**SSN COLLEGE OF ENGINEERING KALAVAKKAM -603110**

**Department of Computer Science and Engineering**

**Mini Project**

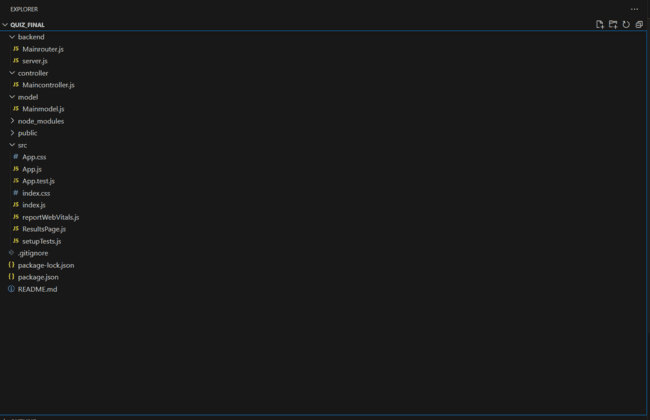
**UCS2611 – Internet Programming Lab**

**Full stack Quiz app using MERN**

**TEAM MEMBERS:**

1. **Ashwin R - 3122 21 5001 014**
2. **Jaanus Sri K.G:- 3122 21 5001 037**
3. **Karthik Vijayakumar:- 3122 21 5001 040**

**App directory:-**

****

**MongoDB database directory inside the database named “IpLab”**

****

**quiz:- The questions is added and stored by the admin for the users to play the game.**

**QuizAdmin:- Admin’s login credential.**

**quizUsers:- User’s login credentials(created by signing up in the app).**

**quizResults:- User’s results are stored for the admin to view the results of the users who all have played and their score.**

**In the React App,**

* MainRouter.js communicates with the database and server.js
* Server.js communicates between MainRouter and the Frontend(App.js)
* MainModel.js is to connect to the database and the collections inside it.
* MainController.js validates the login info for User.
* ResultsPage.js retrieves the results from the quizResults collection and displays it to the admin.
* App.js is the frontend responsible for showing the UI along with the CSS designs done.

**User can,**

* Signup for playing the game
* Login to play the game(if already signed up)
* View questions and answer the questions by selecting a choice
* Switch to the next question
* See the results of the test taken

**Admin can,**

* Login to the admin dashboard
* Add questions to the game by giving the question and the choices along with the right answer
* See user results with the username and score in a tabular form

MainRouter.js

const express = require('express');

const router = express.Router();

const { MongoClient } = require('mongodb');

const { generateMessage } = require('../controller/Maincontroller');

// MongoDB connection setup

const uri = "mongodb://localhost:27017";

const dbName = "ipLab";

const client = new MongoClient(uri);

// Connect to MongoDB

async function connectMongoDB() {

    try {

        await client.connect();

        console.log("Connected to MongoDB");

    } catch (error) {

        console.error("Error connecting to MongoDB:", error);

        throw error;

    }

}

// Connect to MongoDB when the server starts

connectMongoDB();

// Routes

const adminCollection = client.db(dbName).collection("quizAdmin");

const userCollection = client.db(dbName).collection("quizUsers");

const quizCollection = client.db(dbName).collection("quiz");

const resultsCollection = client.db(dbName).collection("quizResults");

// Admin login route

router.post("/admin/login", async (req, res) => {

    const { username, password } = req.body;

    try {

        const data = await generateMessage(username, password, 'quizAdmin');

        res.json({ message: data });

    } catch (error) {

        console.error("Error generating message:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// User login route

router.post("/login", async (req, res) => {

    const { username, password } = req.body;

    try {

        const data = await generateMessage(username, password, 'quizUsers');

        res.json({ message: "Login Successful" });

    } catch (error) {

        console.error("Error generating message:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// User signup route

router.post('/signup', async (req, res) => {

    const { username, password } = req.body;

    try {

        const existingUser = await userCollection.findOne({ username });

        if (existingUser) {

            return res.status(400).json({ message: 'Username already exists' });

        } else {

            const newUser = await userCollection.insertOne({ username, password });

            res.status(201).json({ message: 'User signup successful' });

        }

    } catch (error) {

        console.error("Error signing up:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// Route to add a question to the quiz

router.post('/admin/addQuestion', async (req, res) => {

    try {

        const { question, answer, choice1, choice2, choice3, choice4 } = req.body;

        const newQuestion = await quizCollection.insertOne({ question, answer, choice1, choice2, choice3, choice4 });

        res.status(201).json({ message: 'Question added to quiz' });

    } catch (error) {

        console.error("Error adding question:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// Route to fetch random quiz questions

router.get('/quiz/random', async (req, res) => {

    try {

        const randomQuestions = await quizCollection.aggregate([{ $sample: { size: 10 } }]).toArray();

        res.json(randomQuestions);

