

MINI PROJECT

(2020-21)

QUIZ MANIA

Project Report

Department of Computer Engineering & Applications

Institute of Engineering & Technology



GLA University

Mathura- 281406, INDIA

Supervised By :-

Mrs. Ruchi Gupta

Mr. Anand Gupta

Submitted by –

Vijay Kumar Tyagi (181500787)

Saurabh Saraswat (181500640)

Bhanu Pratap Singh(181500495)

Tanesh Gupta(181500749)

Prateek Sharma (181500495)

Declaration

We hereby declare that the work which is being presented in the Mini Project **“Quiz Mania”**, in partial fulfillment of the requirements for Mini project is an authentic record of our own work carried under the supervision of **Mr. Anand Gupta, Technical Trainer.**

Saurabh Saraswat

Vijay Kumar Tyagi

Tanesh Gupta

Bhanu Pratap Singh

Prateek Sharma

Table of Contents

ACKNOWLEDGEMENT	5
ABSTRACT	6
INTRODUCTION	7
Motivation	7
Problem Statement	8
Objective	8
TECHNOLOGIES USED	9-19
HARDWARE REQUIREMENTS AND SOFTWARES USED	20
DATABASES	20-25
SOFTWARE DESIGN	26-27
TESTING	28-36
CODE SNIPPETS	37-62
REFERENCES	63

ACKNOWLEDGEMENT

We take this opportunity to acknowledge all the people who have helped us wholeheartedly in every stage of this project.

I also extend our sincere thanks to all other faculty members of the Computer Science & Engineering Department.

Last but not the least, we acknowledge our friends for their contribution in the completion of the project

.

ABSTRACT

In this project we are going to build a real-time quiz website which can create quizzes competitively and collaboratively by students as well as teachers for the purpose of reducing the load required for a teacher and promoting interactions among students and between the teacher and students.

INTRODUCTION

Motivation

Nowadays, the demand of taking everything online has become a need of modern times as there is no certainty of being physically available every time because of hectic schedules and busy lifestyles. Therefore, Web Development has become the most basic need, because nowadays during the time of pandemic no School and colleges are opened. All the assessment of students has to be done by some online or without any physical presence of students at the institution . So the increasing demand of web development as well as the need of an Online assessment portal motivated

Problem Statement

Many Colleges, Institutes and academic centers work in manual mode for taking examinations. This is time consuming as well as resource consuming. All such operations are handled on test copies, files or registers manually. This project intends to use the latest advaus to develop the Project “QUIZMANIA”
ncements in information technology and provide a central web solution for automating some basic examination. This will help many stakeholders of a college, institute or school to quickly do examinations online.

It is a good source of interactivities among students and between the teacher and students. It is done in order to improve student’s comprehension levels and learning motivation. As one of their tools, online test tools are quite effective. However, in order to use the online test tool, a teacher is generally required a great deal of labor.

Objective

In Order to solve these problems we aimed to develop an online Web Application that can create quizzes online on the go and host them instantly and giving the full control to the host. Also Declaring the results instantly

TECHNOLOGIES USED

MERN Stack

What is Mern Stack ?

MERN Stack is a Javascript Stack

that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises

4 technologies namely: [MongoDB](#), [Express](#), [React](#) and [Node.js](#). It is designed to make the development process smoother and easier. Each of these 4 powerful technologies provides an end-to-end framework for

The developers to work in and each of these technologies play a big part in the development of web applications



MongoDB

MongoDB is a NoSQL database where each record is a document consisting of key-value pairs that are similar to JSON (JavaScript Object Notation) objects. MongoDB is flexible and allows its users to create schema, databases, tables, etc. Documents that are identifiable by a primary key make up the basic unit of MongoDB. Once MongoDB is installed, users can make use of Mongo Shell as well. Mongo shell provides a JavaScript interface through which the users can interact and carry out operations (eg: querying, updating records, deleting records).

Advantages of MongoDB

- Fast – Being a document-oriented database, easy to index documents. Therefore a faster response.
- Scalability – Large data can be handled by dividing it into several machines.
- Use of JavaScript – MongoDB uses JavaScript which is the biggest advantage.
- Schema Less – Any type of data in a separate document.
- Data stored in the form of JSON –
 1. Objects, Object Members, Arrays, Values and Strings
 2. JSON syntax is very easy to use.
 3. JSON has a wide range of browser compatibility.
 4. Sharing Data: Data of any size and type(video, audio) can be shared easily.
- Simple Environment Setup – It's really simple to set up MongoDB.
- Flexible Document Model – MongoDB supports document-model(tables, schemas, columns & SQL) which is faster and easier.
- Creating a database: Simply done using a “**use**” command:

Express JS

Express is a Node.js framework. Rather than writing the code using Node.js and creating loads of Node modules, Express makes it simpler and easier to write the back-end code. Express helps in designing great web applications and APIs. Express supports many middlewares which makes the code shorter and easier to write

Advantages Of Express JS

- Asynchronous and Single-threaded.
- Efficient, fast & scalable
- Has the biggest community for Node.js
- Express promotes code reusability with its built-in router.
- Robust API
- Create a new folder to start your express project and type below command in the command prompt to initialize a package.json file. Accept the default settings and continue.

React JS

React is a JavaScript library that is used for building user interfaces. React is used for the development of single-page applications and mobile applications because of its ability to handle rapidly changing data. React allows users to code in JavaScript and create UI components

Advantages of React

- Virtual DOM – A virtual DOM object is a representation of a DOM object. Virtual DOM is actually a copy of the original DOM. Any modification in the web application causes the entire UI to re-render the virtual DOM. Then the difference between the original DOM and this virtual DOM is compared and the changes are made accordingly to the original DOM.
- JSX – Stands for JavaScript XML. It is an HTML/XML JavaScript Extension which is used in React. Makes it easier and simpler to write React components.
- Components – ReactJS supports Components. Components are the building blocks of UI wherein each component has a logic and contributes to the overall UI. These components also promote code reusability and make the overall web application easier to understand.
- High Performance – Features like Virtual DOM, JSX and Components makes it much faster than the rest of the frameworks out there.
- Developing Android/Ios Apps – With React Native you can easily code Android-based or IOS-Based apps with just the knowledge of JavaScript and ReactJS.
- You can start your react application by first installing “create-react-app” using npm or yarn.

Node.js

Node.js provides a JavaScript Environment which allows the user to run their code on the server (outside the browser). Node pack manager i.e. npm allows the user to choose from thousands of free packages (node modules) to download.

Advantages Of Node.js

- Open source JavaScript Runtime Environment
- Single threading – Follows a single threaded model.
- Data Streaming
- Fast – Built on Google Chrome's JavaScript Engine, Node.js has a fast code execution.
- Highly Scalable
- Initialize a Node.js application by typing running the below command in the command window. Accept the standard settings.

"npm init"

Why MERN Stack ?

- The speed of design and development of websites and web applications,
- Reducing server costs,
- The performance of greatly optimized web applications and software,
- The ease of transposing a web application to a mobile application or software, thanks in particular to React Native,
- The luxury of designing a website using a single HTML document,

- The development of a computer application using a single language, JavaScript.

SOME OTHER TECHNOLOGIES USED

HTML

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as `` and `<input />` directly introduce content into the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997

The JSX components in our Application also returns HTML as the final output

CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

JavaScript

JavaScript is the **base technology** of our whole project. Whole Backend as well as some part frontend is developed using Javascript

JavaScript often abbreviated as JS, is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is a programming language that is characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web.[9] JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it,[10] and major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has APIs for working with text, arrays, dates, regular expressions, and the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities. It relies upon the host environment in which it is embedded to provide these features.

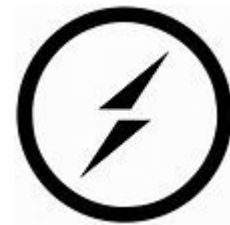
Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

The terms Vanilla JavaScript and Vanilla JS refer to JavaScript not extended by any frameworks or additional libraries. Scripts written in Vanilla JS are plain JavaScript code.

Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design. JavaScript was influenced by programming languages such as Self and Scheme.

SOCKET.IO

Socket.io is a Javascript networking library that runs server-side on Node.js and in the browser. It abstracts away websockets and other communication schemes, depending upon browser capabilities. It also includes convenient features such as broadcasts and multicasts, which are beyond the features of plain websockets.



Some Examples of Real-Time Applications

A real-time application (RTA) is an application that functions within a period that the user senses as immediate or current.

Some examples of real-time applications are –

- Instant messengers – Chat apps like Whatsapp, Facebook Messenger, etc. You need not refresh your app/website to receive new messages.
- Push Notifications – When someone tags you in a picture on Facebook, you receive a notification instantly.
- Collaboration Applications – Apps like google docs, which allow multiple people to update same documents simultaneously and apply changes to all people's instances.
- Online Gaming – Games like Counter Strike, Call of Duty, etc., are also some examples of real-time applications.

Why Socket.io

- Fast Communication

Writing a real-time application with popular web applications stacks like LAMP (PHP) has traditionally been very hard. It involves polling the server for changes, keeping track of timestamps, and it is a lot slower than it should be.

Sockets have traditionally been the solution around which most real-time systems are architected, providing a bi-directional communication channel between a client and a server. This means that the server can push messages to clients. Whenever an event occurs, the idea is that the server will get it and push it to the concerned connected clients.

Socket.IO is quite popular, it is used by Microsoft Office, Yammer, Zendesk, Trello, and numerous other organizations to build robust real-time systems. It is one of the most powerful JavaScript frameworks on GitHub, and most dependent upon the NPM (Node Package Manager) module. Socket.IO also has a huge community, which means finding help is quite easy.

- **Express JS**

We will be using express to build the web server that Socket.IO will work with. Any other node-server-side framework or even node HTTP server can be used. However, ExpressJS makes it easy to define routes and other things.

