p>}
    </div>
  );
};
export default Contact;
```

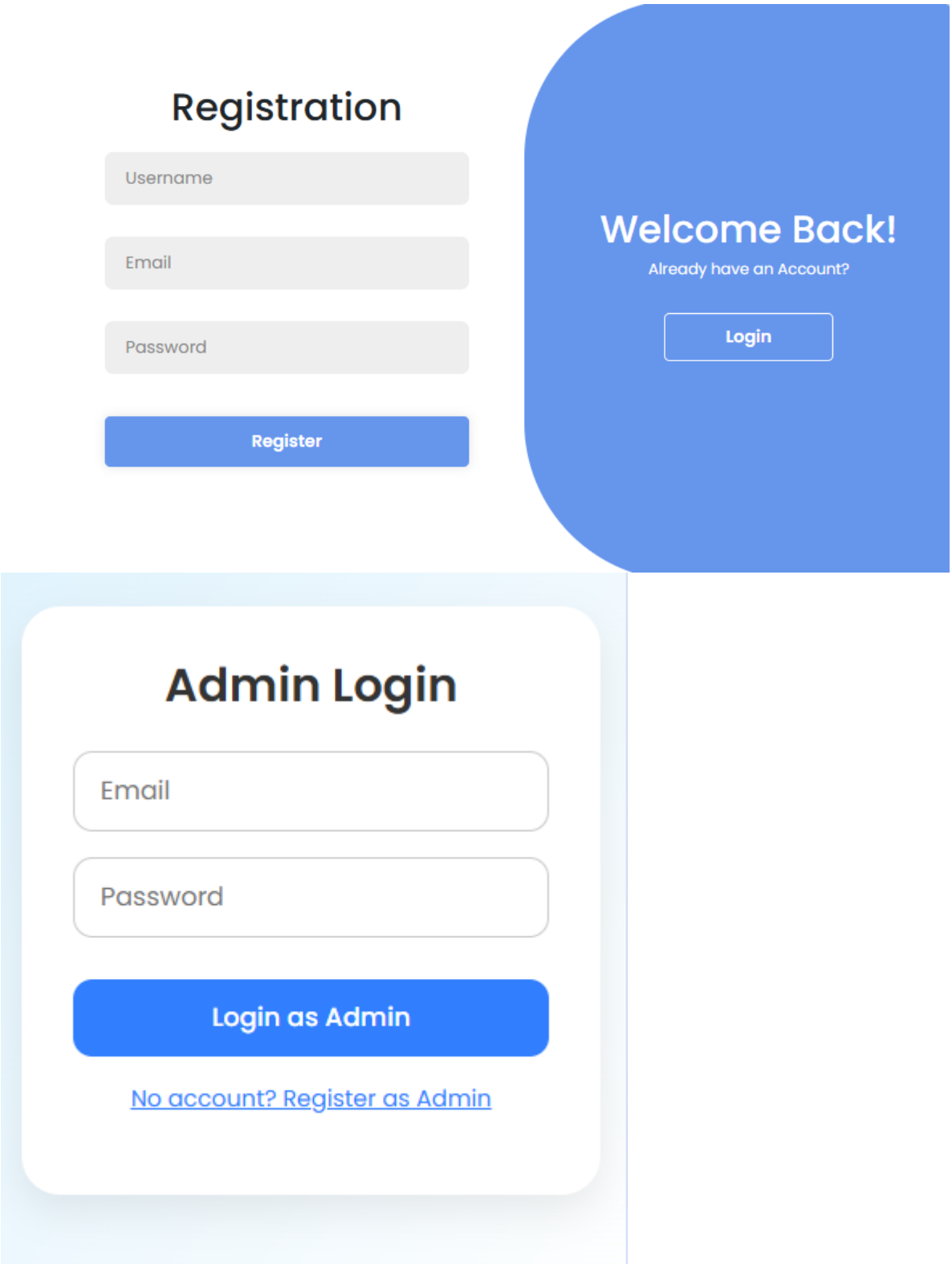
Home.jsx:

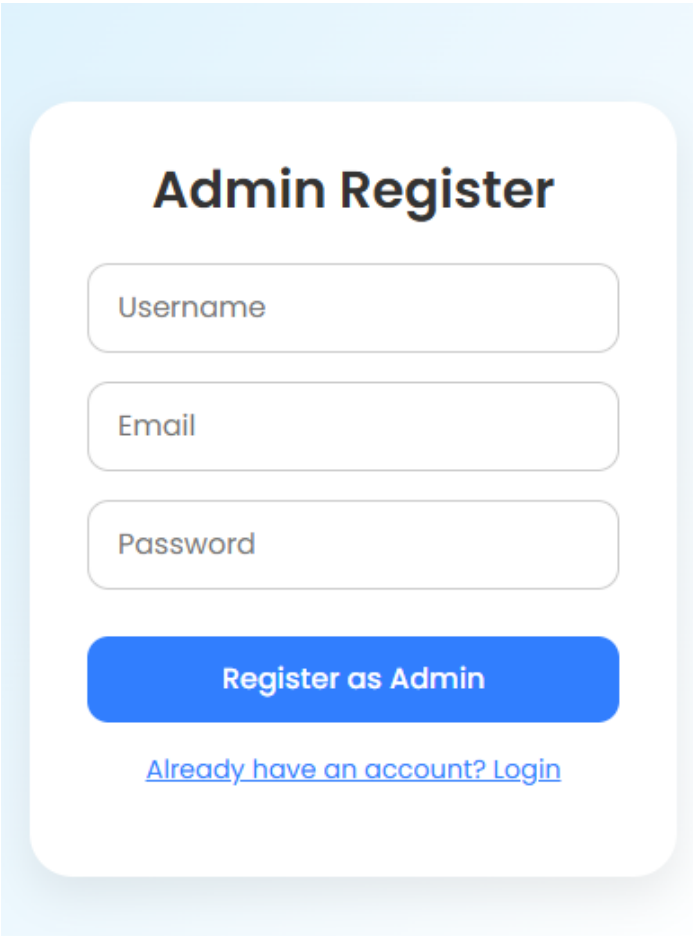
```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import Categorycard from "../components/category";
import "../css/home.css";
import axios from "axios";
import webdevIcon from "../assets/webdev.png";
import pythonIcon from "../assets/python.png";