    } catch (error) {

        console.error("Error fetching random questions:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// Route to update user score

router.post('/user/updateScore', async (req, res) => {

    const { username, password, score } = req.body;

    try {

        // Update user's score

        const result = await resultsCollection.insertOne({ username, password , score  });

        res.status(201).json({ message: 'User score updated successfully', result });

    } catch (error) {

        console.error("Error updating user score:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

// Route to fetch quiz results

router.get('/quiz/results', async (req, res) => {

    try {

        const quizResults = await resultsCollection.find({}).toArray();

        res.json(quizResults);

    } catch (error) {

        console.error("Error fetching quiz results:", error);

        res.status(500).json({ message: 'Internal server error' });

    }

});

module.exports = router;

Server.js

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const mainRoute = require('./Mainrouter');

const app = express();

const PORT = process.env.PORT || 5000;

app.use(bodyParser.json());

app.use(cors());

app.use('/', mainRoute);

app.listen(PORT, () => {

    console.log("Server Listening on port", PORT);

});

MainController.js

const { connect } = require('../model/Mainmodel');

async function generateMessage(username, password, collection) {

    try {

        const db = await connect();

        const usersCollection = db.collection(collection);

        const user = await usersCollection.findOne({ username });

        if (!user) {

            return 'incorrect username or password'; // Return error message if user not found

        }

        if (user.password !== password) {

            return 'incorrect username or password'; // Return error message if password is incorrect

        }

        console.log("Success. ", user.username + " " + user.password);

        return user.username;

    } catch (error) {

        console.log("Error logging in:", error);

        throw error;

    }

}

module.exports = { generateMessage };

MainModel.js

const { MongoClient } = require('mongodb');

const uri = "mongodb://localhost:27017";

const dbName = "ipLab";

async function connect() {

    try {

        const client = new MongoClient(uri);

        await client.connect();

        const db = client.db(dbName);

        return db;

    } catch (error) {

        console.error("Error connecting to MongoDB:", error);

        throw error;

    }

}

module.exports = { connect };

App.js

// App.js

import React, { useState, useEffect } from 'react';

import './App.css'

import ResultsPage  from './ResultsPage';

export default function App() {

    const [currentPage, setCurrentPage] = useState("home");

    const [message, setMessage] = useState("");

    const [username, setUsername] = useState("");

    const [password, setPassword] = useState("");

    const [question, setQuestion] = useState("");

    const [answer, setAnswer] = useState("");

    const [choice1, setChoice1] = useState("");

    const [choice2, setChoice2] = useState("");

    const [choice3, setChoice3] = useState("");

    const [choice4, setChoice4] = useState("");

    const [isAdminLoggedIn, setIsAdminLoggedIn] = useState(false);

    const [isUserLoggedIn, setIsUserLoggedIn] = useState(false);

    const [questionData, setQuestionData] = useState([]);

    const [quizStarted, setQuizStarted] = useState(false);

    const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);

    const [score, setScore] = useState(0);

    const [timer, setTimer] = useState(10);

    const [selectedAnswerIndex, setSelectedAnswerIndex] = useState(null);

    // Timer logic

    useEffect(() => {

        let interval;

        if (quizStarted && timer > 0) {

            interval = setInterval(() => {

                setTimer(prevTime => prevTime - 1);

            }, 1000);

        } else if (quizStarted && timer === 0) {

            handleTimeout(); // Call function to handle timeout

        }

        return () => clearInterval(interval);

    }, [quizStarted, timer]);

    useEffect(() => {

        fetchRandomQuestions();

    }, []);

    const handleAdminClick = () => {

        setCurrentPage("adminLogin");

    };

    const handleUserClick = () => {

        setCurrentPage("userOptions");

    };

    // Handle timeout

    const handleTimeout = () => {

        handleAnswer(null, null); // Move to the next question with no selected answer

    };

    const handleLogin = () => {

        if (currentPage === "adminLogin") {

            adminLogin();

        } else if (currentPage === "userLogin") {

            userLogin();

        }

    };

    const handleSignup = () => {

        fetch('http://localhost:5000/signup', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ username, password }),

        })

            .then(res => res.json())

            .then(data => {

                setMessage(data.message);

                console.log(data.message);

                if (data.message === 'User signup successful') {

                    setIsUserLoggedIn(true);

                    setCurrentPage("userDashboard");

                }

            })

            .catch(error => console.error('Error:', error));

    };

    const handleAddQuestion = () => {

        fetch('http://localhost:5000/admin/addQuestion', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ question, answer, choice1, choice2, choice3, choice4 }),

        })

            .then(res => res.json())

            .then(data => { setMessage(data.message); console.log(data.message);alert("Quiz added to the database")})

            .catch(error => console.error('Error:', error));