- **Optimization with React**

Socket io is widely used with MERN stack for achieving Real-Time Goodness. Because of its Optimization and high Compatibility with it Which is why it makes it the best choice for implementing Real time Connections

REDUX

Redux is a predictable state container for JavaScript apps.

It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger.

You can use Redux together with React, or with any other view library. It is tiny (2kB, including dependencies), but has a large ecosystem of addons available.

Principles of Redux

Predictability of Redux is determined by three most important principles as given below –

Single Source of Truth

The state of your whole application is stored in an object tree within a single store. As whole application state is stored in a single tree, it makes debugging easy, and development faster.

State is Read-only

The only way to change the state is to emit an action, an object describing what happened. This means nobody can directly change the state of your application.

Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers. A reducer is a central place where state modification takes place. Reducer is a function which takes state and action as arguments, and returns a newly updated state.

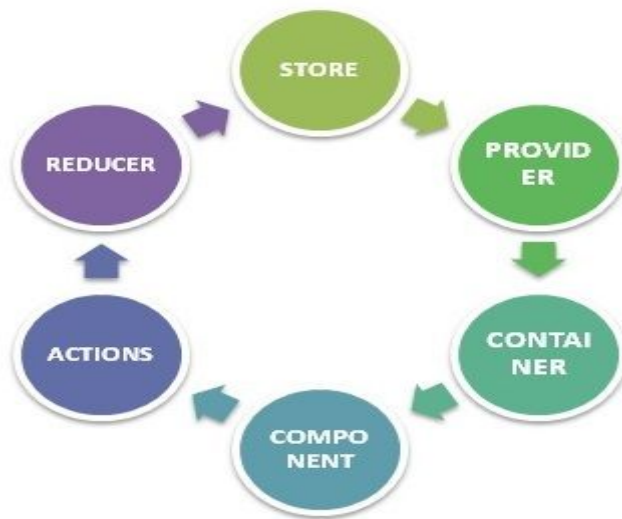
Redux Integration With Redux

Let us say if various react components need to display the same data in different ways without passing it as a prop to all the components from top-level component to the way down. It would be ideal to store it outside the react components. Because it helps in faster data retrieval as you need not pass data all the way down to different components.

When using Redux with React, **states will no longer need to be lifted up**; thus, it makes it easier for you to trace which action causes any change. As seen above, the component does not need to provide any state or method

for its children components to share data among themselves. Everything is handled by Redux.

Redux Workflow



STORE – Stores all your application state as a JavaScript object

PROVIDER – Makes stores available

CONTAINER – Get apps state & provide it as a prop to components

COMPONENT – User interacts through view component

ACTIONS – Causes a change in store, it may or may not change the state of your app

REDUCER – Only way to change app state, accept state and action, and return updated state.

However, Redux is an independent library and can be used with any UI layer. React-redux is the official Redux, UI binding with the react. Moreover, it encourages a good react Redux app structure. React-redux internally implements performance optimization, so that component re-render occurs only when it is needed.

To sum up, Redux is not designed to write the shortest and fastest code. It is intended to provide a predictable state management container. It helps us understand when a certain state changed, or where the data came from.

HARDWARE REQUIREMENTS AND SOFTWARES USED

Hardware Requirements:

- RAM:- 4.00GB ·
- Processor:- Intel(R)Core(TM) i3-4005U CPU @ 1.70GHz

Software Used :

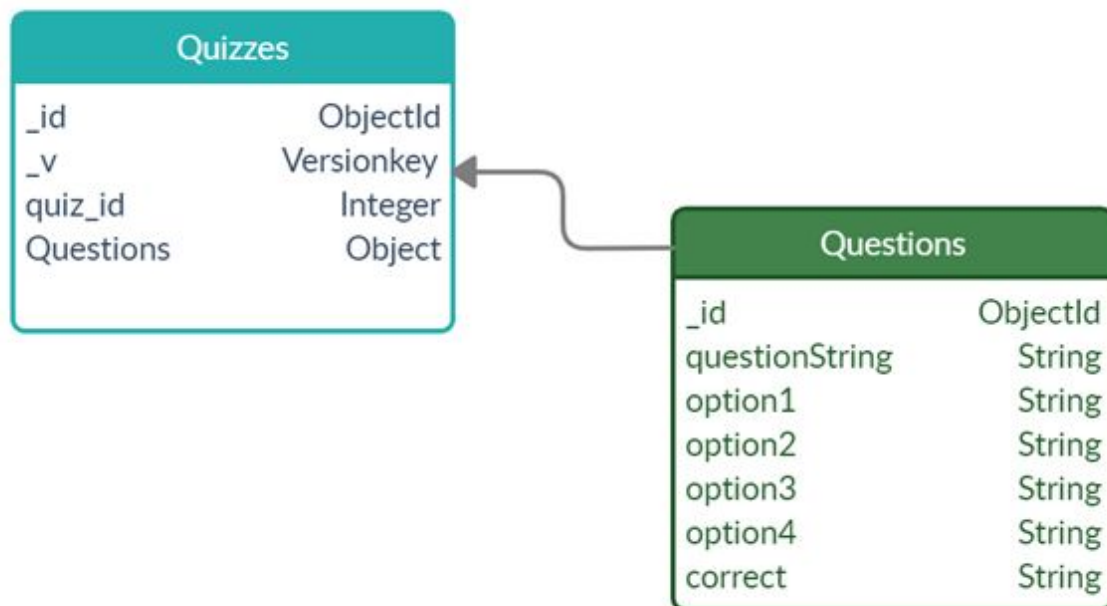
- VSCode text editor for creating API. ·
- Insomnia Designer for handling and testing http requests
- Chrome and Edge Browsers for testing
- React and Redux DevTools Extensions for Chrome and Edge
- Node js for hosting server on localhost .
- RoboMongo 3T for visual representation of Database

DATABASES

- There are 5 database models to be used in the project
 - ❖ Main Database (for Quizzes)
 - ❖ Backup Database (for Quizzes)
 - ❖ UserAuth (Authentication data of users)
 - ❖ Real Time users (without logged in)
 - ❖ Quiz History (stores every quiz Hosted)
- The user has a direct Interaction with the main database but No access to the backup database is granted to the user for keeping data safe
- The data in the backup database is saved automatically and instantly when the main database is updated
- The real Time users database does not stores any type of data of the user , However if he attempted the quiz under a host then the user_id , username and score is saved in the host quiz database
- The Auth info is stored in encrypted format in the database by using JWT(Json web query tokens)

Database Structures

1. Main Database(Quizzes)



SOME KEY POINTS

- The database document consists of a “Quiz” model and a “Question” model embedded within it.
- The Quiz model consists of 4 attributes out of which `_id` and `_v` are auto-generated by MongoDB.
- `_id` and `_v` are of datatype `objectId` and `versionKey` respectively.
- Another attribute is `quiz_id` that is provided to every individual quiz
- Question attribute is an Object that has an array consisting questions of a particular quiz in json format.
- Question model has 7 attributes out of which `_id` is autogenerated.
- Other six attributes are 1 Question string , 4 provided options, and 1 correct option.

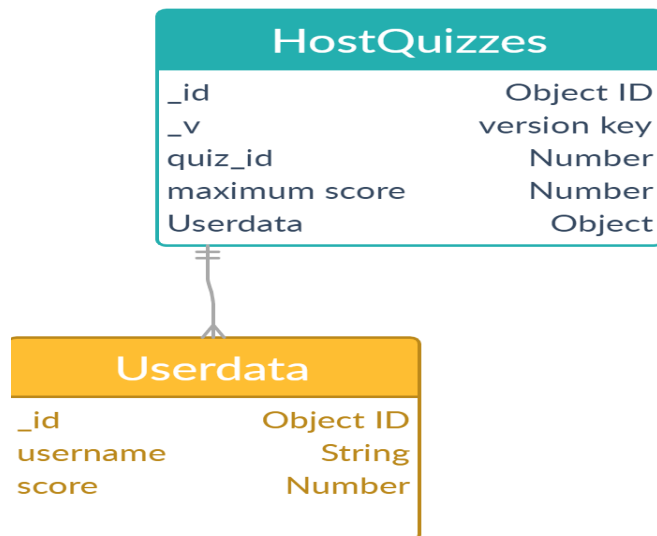
2. Backup Database schema(Questions)

Questions	
_id	ObjectId
quiz_id	Integer
questionString	String
option1	String
option2	String
option3	String
option4	String
correct	

SOME KEY POINTS

- The database consists of a single Object Question that contains every single question without any grouping.
- The database is simple but not highly indexed as it is not to be used by users or the application for accessing data frequently.
- The database is just going to be used for adding the data and because of which the schema is made as simple as possible.
- The model is almost similar with the embedded model of the main database, the only difference is just an extra quiz_id attribute.

3. Quiz History database(Hostquizzes)



Some Key Points

- The Database stores the history of quiz that was hosted by a Host
- Even if the Quiz Questions are changed i.e., added or deleted it still takes care as it stores the quiz with the Maximum score.
- Stores the username and the score of the user attempting the quiz

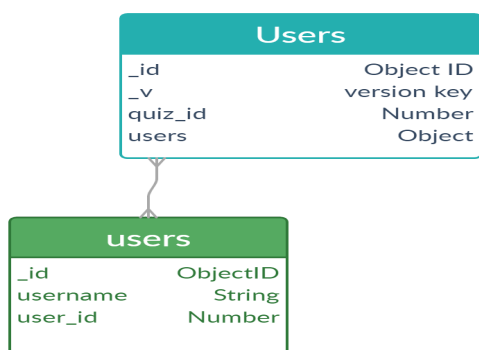
4. User Auth Database

UserAuth	
_id	Object ID
_v	version key
Email	String
First name	String
Last name	String
password	String

Some Key Points

- This Database stores the login information of the User or Say Host
- The passwords are stored in a encrypted format for security purpose
- The passwords stored are basically a key or a token that is matched at the time of the login and a session is created.