import scienceIcon from "../assets/science.png";
import socialIcon from "../assets/social.png";
import gkIcon from "../assets/gk.png";
import funnyIcon from "../assets/funny.png";
import politicalIcon from "../assets/political.png";
const defaultCategories = {
  WebDev: webdevIcon,
  Python: pythonIcon,
  Science: scienceIcon,
  Social: socialIcon,
  GK: gkIcon,
  Funny: funnyIcon,
  Political: politicalIcon,
};
const Home = () => {
  const navigate = useNavigate();
  const username = sessionStorage.getItem("username") || "User";
  const [categories, setCategories] = useState([]);
  useEffect(() => {
    axios.get("http://localhost:8000/api/quiz/categories")
      .then(res => {
```

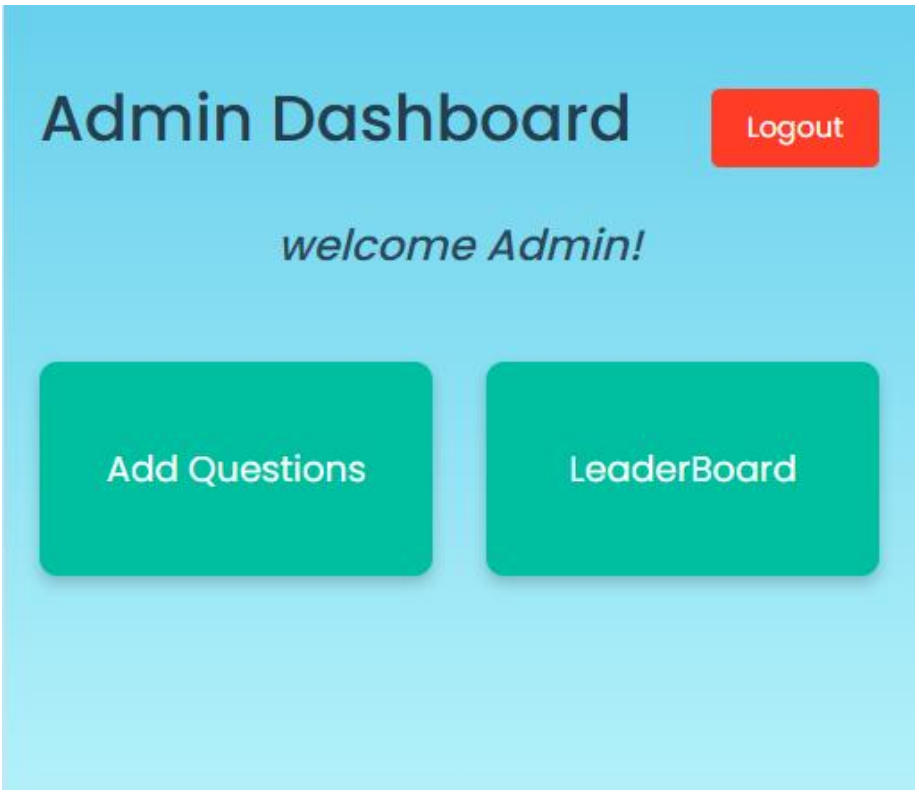
```
        setCategories(res.data);
      })
      .catch(err => console.error("❌ Failed to load categories:", err));
  }, []);