    };

    const adminLogin = () => {

        fetch('http://localhost:5000/admin/login', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ username, password }),

        })

            .then(res => res.json())

            .then(data => {

                setMessage(data.message);

                console.log(data.message);

                if(data.message === 'Admin') {

                    setIsAdminLoggedIn(true);

                    setCurrentPage("renderAdminDashboard");

                }

            })

            .catch(error => console.error('Error:', error));

    };

    const userLogin = () => {

        fetch('http://localhost:5000/login', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ username, password }),

        })

            .then(res => res.json())

            .then(data => {

                setMessage(data.message);

                console.log(data.message);

                if (data.message !== 'incorrect username or password') {

                    setIsUserLoggedIn(true);

                    setCurrentPage("userDashboard");

                }

            })

            .catch(error => console.error('Error:', error));

    };

    const fetchRandomQuestions = () => {

        fetch('http://localhost:5000/quiz/random')

            .then(res => res.json())

            .then(data => {

                setQuestionData(data);

            })

            .catch(error => console.error('Error fetching questions:', error));

    };

    const handleStartQuiz = () => {

        setQuizStarted(true);

        setTimer(10);

        setScore(0);

        setCurrentQuestionIndex(0);

    };

    const renderHomePage = () => {

        return (

            <center>

                <div className="d1">

                    <h2>Welcome to Quiz</h2>

                    <button onClick={handleAdminClick}>Admin</button>

                    <button onClick={handleUserClick}>User</button>

                </div>

            </center>

        );

    };

    const renderUserOptions = () => {

        return (

            <center>

                <div className="d1">

                    <h2>User Options</h2>

                    <button onClick={() => setCurrentPage("userLogin")}>Login</button>

                    <button onClick={() => setCurrentPage("userSignup")}>Signup</button>

                    <button onClick={() => setCurrentPage("home")}>Back</button>

                </div>

            </center>

        );

    };

    const renderAdminLoginPage = () => {

        return (

            <center>

                <div className="d1">

                    <h2>Login</h2>

                    <label>Username: <input type="text" value={username} onChange={(e) => setUsername(e.target.value)} /></label><br /><br />

                    <label>Password: <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} /></label><br /><br />

                    <button onClick={() => setCurrentPage("home")}>Back</button>

                    <button onClick={handleLogin}>Login</button>

                </div>

            </center>

        );

    };

    const renderUserLoginPage = () => {

        return (

            <center>

                <div className="d1">

                    <h2>Login</h2>

                    <label>Username: <input type="text" value={username} onChange={(e) => setUsername(e.target.value)} /></label><br /><br />

                    <label>Password: <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} /></label><br /><br />

                    <button onClick={() => setCurrentPage("userOptions")}>Back</button>

                    <button onClick={handleLogin}>Login</button>

                </div>

            </center>

        );

    };

    const renderSignupPage = () => {

        return (

            <center>

                <div className="d1">

                    <h2>Signup</h2>

                    <label>Username: <input type="text" value={username} onChange={(e) => setUsername(e.target.value)} /></label><br /><br />

                    <label>Password: <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} /></label><br /><br />

                    <button onClick={() => setCurrentPage("userOptions")}>Back</button>

                    <button onClick={handleSignup}>Signup</button>

                </div>

            </center>

        );

    };

    const renderAddQuestionPage = () => {

        return (

            <center>

                <div className="addQuiz">

                    <h2>Add Quiz</h2>

                    <label>Question: <input type="text" value={question} onChange={(e) => setQuestion(e.target.value)} /></label><br /><br />

                    <label>Answer: <input type="text" value={answer} onChange={(e) => setAnswer(e.target.value)} /></label><br /><br />

                    <label>Choice 1: <input type="text" value={choice1} onChange={(e) => setChoice1(e.target.value)} /></label><br /><br />

                    <label>Choice 2: <input type="text" value={choice2} onChange={(e) => setChoice2(e.target.value)} /></label><br /><br />

                    <label>Choice 3: <input type="text" value={choice3} onChange={(e) => setChoice3(e.target.value)} /></label><br /><br />

                    <label>Choice 4: <input type="text" value={choice4} onChange={(e) => setChoice4(e.target.value)} /></label><br /><br />

                    <button onClick={() => setCurrentPage("adminLogin")}>Back</button>

                    <button onClick={handleAddQuestion}>Add Question</button>

                </div>

            </center>

        );

    };

    // const renderQuizQuestion = () => {

    //     const currentQuestion = questionData[currentQuestionIndex];

    //     if (!currentQuestion) {

    //         return null;

    //     }

    //     const choices = [currentQuestion.choice1, currentQuestion.choice2, currentQuestion.choice3, currentQuestion.choice4];

    //     const correctAnswer = currentQuestion.answer;