5. Real Time users database (Not logged in)



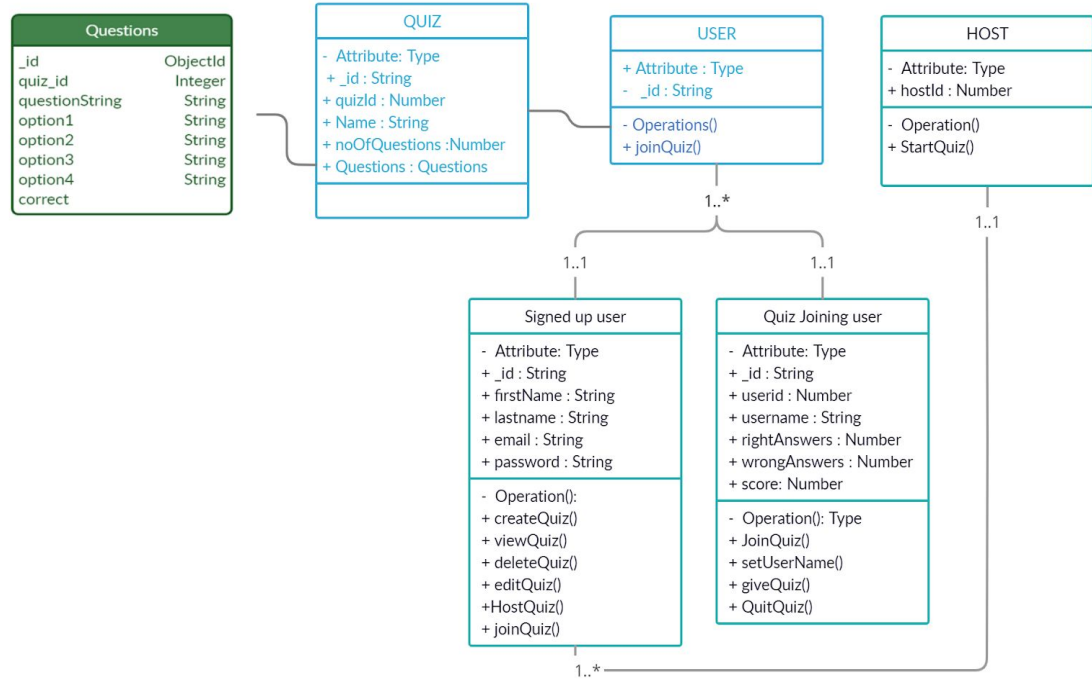
Some Key Points

- This Database is only a temporary database; no data will be saved permanently in the database .
- Data will be stored at the time when a quiz is being hosted to keep track of connected users
- will be deleted from this particular database, However the quiz stats are still saved in the host quiz database

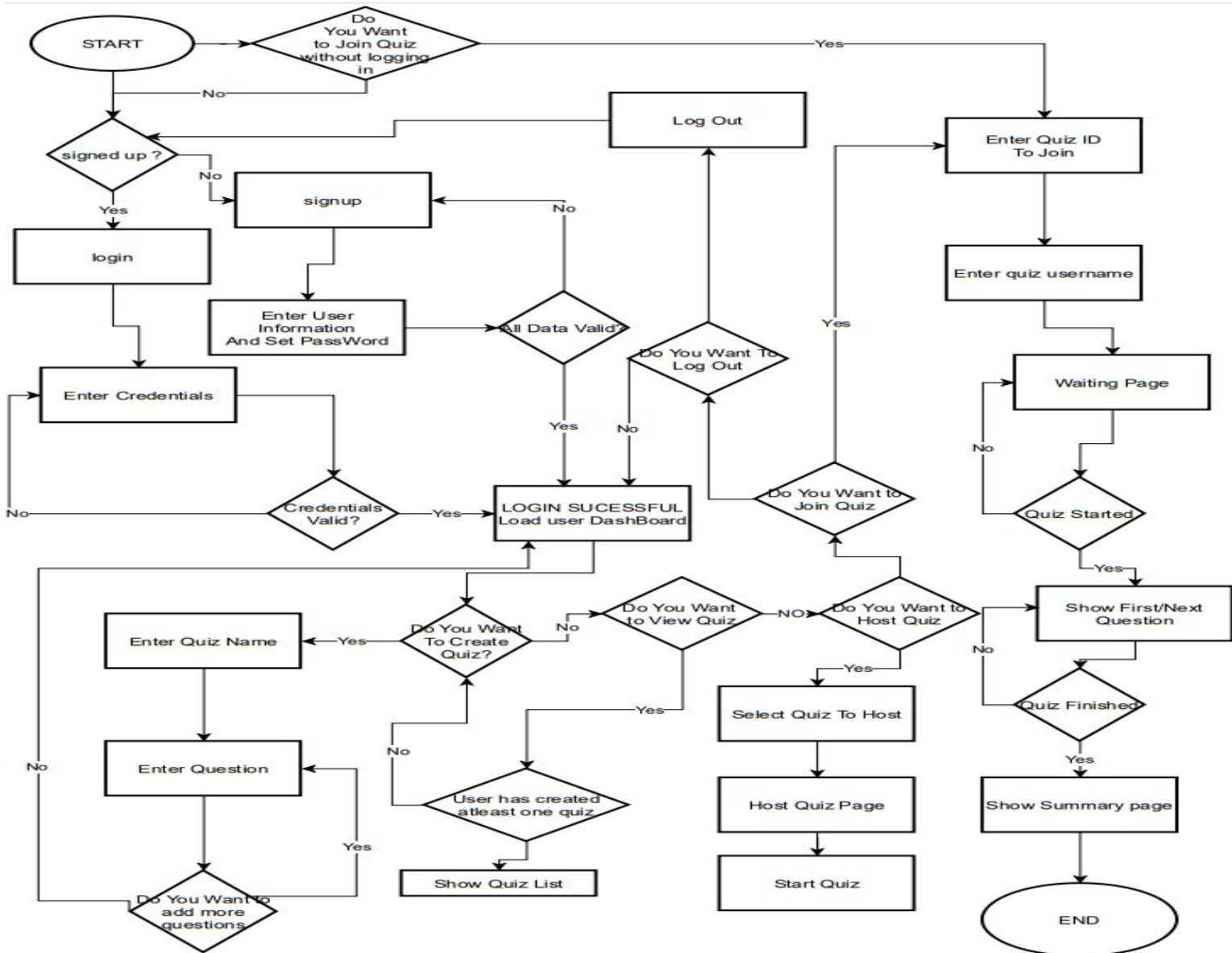
- This database gives the real time numbers and statistics of the users to the host as well as the other users
- As soon as the user leave or exit after finishing the quiz the details

SOFTWARE DESIGN

Class Diagram



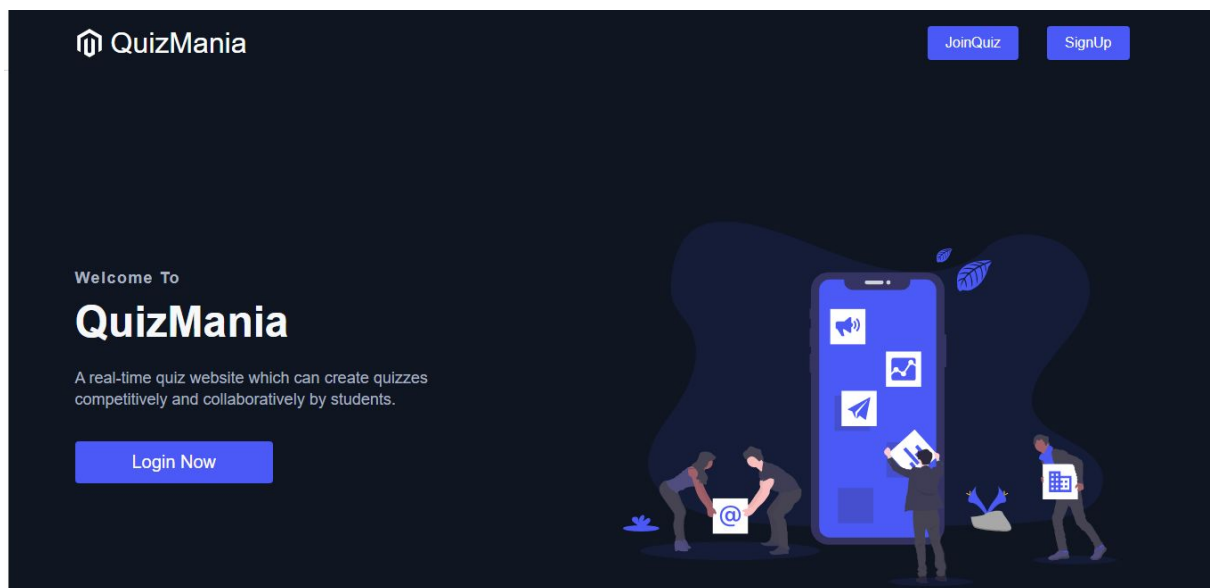
Flowchart



TESTING

1. HomePage:- This Page is the landing page or the homepage of the WebSite. Any new user or logged out user will be seeing this as Entry Point it will have

- Basic Introduction about the website
- Login or Signup buttons
- An instant join quiz button to join the quiz
- Footer information like Useful links etc