  const handleCategoryClick = (category) => {
    navigate(`/quiz/${category}`);
  };
  return (
    <div className="home-container">
      <h1 className="welcome-title">
        Welcome <span className="highlighted">{username}</span>!
      </h1>
      <p className="subtitle">Select a category</p>
      <div className="category-list">
        {categories.map((cat) => (
          <Categorycard
            key={cat}
            category={cat}
            icon={defaultCategories[cat] || null}
            onClick={handleCategoryClick}
          />
        ))}
      </div>
    </div>
  );
};
export default Home;
```

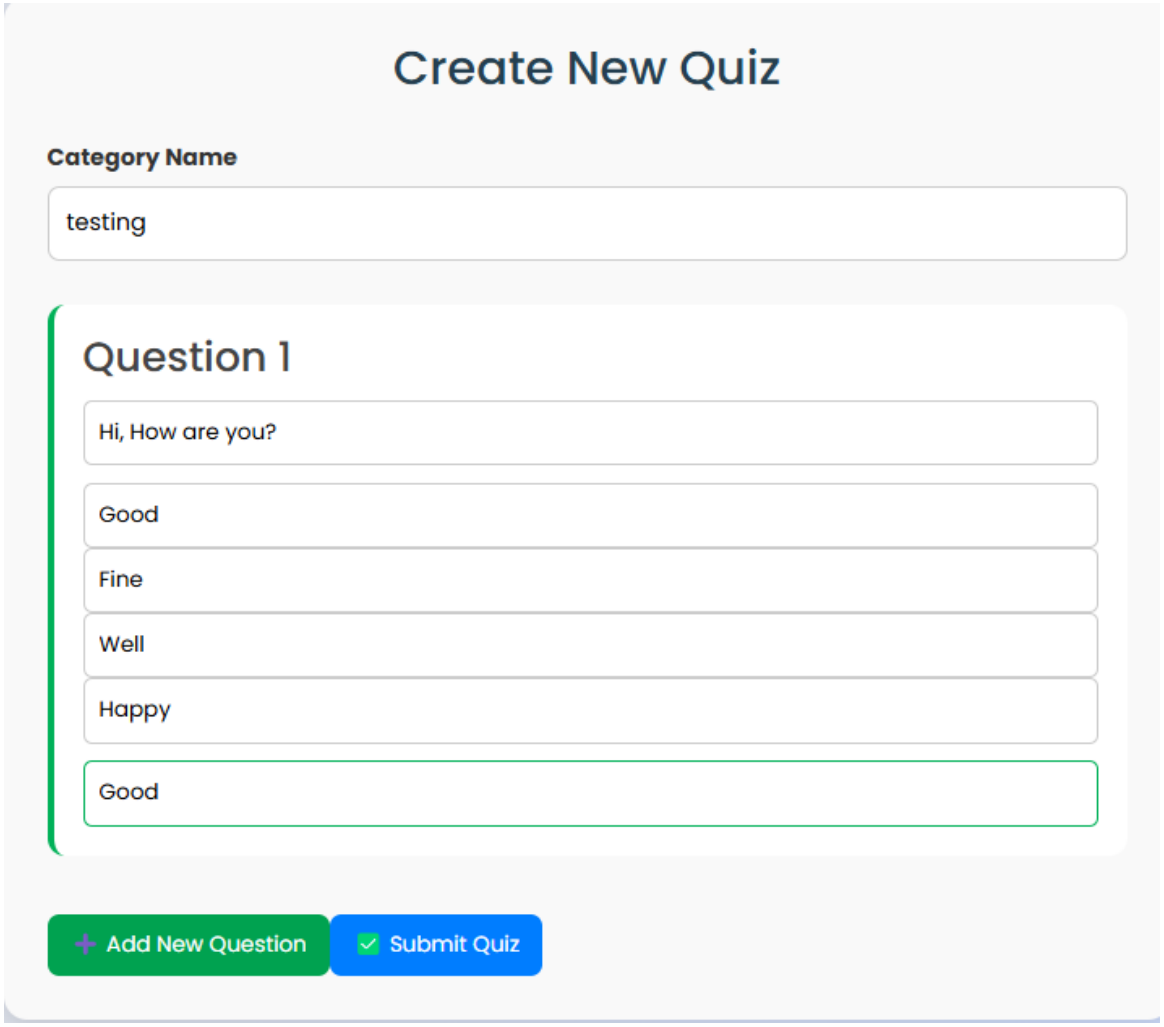
8) Result

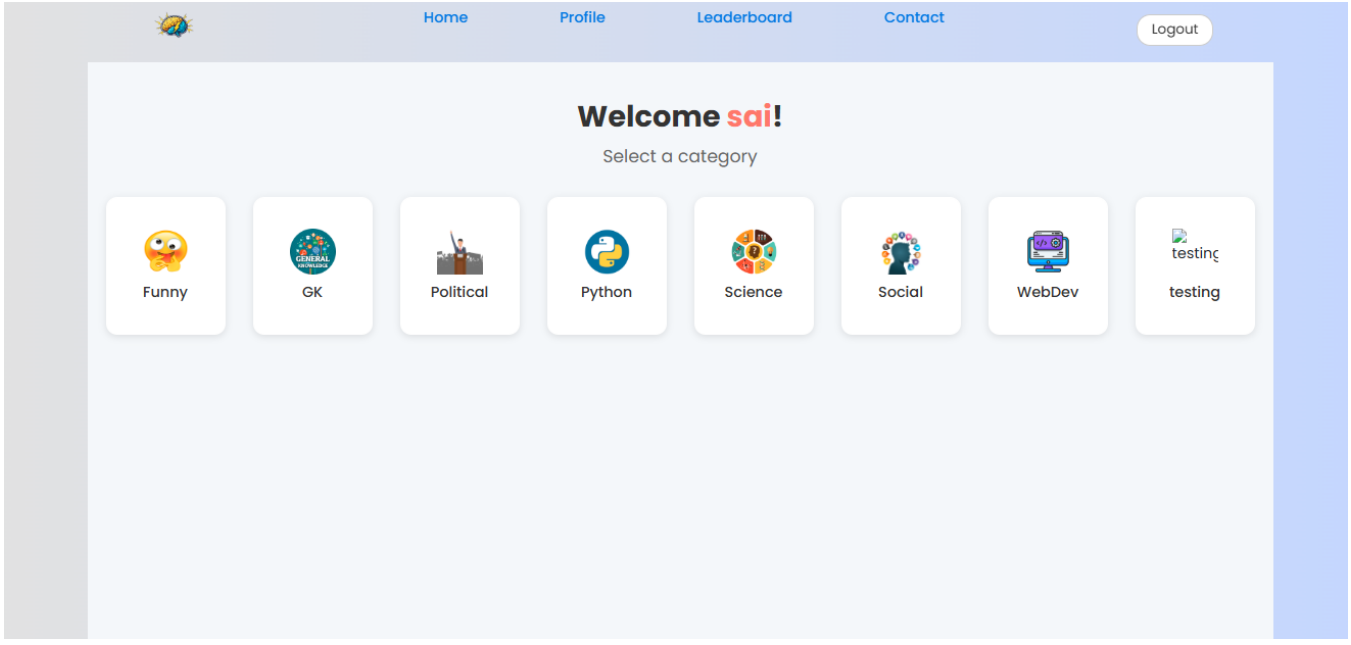


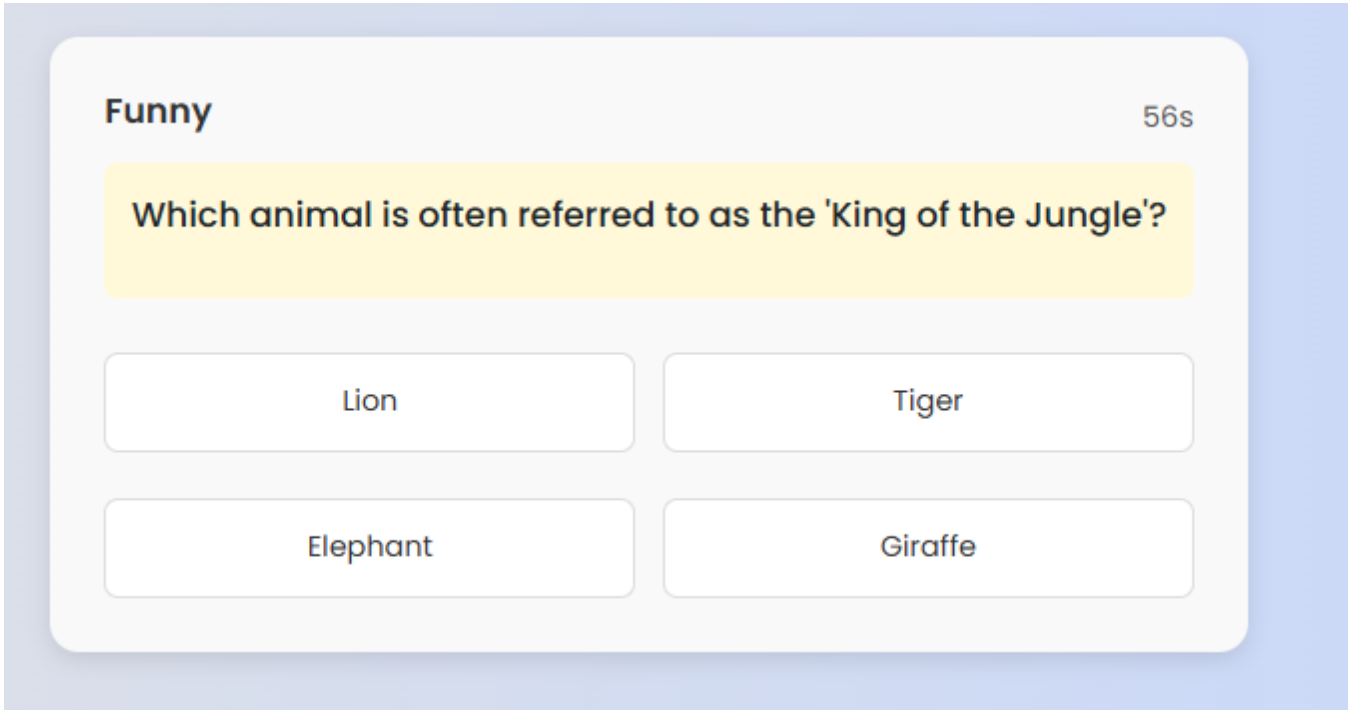

Registration
Username
Email
Password
Register
Welcome Back!
Already have an Account?