    //     return (

    //         <div>

    //             <h3 id ="qno">Question No: {currentQuestionIndex + 1}/10</h3>

    //             <h3>{currentQuestion.question}</h3>

    //             {choices.map((choice, index) => {

    //                 const isCorrect = choice === correctAnswer;

    //                 const buttonStyle = selectedAnswerIndex === index ? (isCorrect ? 'correct' : 'incorrect') : 'default';

    //                 return (

    //                     <button key={index} className={buttonStyle} onClick={() => handleAnswer(choice, index)}>{choice}</button>

    //                 );

    //             })}

    //         </div>

    //     );

    // };

    const renderQuizQuestion = () => {

        const currentQuestion = questionData[currentQuestionIndex];

        if (!currentQuestion) {

            return null;

        }

        const choices = [currentQuestion.choice1, currentQuestion.choice2, currentQuestion.choice3, currentQuestion.choice4];

        const correctAnswer = currentQuestion.answer;

        return (

            <div>

                <h3 id="qno">Question No: {currentQuestionIndex + 1}/10</h3>

                <h3>{currentQuestion.question}</h3>

                {choices.map((choice, index) => {

                    const isCorrect = choice === correctAnswer;

                    const buttonStyle = selectedAnswerIndex === index ? (isCorrect ? 'correct' : 'incorrect') : 'default';

                    return (

                        <button key={index} className={buttonStyle} onClick={() => handleAnswer(choice, index)}>{choice}</button>

                    );

                })}

            </div>

        );

    };

    // const handleAnswer = (selectedAnswer, selectedIndex) => {

    //     const currentQuestion = questionData[currentQuestionIndex];

    //     let isCorrect = false;

    //     if (selectedAnswer === currentQuestion.answer) {

    //         // Increment score if answer is correct

    //         setScore(prevScore => prevScore + 1);

    //         isCorrect = true;

    //     }

    //     // Update the UI to show the correct and incorrect answers

    //     setQuestionData(prevQuestionData => {

    //         const updatedQuestionData = [...prevQuestionData];

    //         updatedQuestionData[currentQuestionIndex] = { ...currentQuestion, selectedAnswer, isCorrect };

    //         return updatedQuestionData;

    //     });

    //     // Set the selected answer index

    //     setSelectedAnswerIndex(selectedIndex);

    //     // Delay the transition to the next question

    //     setTimeout(() => {

    //         if (currentQuestionIndex === 9) {

    //             // Stop the quiz

    //             setQuizStarted(false);

    //             // Display the score

    //             setCurrentPage("quizResult");

    //         } else {

    //             // Move to the next question

    //             setCurrentQuestionIndex(prevIndex => prevIndex + 1);

    //             // Reset selected answer index

    //             setSelectedAnswerIndex(null);

    //         }

    //     }, 1000);

    // };

    const handleAnswer = (selectedAnswer, selectedIndex) => {

        const currentQuestion = questionData[currentQuestionIndex];

        let isCorrect = false;

        if (selectedAnswer === currentQuestion.answer) {

            // Increment score if answer is correct

            setScore(prevScore => prevScore + 1);

            isCorrect = true;

        }

        // Update the UI to show the correct and incorrect answers

        setQuestionData(prevQuestionData => {

            const updatedQuestionData = [...prevQuestionData];

            updatedQuestionData[currentQuestionIndex] = { ...currentQuestion, selectedAnswer, isCorrect };

            return updatedQuestionData;

        });

        // Set the selected answer index

        setSelectedAnswerIndex(selectedIndex);

        // Delay the transition to the next question

        setTimeout(() => {

            if (currentQuestionIndex === 9) {

                // Stop the quiz

                setQuizStarted(false);

                // Display the score

                setCurrentPage("quizResult");

                // Update user data with score

                updateScore();

            } else {

                // Move to the next question

                setCurrentQuestionIndex(prevIndex => prevIndex + 1);

                // Reset selected answer index

                setSelectedAnswerIndex(null);

                // Reset timer for the next question

                setTimer(10);

            }

        }, 1000);

    };

    // Function to update user score

    const updateScore = () => {

        // Fetch endpoint to update user score

        fetch('http://localhost:5000/user/updateScore', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({ username, password, score }), // Pass username, password, and score

        })

            .then(res => res.json())

            .then(data => {

                console.log(data.message);

            })

            .catch(error => console.error('Error updating user score:', error));

    };

    const renderAdminDashboard = () => {

        return(

            <center>

                <h3>Admin Dashboard</h3>

                {isAdminLoggedIn && ( // Conditionally render the "Add Question" button

                    <div>

                        <button onClick={() => setCurrentPage("addQuestion")}>Add Question</button>

                        <button onClick={() => setCurrentPage("viewResults")}>View Results</button>

                    </div>

                )}

                <button className="adminBack" onClick={() => setCurrentPage("adminLogin")} style={{ position: 'absolute', bottom: 10, left: 10 }}>Back</button> {/\* Back button \*/}