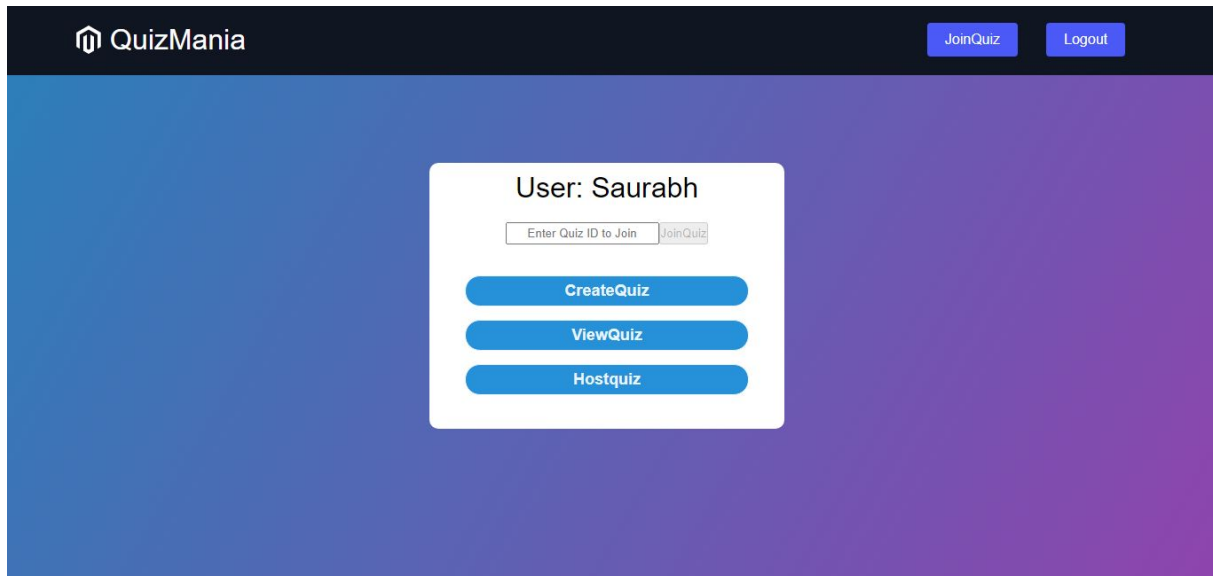


2. User Dashboard:-

This page is responsible for Main operations such as Creating new Quiz , joining Quiz , Viewing Quizzes ,Hosting Quizzes .This is the first page which a user will interact with when he visits **Quiz Mania**.

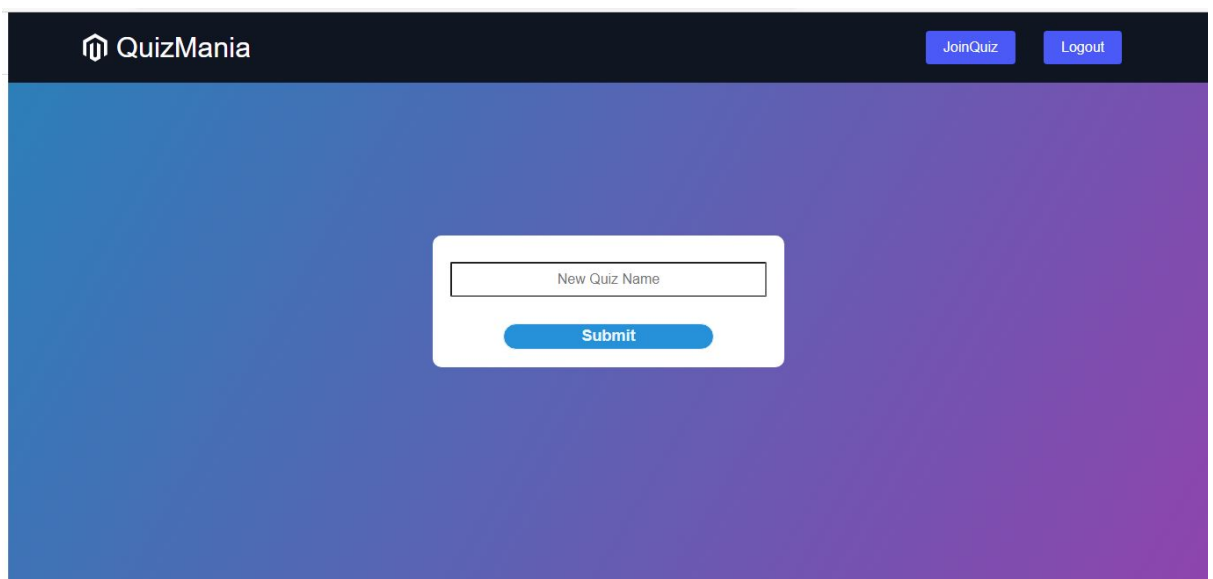
Currently the homepage consist mainly of two options

- I. To Create A Quiz : Here the user will be allowed add his/her questions and will be able to host a quiz event.
- II. To Join Quiz: Here the user will simply enter a code correspond to a quiz
- III . To Host a Quiz and let others Join the Quiz



3. Create Quiz Section:

This section is responsible for creating the Quiz with the help of a Quiz Name which the user has to specify. A **unique QUIZ_ID** will be generated that will be used for sharing and letting others join the Quiz. Then the user is required to input the questions in MCQ format along with one correct answer. After filling the required number of questions, the user can click on Finish to finally submit all the questions which will be saved in our database.



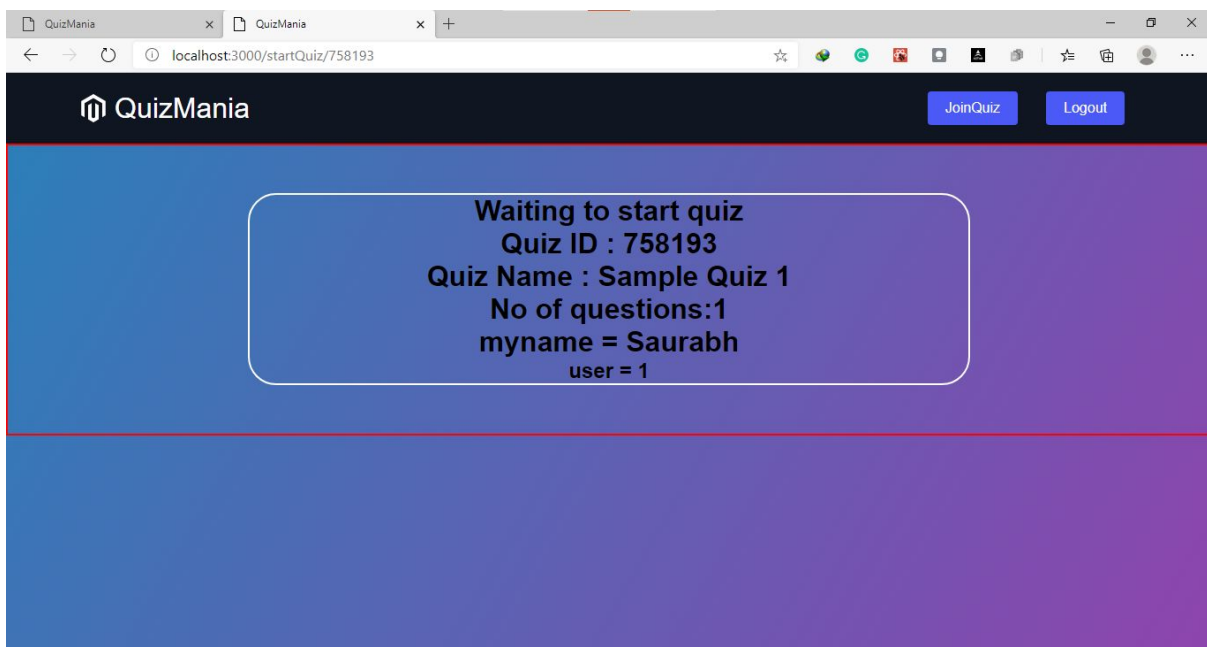
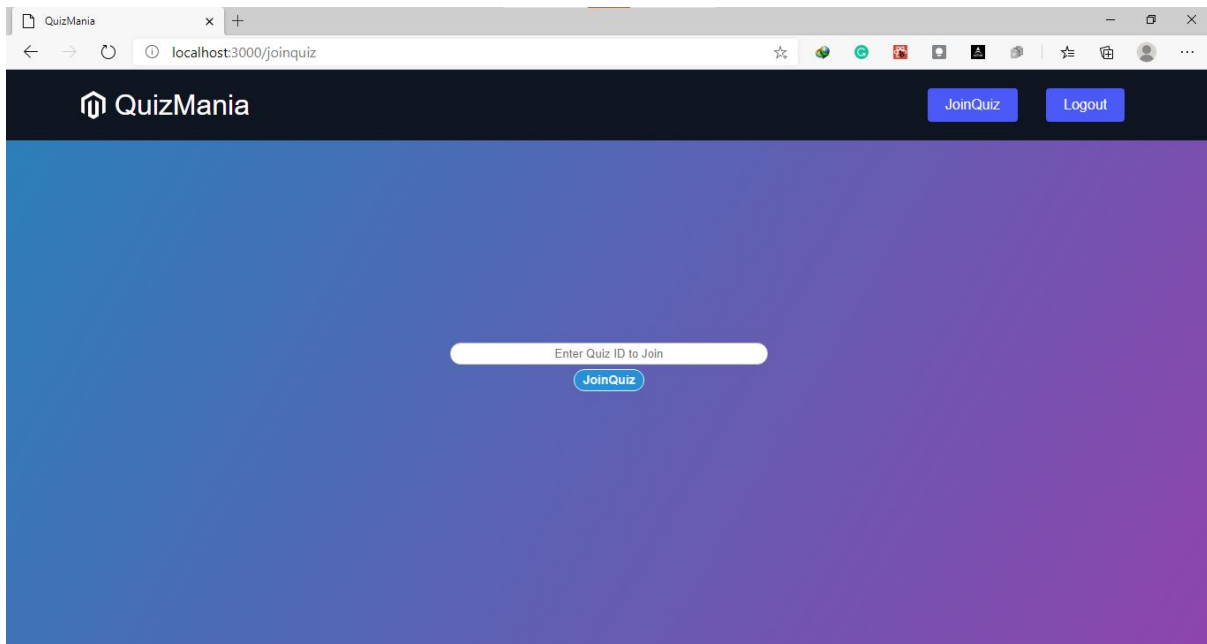
The screenshot shows the 'Enter Questions' form on the QuizMania website. The form is centered on a light gray background with a blue gradient on the left and a purple gradient on the right. At the top, the QuizMania logo is on the left, and 'JoinQuiz' and 'Logout' buttons are on the right. The form itself has a title 'Enter Questions' and contains several input fields: 'Sample Quiz 1', 'question', 'option1', 'option2', 'option3', 'option4', and a dropdown menu. At the bottom of the form are two blue buttons labeled 'Submit' and 'Finish'.