Login
Admin Login
Email
Password
Login as Admin
No account? Register as Admin

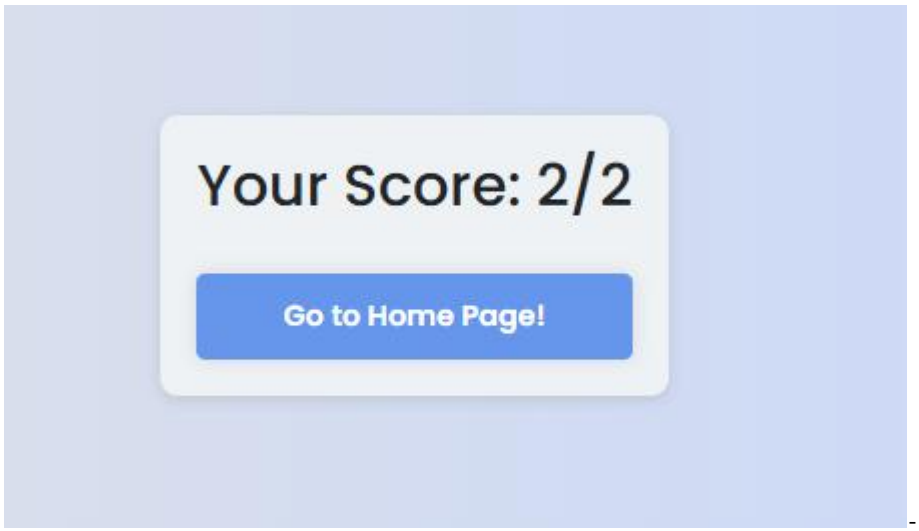

Admin Register
Username
Email
Password
Register as Admin
Already have an account? Login

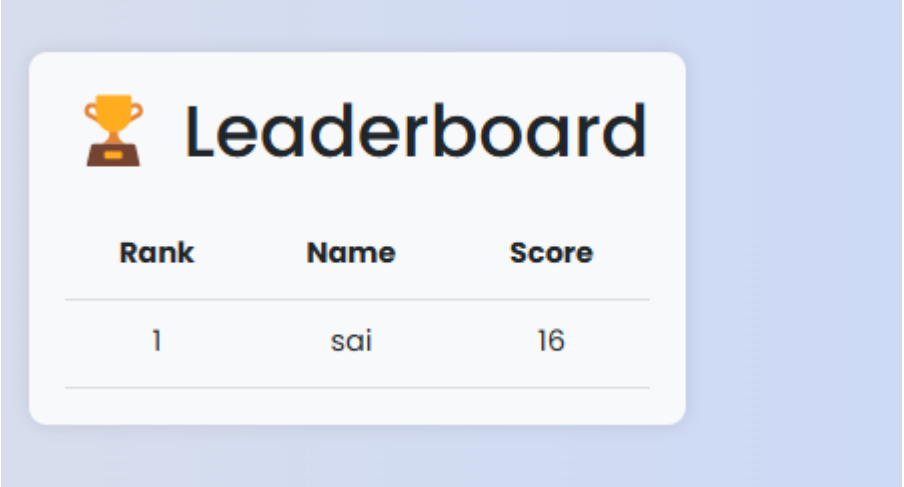

Admin Dashboard
Logout
welcome Admin!
Add Questions
LeaderBoard

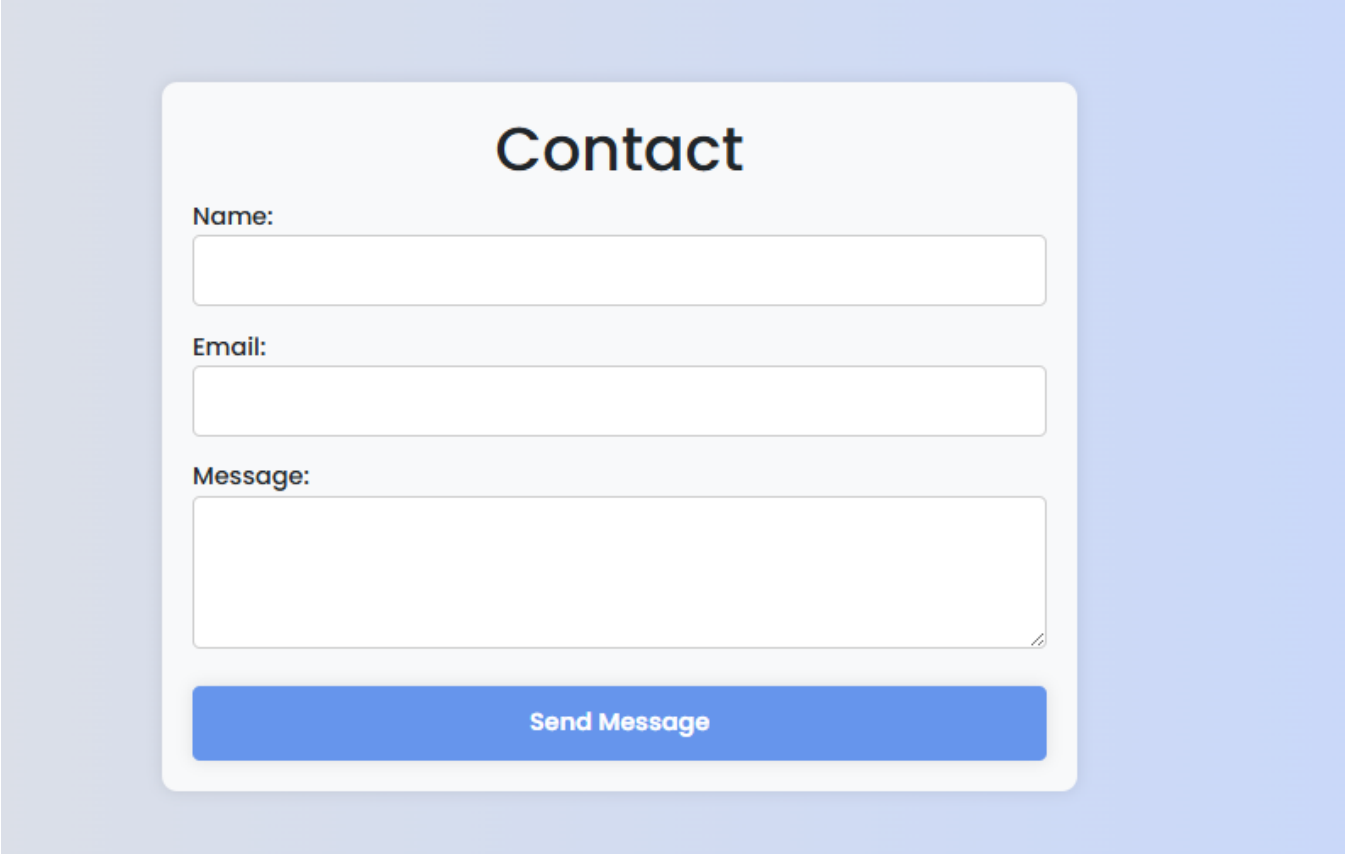

Create New Quiz
Category Name
testing
Question 1
Hi, How are you?
Good
Fine
Well
Happy
Good
Add New Question
Submit Quiz


Home
Profile
Leaderboard
Contact
Logout
Welcome sai!
Select a category
Funny
GK
Political
Python
Science
Social
WebDev
testing


Funny
56s
Which animal is often referred to as the 'King of the Jungle'?
Lion
Tiger
Elephant
Giraffe


Your Score: 2/2
Go to Home Page!





9) Future Enhancements:

- **Advanced Analytics:** Dashboards showing per-category performance, time taken per question.
- **Additional Question Types:** True/false, fill-in-the-blank, image-based questions.
- **Responsive Design:** Mobile-first layout optimizations and offline support via service workers.
- **Social Features:** Sharing results on social media and friend challenges.
- **Deployment Automation:** CI/CD with Docker and Kubernetes for scalable hosting.

10) Conclusion

The Quiz Application project has been a rewarding journey, reflecting my passion for web development and education as a student. By building this platform from scratch using React.js, Node.js, and MongoDB, I successfully created an interactive and user-friendly tool that enables users to take quizzes, track scores, and compete on a leaderboard, while administrators manage content efficiently. This project not only honed my technical skills in full-stack development but also taught me problem-solving, time management, and the importance of user-centric design.

Overcoming challenges like integrating real-time scoring and securing user authentication deepened my understanding of modern web technologies. The positive feedback from peers who tested the application, coupled with its reliable performance across devices, affirmed the project's success. This experience has fueled my enthusiasm for creating technology-driven solutions that make learning engaging and accessible.

The Quiz Application lays a strong foundation for future enhancements, such as multiplayer modes, mobile app development, and advanced analytics. As a personal endeavor driven by my curiosity and interest, this project showcases my ability to transform ideas into functional applications. I am excited to continue exploring the intersection of technology and education, using the skills and insights gained to develop innovative solutions that inspire and educate.