            </center>

        )

    }

    const handleGoBack = () => {

        setCurrentPage("renderAdminDashboard");

    };

    const renderUserDashboard = () => {

        return (

            <center>

                <div className='quizPage'>

                    <h2>QUIZ TIME</h2>

                    <button className="quizBack" onClick={() => setCurrentPage("home")}>Back</button>

                    {!quizStarted && (

                        <button className="quizStart" onClick={handleStartQuiz}>Start Quiz</button>

                    )}

                    {quizStarted && <h3 id="timer">Timer : {timer}</h3>}

                    {quizStarted && renderQuizQuestion()}

                </div>

            </center>

        );

    };

    const QuizResultPage = () => {

        return (

            <center>

                <div>

                    <h2>You Scored {score} out of 10</h2>

                    <button onClick={() => setCurrentPage("home")}>Back to Home</button>

                </div>

            </center>

        );

    };

    return (

        <>

            {currentPage === "home" && renderHomePage()}

            {currentPage === "adminLogin" && renderAdminLoginPage()}

            {currentPage === "userOptions" && renderUserOptions()}

            {currentPage === "userLogin" && renderUserLoginPage()}

            {currentPage === "userSignup" && renderSignupPage()}

            {isAdminLoggedIn && currentPage === "renderAdminDashboard" && renderAdminDashboard()} {/\* Updated this line \*/}

            {isAdminLoggedIn && currentPage === "addQuestion" && renderAddQuestionPage()}

            {isUserLoggedIn && currentPage === "userDashboard" && renderUserDashboard()}

            {currentPage === "quizResult" && <QuizResultPage/>}

            {isAdminLoggedIn && currentPage === "viewResults" && <ResultsPage goBack={handleGoBack}/>}

        </>

    );

}

**App.css**

.d1{

  margin-top: 15%;

  background-color: rgb(50, 50, 255,0.3);

  border: solid 2px rgb(12, 1, 10);

  padding: 80px;

  width: 450px;

}

button{

  margin: 30px;

  padding: 10px;

  border: solid 1px ;

  background: rgba(142, 26, 138, 0.396);

  border-radius: 10%;

  color: aliceblue;

  font-size: large;

}

.addQuiz{

  background-color: rgb(50, 50, 255,0.3);

  border: solid 2px rgb(12, 1, 10);

  padding: 30px;

  background-size: 100vh 100vm;

  height: 493px;

}

#timer{

  display: flex;

  margin-left: 2%;

  color: red;

  top:2px;

}

#qno{

  display: flex;

  position: absolute;

  right:2px;

  top:50px;

  margin-right: 2%;

  color: rgb(90, 205, 247);

}

.quizBack{

  display: flex;

  bottom: 0%;

  position: absolute;

  background: rgba(92, 87, 92, 0.396);

  color: black;

}

.quizStart{

  margin: 50px;

  padding: 10px;

  border: solid 2px;

  color: aliceblue;

  background: rgba(142, 26, 138, 0.396);

  border-radius: 10px;

  font-size: large;

  size: 50px;

  display: flex;

  margin-top: 20%;

}

.correct {

  background-color: green;

}

.incorrect {

  background-color: red;

}

#scoretable{

  font: 25px;

  color: white;

}

**index.css**

body {

  background-image: url('https://miro.medium.com/v2/resize:fit:1400/format:webp/1\*xMuIOwjliGUPjkzukeWKfw.jpeg');

  background-repeat: no-repeat;

  background-size: cover;

  margin: 0;

  padding: 0;

  font-family: Arial, sans-serif; /\* Example font family \*/

  color: white;

  font-size: 30px;

}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

**ResultsPage.js**

import React, { useState, useEffect } from 'react';

const ResultsPage = ({ goBack }) => {

    const [quizResults, setQuizResults] = useState([]);

    const [error, setError] = useState(null);

    useEffect(() => {

        fetchQuizResults();

    }, []);

    const fetchQuizResults = () => {

        fetch('http://localhost:5000/quiz/results')

            .then(res => res.json())

            .then(data => {

                setQuizResults(data);

            })

            .catch(error => {

                console.error('Error fetching quiz results:', error);

                setError('Failed to fetch quiz results. Please try again later.');

            });

    };

    return (

        <div id ="scoretable">

            {error && <p>Error: {error}</p>}

            <h2>Quiz Results</h2>

            <table border="1">

                <thead>

                    <tr>

                        <th>Username</th>

                        <th>Password</th>

                        <th>Score</th>

                    </tr>

                </thead>

                <tbody>

                    {quizResults.map(result => (

                        <tr key={result.\_id}>

                            <td>{result.username}</td>

                            <td>{result.password}</td>

                            <td>{result.score}</td>

                        </tr>

                    ))}

                </tbody>

            </table>

            <button onClick={goBack} style={{ position: 'absolute', bottom: 10, left: 10 }}>Back</button>