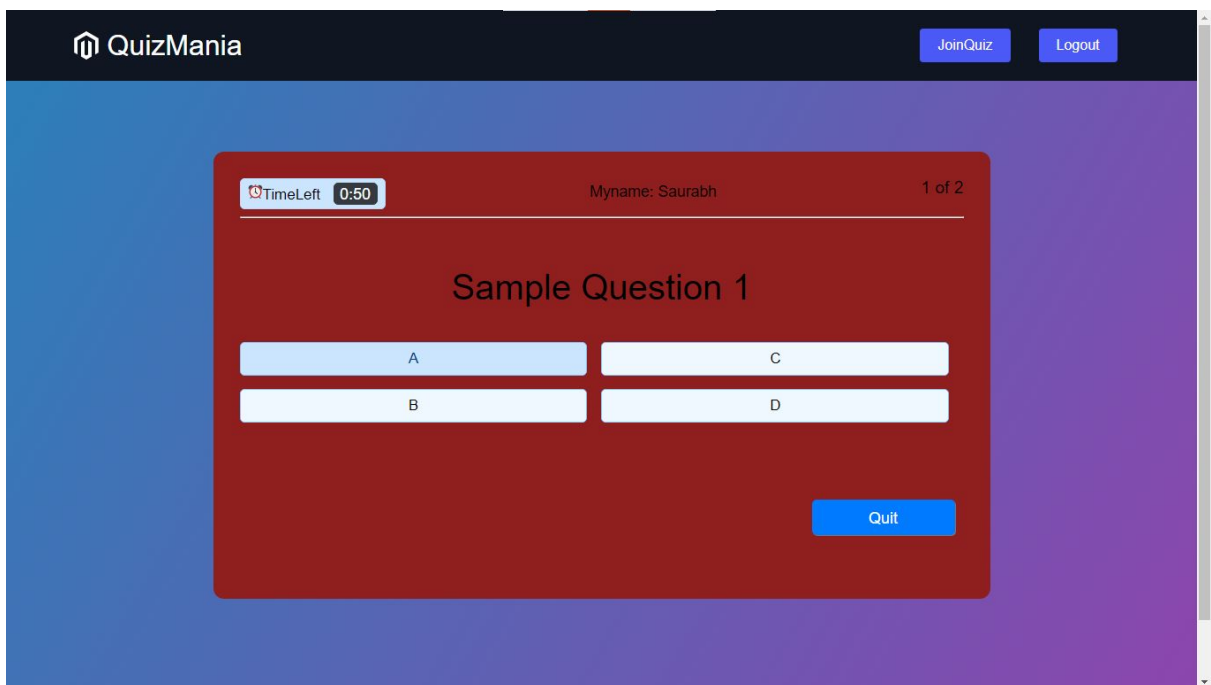
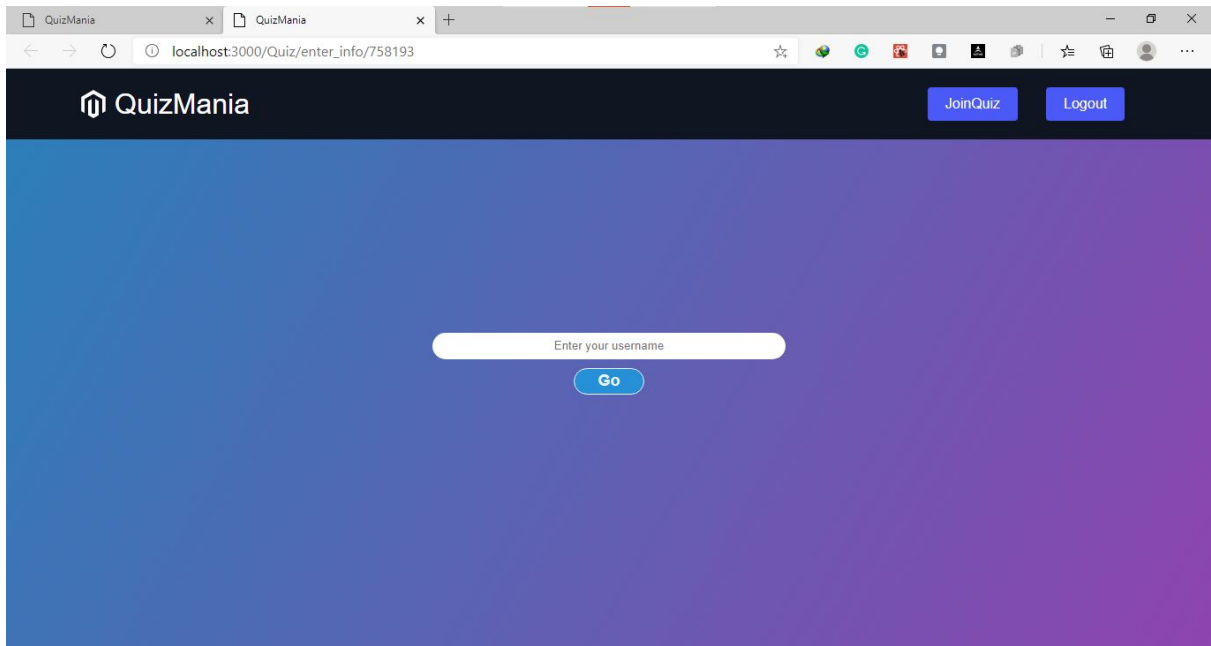
This screenshot shows the same 'Enter Questions' form, but with sample data entered into the fields. The inputs are: 'Sample Quiz 1', 'Sample Question', 'Sample Option 1', 'Sample Option 2', 'Sample Option 3', 'Sample Option 4', and a dropdown menu showing 'Sample Option 4'. The 'Submit' and 'Finish' buttons remain at the bottom.

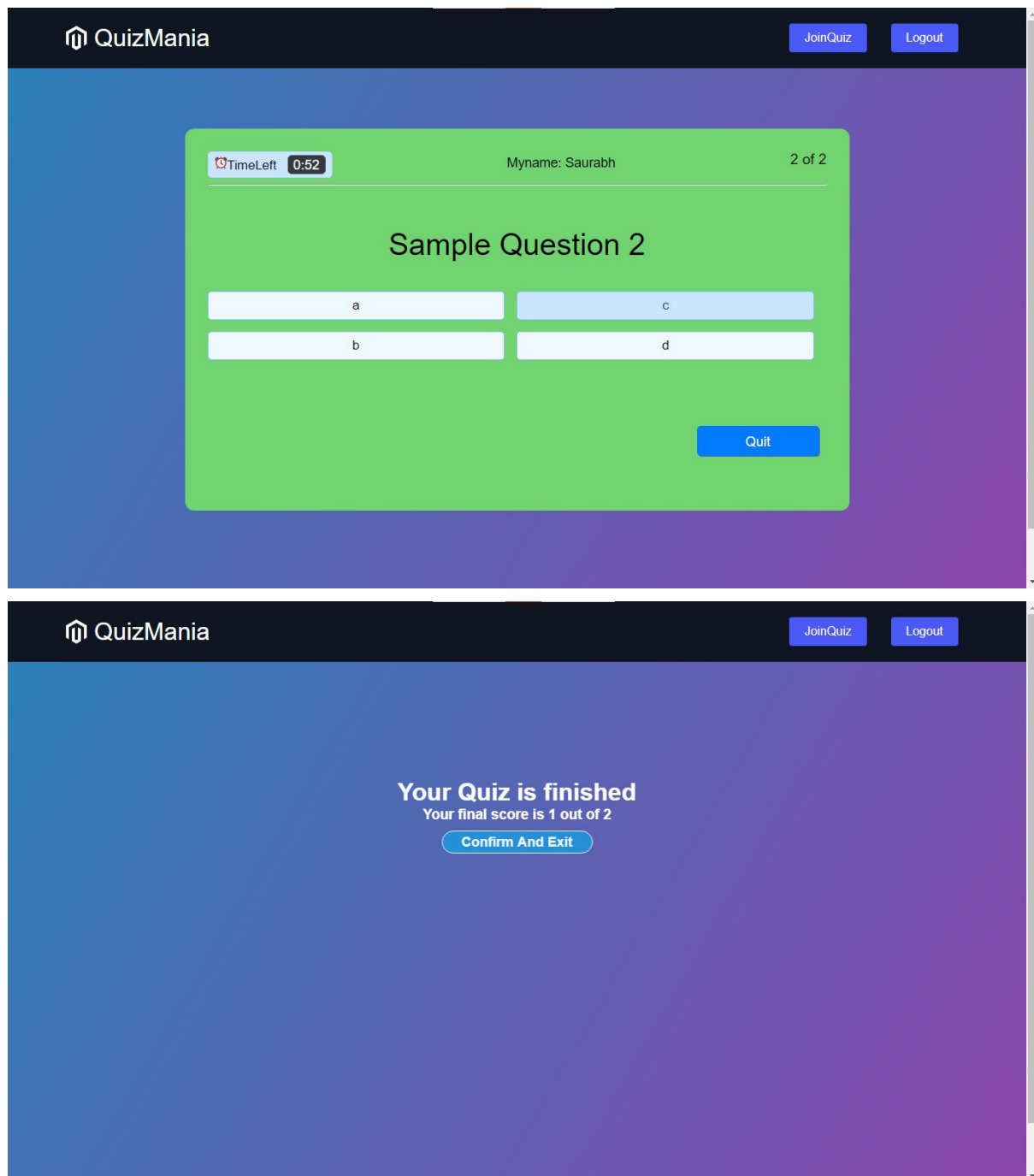
4. Join Quiz:

This section is responsible for joining and starting the Quiz. As soon as a user joins a quiz he is redirected to a page where a name is asked to be entered by which he want to take the quiz as soon as the user enter and confirm the name **a unique user_id** is generated for the user for that quiz only just to make sure the uniqueness and n confusion occurs even if the name is same and the user is redirected to the waiting page where he sees the number of active users also waiting for the quiz to start. As soon as the host starts the quiz , the quiz starts with each question having a time limit of 30 seconds.Each question consists of four options from which only one

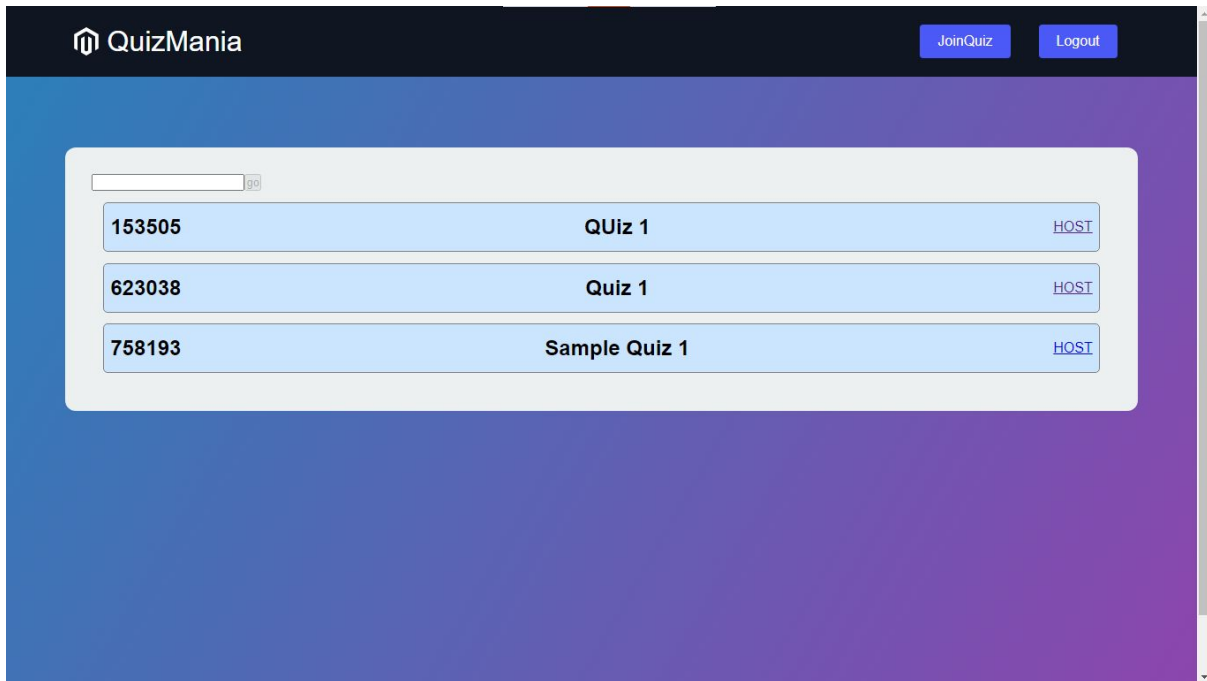
option will be the right answer. And later on when the candidate has completed his/her quiz then it will return a summary of the quiz stats



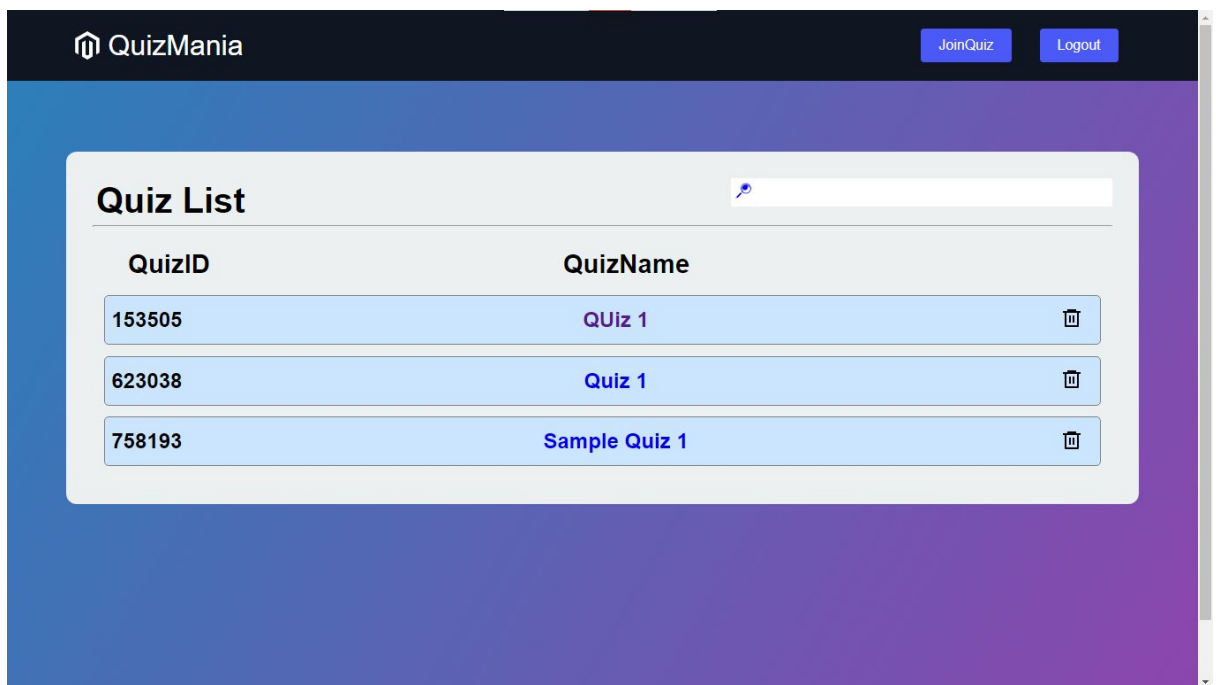




5. Host Quiz: This section is responsible for hosting the quiz. The user gets a list to be hosted when a quiz is selected. The user is redirected to the control panel of the host quiz where he sees the number of users waiting for the quiz to start and also see the stats and final score of a user as soon as he finishes the quiz. The Quiz is Started Only when the host wants it to start.

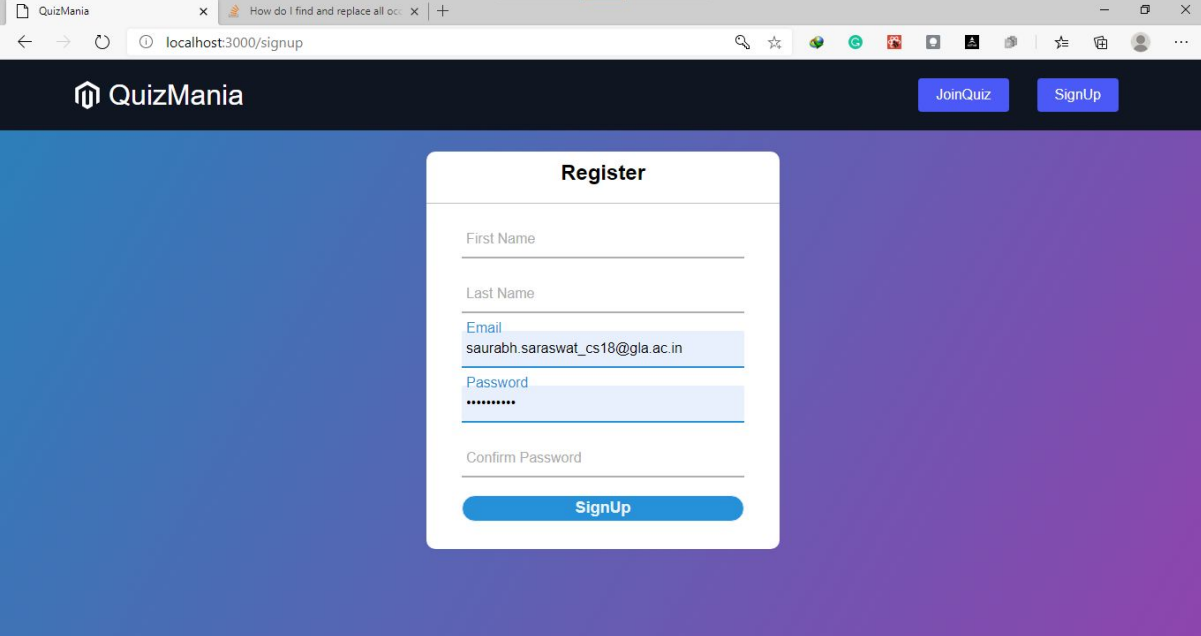


6. View Quiz: This section lists all the quizzes saved by the user. The user can delete the quiz or get details of the quiz by clicking on the name . This is pretty much straight forward.