        </div>

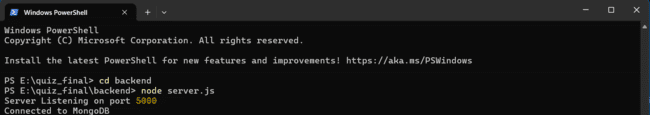
    );

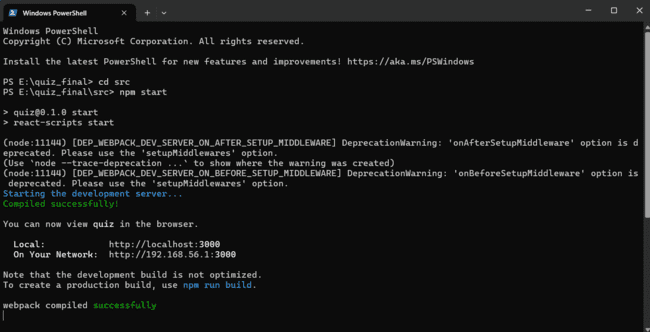
};

export default ResultsPage;

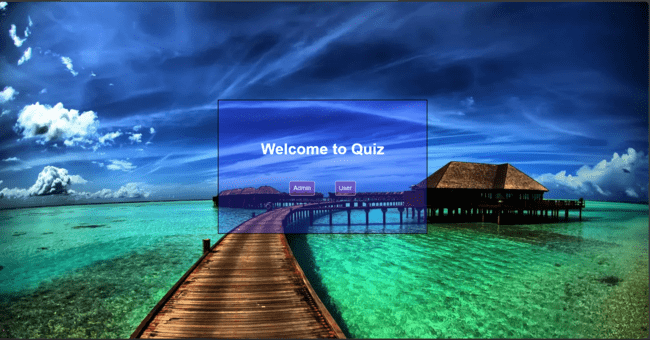
Run the server.js using “node server.js “ in it’s directory.

Run the App.js using “npm start” in it’s directory.

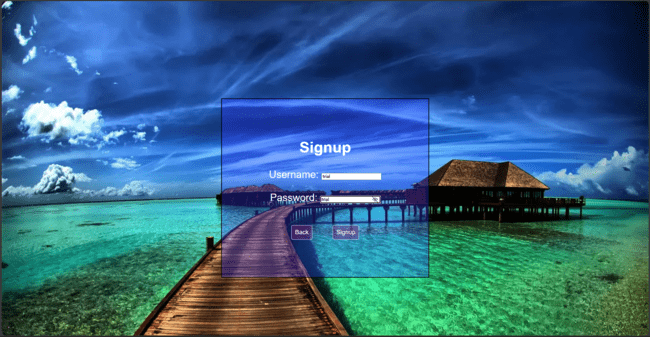


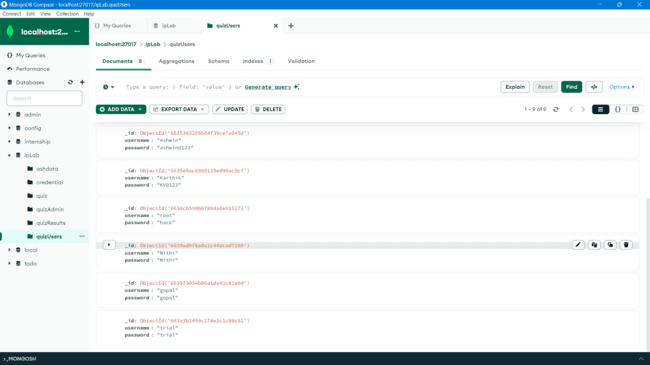


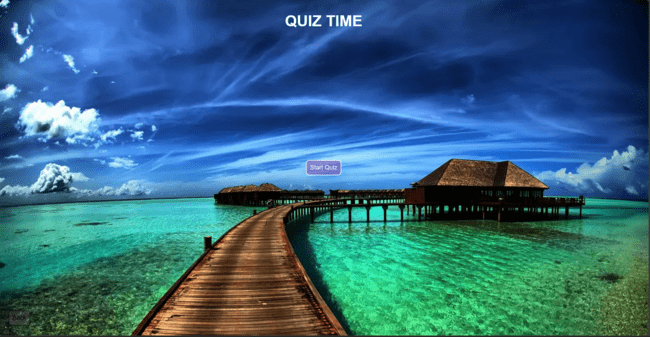
Now, the app gets deployed in the browser using the URL “http://localhost:3000”

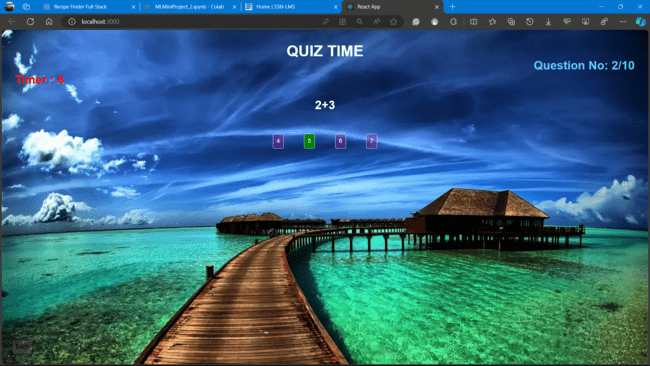


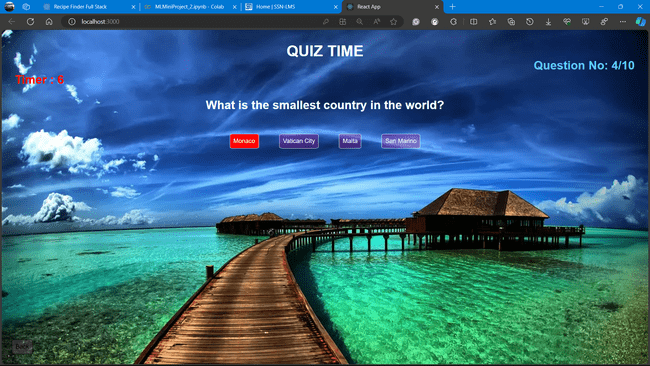
**1. For user:-**



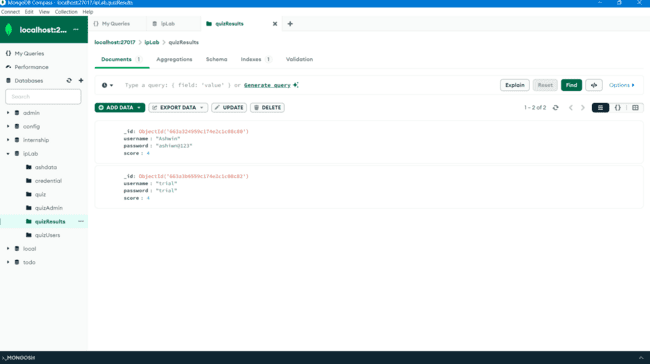












User after signing up, can start playing the game. If he/she has already played the game earlier, they can directly go for log in.

After signup/login, the user can start the test.

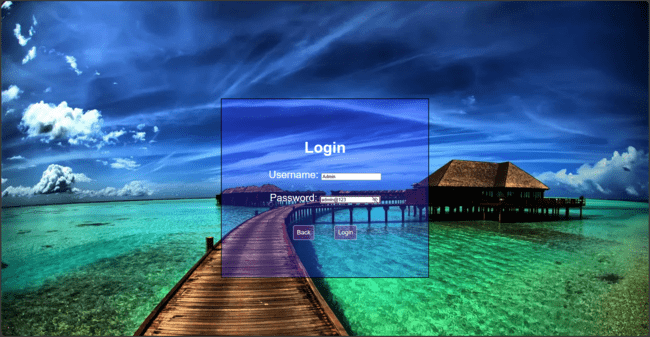
The timer will keep running and the page will automatically move to the next question after 10 seconds and if no option is selected.

If selected, the user will get a red/green box surrounding the option. If it is green the selected choice is correct and goes to the next question immediately. Else if it is red, the selected choice is wrong and goes to the next question immediately.

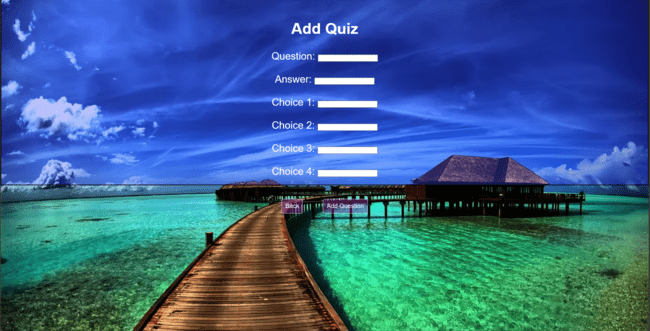
At the end of the test, the user can see the test score. The score is added to the quizResults collection of the ipLab database with the “username” , “password” and “score” attained. This information is visible to admin only and we will see below how the admin can retrieve and see the details.

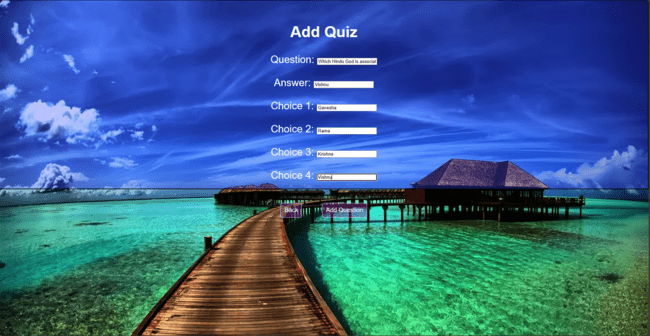
2. **For Admin:-**

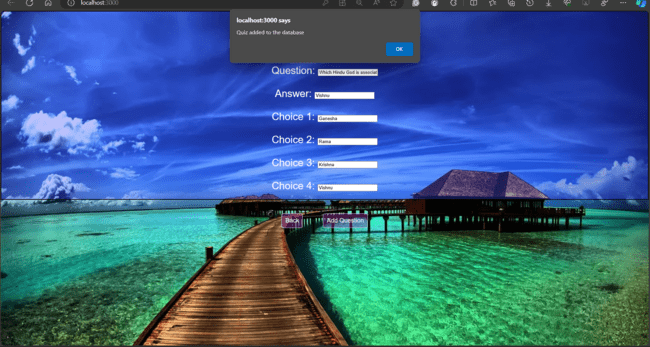
****

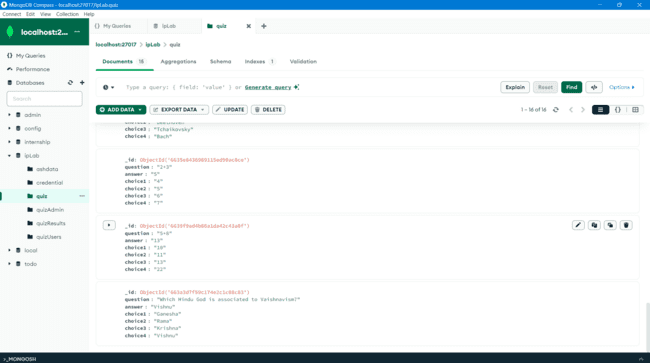
****

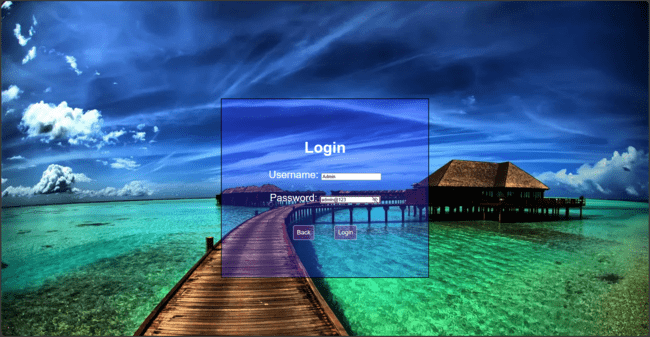


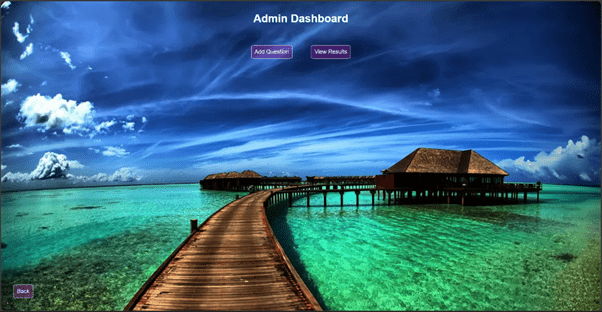








****



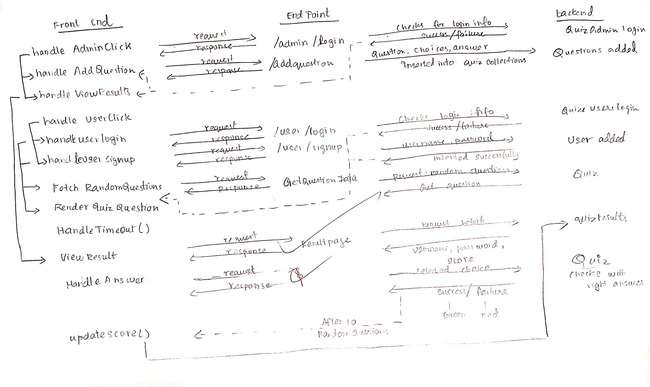


The admin logs in using the credential allotted. The there are two tabs in the Admin Dashboard. One is “Add Question” and other is “View Results”.

If the Admin selects Add Question, admin gives the question, answer and the choices. After adding, there is a window.alert() prompt that the question is added to the database. This is verified by checking the “quiz” collection of ipLab database and we see the question is added.

If the Admin selects View Results, the admin can see the Quiz results of the user in a tabular column. Each row consists of the username, password and the score attained by the user(s).

**Flow Architecure diagram**



**GitHub Link:-**

<https://github.com/SolitudeAsh/Quiz-app-using-MERN>