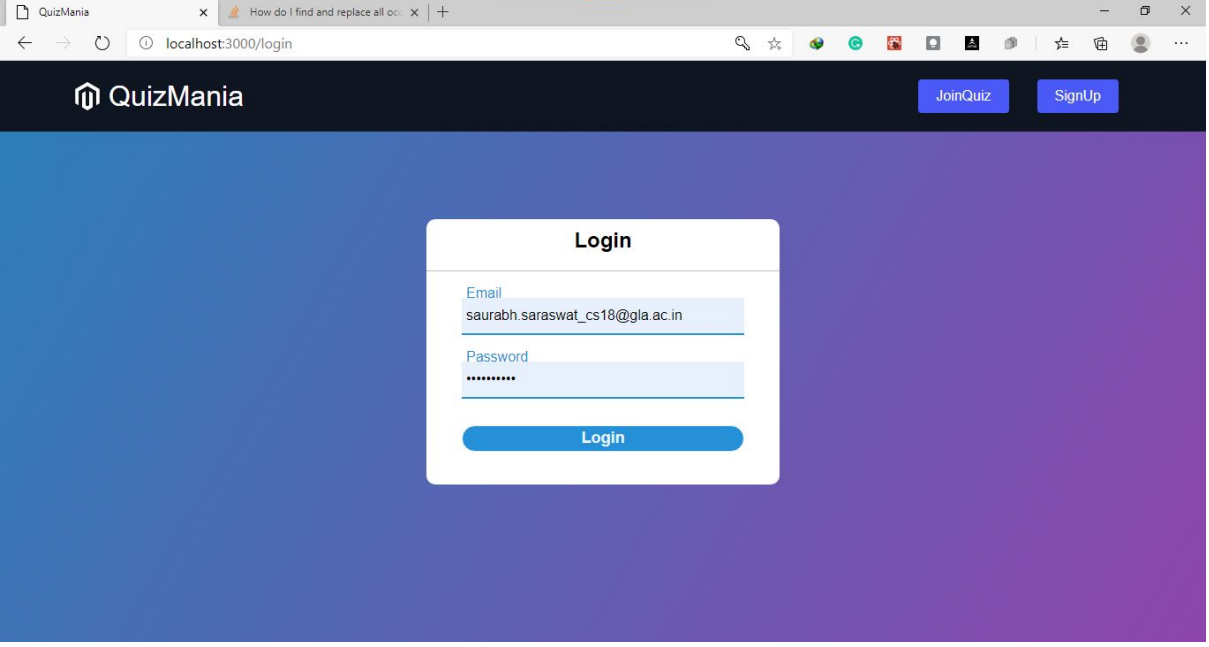
7. Get Quiz : This section displays the details of a particular quiz the user can edit questions, delete a question and Add more Questions .

8. Sign Up : This Section creates a new user profile of the user after which he can create and host quizzes



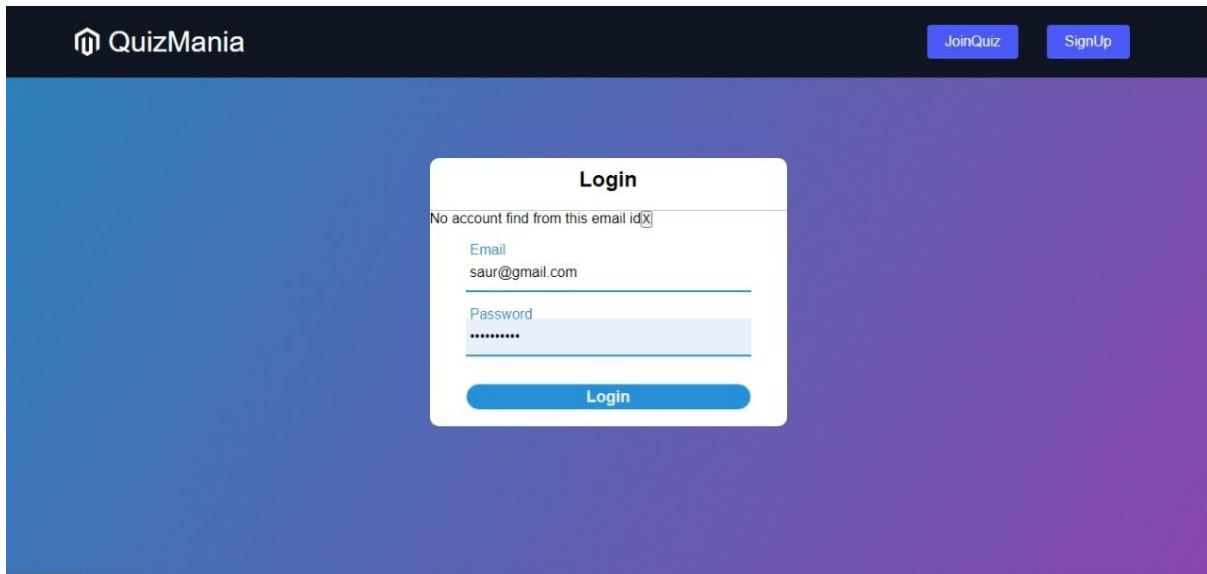
The screenshot shows a web browser window with the URL `localhost:3000/signup`. The page features a dark blue header with the "QuizMania" logo on the left and "JoinQuiz" and "SignUp" buttons on the right. The main content area has a blue-to-purple gradient background. In the center, there is a white "Register" form. The form contains the following fields: "First Name", "Last Name", "Email" (with the value `saurabh.saraswat_cs18@gla.ac.in`), "Password" (masked with dots), and "Confirm Password". A blue "SignUp" button is located at the bottom of the form.

8. Sign in : Make the user logged in and displays dashboard on a success



The screenshot shows a web browser window with the URL `localhost:3000/login`. The page layout is identical to the Sign Up page, with a dark blue header and a blue-to-purple gradient background. In the center, there is a white "Login" form. The form contains the following fields: "Email" (with the value `saurabh.saraswat_cs18@gla.ac.in`) and "Password" (masked with dots). A blue "Login" button is located at the bottom of the form.

9.Error Pages:



QuizMania

JoinQuiz SignUp

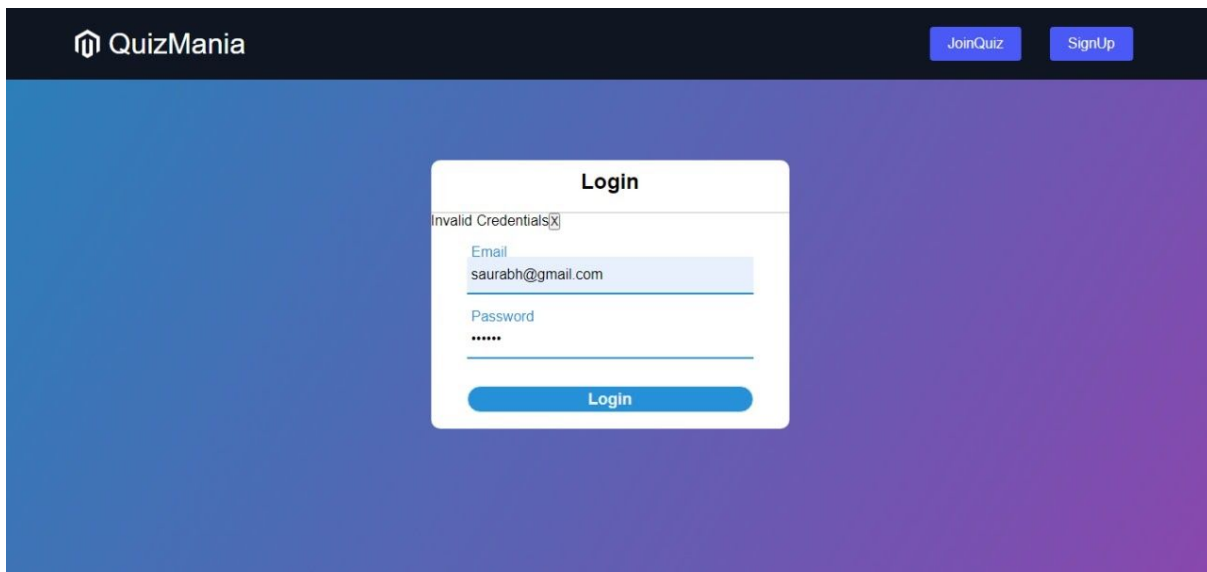
Login

No account find from this email id

Email
saur@gmail.com

Password

Login



QuizMania

JoinQuiz SignUp

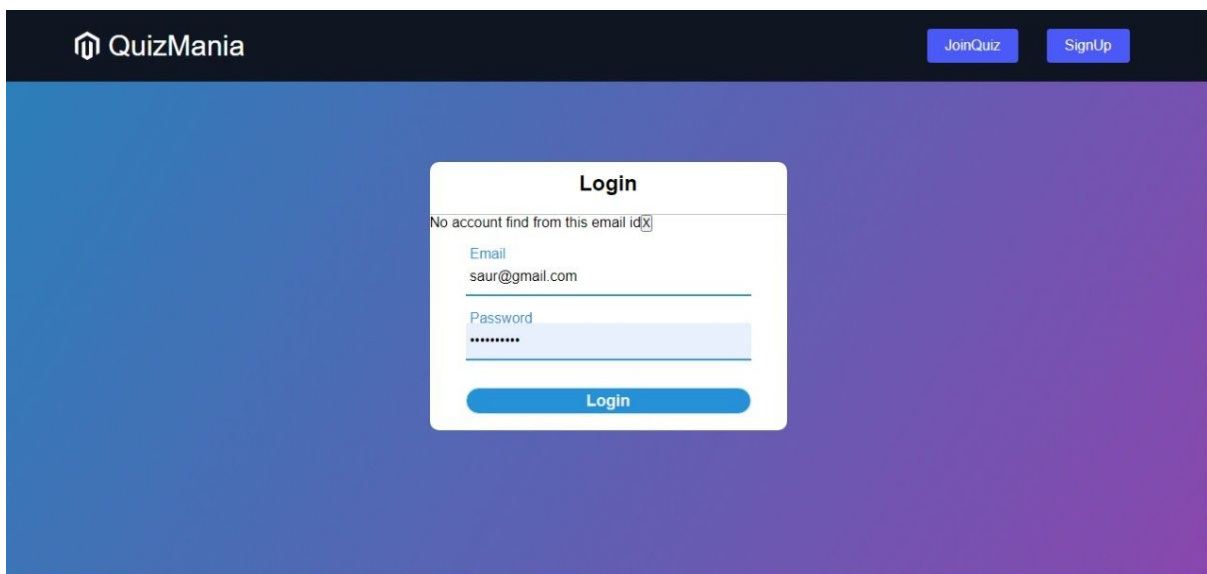
Login

Invalid Credentials

Email
saurabh@gmail.com

Password

Login



QuizMania

JoinQuiz SignUp

Login

No account find from this email id

Email
saur@gmail.com

Password

Login

Code Snippets

Main BackEnd File

```
const router = require('./routers/submitQues')

const router2 = require('./routers/viewQuiz')

const router3 = require('./routers/startQuiz')

const userRoute = require('./routers/UserCradRoute')

const express = require('express');

const { urlencoded } = require('body-parser');

const mongoose = require('mongoose')

const cors = require('cors')

const app = express();

const port = 80;

const server = app.listen(port, () => { console.log('Listening..') })

app.use(urlencoded({ extended: true }))

app.use(cors());

mongoose.connect('mongodb://localhost:27017/quizmania', { useNewUrlParser: true,
useUnifiedTopology: true, useCreateIndex: true })

    .then((result) => server)

    .catch((err) => console.log(err));
```

```

mongoose.connection.once('open', () => {

  console.log('connection made');

}).on('err', (err) => {

  console.log(err);

})

router2(app);

router(app);

router3(app, server);

userRoute(app);

```

Router 1

```

module.exports = (app, server) => {

  // require necessary dependencies
  const mongoose = require('mongoose')

  // require the user schema to save the users
  const users_model = require("../databaseModel/users")

  // compile the schema to modal
  const userModel = mongoose.model('user',
users_model.usersListSchema);
  const bodyParser = require('body-parser');
  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());

  // require socket io for establishing real time connections
  const socket = require('socket.io');
  const io = socket(server);

```

```

var user = {
  user_id: 0,
  disconnected: true
}

var host = {
  disconnected: true,
  notUpdated: true
}

const getQuizData = (quiz_id) => {
  return new Promise((resolve, reject) => {
    userModel.findOne({ quiz_id: quiz_id }).then((response,
err) => {
      if (err) reject(err)
      resolve(response)
    })
  })
}

const checkIfSaved = (response, user_id) => {
  if (response) {
    return response.users.find(user => user.user_id ===
user_id)
  }

  else return false
}

const deleteUser = (quiz_id, user_id) => {

  return new Promise((resolve, reject) => {

    userModel.findOne({ quiz_id: quiz_id }, (err, response) =>
{
      if (err) reject(err)
      const retUser = checkIfSaved(response, user_id)
      if (retUser) {
        userModel.updateOne(
          { "quiz_id": quiz_id },
          {
            $pull: {
              "users": { "user_id": user_id }
            }
          }
        )
      }
    })
  })
}

```

```

        }
        ).then((stats) => {

            resolve(stats)

        })
    }
    })

})

// method to save new users to the database
const saveUser = (userData, quiz_id) => {

    // returning promise to handle errors when function called

    // resolve means success and reject means failure
    return new Promise((resolve, reject) => {

        // finding the quizId if there is one
        userModel.findOne({ quiz_id: quiz_id }, (err, response) =>
{

            // if anytype of error occurs return with reject
            if (err) reject(err)

            // if there is no previous record of the user at the
quiz id

            if (response == null) {

                // initializing new user to be stored
                var newuser = userModel({
                    quiz_id: quiz_id,
                    users: userData
                })
                // save it as a new entry
                newuser.save().then((data) => {
                    console.log("saved");

                    // call the sucess function
                    resolve(data)
                })
            }
        })
    })
}

```



```

        // if a entry already present
        else {

            const retUser = checkIfSaved(response,
responseData.user_id)

            if (retUser) {
                console.log("Already exist");
                resolve(response)
            }
            else {
                // push the user to the userlist of the
response

                response.users.push(userData)

                // resave the updated response
                response.save().then((data) => {
                    console.log("saved");
                    // call the sucess function
                    resolve(data)
                })
            }

        }

    })

})

}

// the method to handle all the realtime connection and events

// called when any new connection establish
io.on('connection', socket => {

    socket.on('start', (data) => {
        io.to(data).emit('startquiz')
    })

    socket.on('host_connected', (quiz_id) => {

        socket.host_id = quiz_id
    })
})

```

```

        socket.join("host" + quiz_id)
        host.disconnected = false

        console.log("host of  quiz_id " + quiz_id + " connected");

        if (!host.disconnected && host.notUpdated) {
            getQuizData(socket.host_id).then((data) => {
                io.to("host" + data.quiz_id).emit('update',
data.users)

                host.notUpdated = false
            }).catch((err) => {
                console.log(err);
            })
        }

    })

    socket.on('quiz_ended', (data) => {
        const sentdata = {
            scores: data,
            username: socket.username
        }
        io.to("host" + socket.quiz_id).emit('a_quiz_ends',
sentdata)

    })

    socket.on('user_connected', (data) => {

        const { quiz_id, username, user_id } = data
        user.user_id = user_id

        socket.username = username;
        socket.quiz_id = quiz_id
        socket.user_id = user_id

        socket.join(quiz_id)
        host.notUpdated = true

        user.disconnected = false;

        console.log(socket.username + "connected on quiz id " +
socket.quiz_id);

```

```

    const userData = {
      username: username,
      user_id: user_id
    }

    saveUser(userData, quiz_id).then((data) => {

      io.to(quiz_id).emit('update_user_list', data)

    })
  })

  socket.on('client_is_updated', (data) => {

    if (host.notUpdated) {
      io.to("host" + data.quiz_id).emit('update', data.users)
      host.notUpdated = false
    }

  })

  // when socket disconnects like leave in behind or quiz is
  completed

  socket.on('disconnect', () => {

    if (socket.username) {

      user.disconnected = true;
      console.log(socket.username + "disconnected waiting for
rejoin");

      const { user_id, quiz_id } = socket;
      setTimeout(() => {
        if (user.disconnected) {
          deleteUser(quiz_id, user_id).then((stats) => {

            console.log(socket.username +
"disconnected");

            getQuizData(quiz_id).then((data) => {
              io.to("host" +
data.quiz_id).emit('update', data.users)
              host.notUpdated = false

```

```

                                io.to(quiz_id).emit('update_user_list',
data)

                                }).catch((err) => {
                                    console.log(err);
                                })

                            })
                        }
                    else {
                        console.log(socket.username + "reconnected ");
                    }
                }, 2000);
            }
            if (socket.host_id) {
                host.disconnected = true
                setTimeout(() => {
                    if (host.disconnected) {
                        console.log("host is disconnected");
                        host.notUpdated = true
                    }
                    else {
                        host.notUpdated = true
                    }
                }, 2000);
            }

        })
    })
}

```

Router 2

```
module.exports = (app) => {
  const mongoose = require('mongoose')
  const model = require("../databaseModel/quizdata")
  const quesmodel = mongoose.model('question', model.queSchema);
  const quizmodel = mongoose.model('quiz', model.QuizSchema);
  const bodyParser = require('body-parser');
  var ObjectId = require('mongodb').ObjectId;
  app.use(bodyParser.urlencoded({ extended: true }))
  app.use(bodyParser.json());

  app.post('/submitques', (req, res) => {

    var ques = {
      questionString: req.body.question.questionString,
      option1: req.body.question.option1,

      option2: req.body.question.option2,

      option3: req.body.question.option3,

      option4: req.body.question.option4,

      correct: req.body.question.correct
    }
    var quiz_id = (req.body.question.quiz_id);
    quizmodel.findOne({ quiz_id: quiz_id }, (err, response) => {
      if (err) return handleError(err);
      if (response == null) {
        var newq = quizmodel({ quiz_id: quiz_id, quizName: req.body.quizName, questions:
ques });
        newq.save().then((result, err) => {
          if (err) console.log(err);
          else
            console.log("new quiz created")
        })
      }
      else {
        response.questions.push(ques)
        response.save().then(() => {
          console.log("old quiz updated");
        })
      }
    })
  })
}
```

```

    })

    var newques = quesmodel(req.body.question).save()
      .then(function (data) {
        console.log("saved")
      });

  })

  app.post('/editques',(req)=>{

    const quiz_id = req.body.quiz_id;
    const id = req.body._id;

    const question = {
      questionString : req.body.questionString,
      option1: req.body.option1,
      option2: req.body.option2,
      option3: req.body.option3,
      option4: req.body.option4,
      correct: req.body.correct,

    }

    quizmodel.update (
      {"quiz_id" :quiz_id,"questions._id":id},
      {
        "$set":{
          "questions.$":question
        }
      }
    ).then(console.log("hii"))
  })

  app.post('/deleteques',(req,res)=>{
    const quiz_id = req.body.id
    const question = req.body.question
    quizmodel.update (
      {"quiz_id" :quiz_id},
      {
        "$pull":{
          "questions":{"_id" : ObjectId(question._id) }
        }
      }
    )
  })

```

```

    ).then((result)=>{
      console.log(result);
    })

  }
)
}

```

Main Front End File

```

import React, { useState, useEffect } from 'react';
//import router for routing
import { BrowserRouter, Route, Switch, } from 'react-router-dom'

// import various components to be rendered
import Home from './Components/MainPagesComp/home'
import insertques from './Components/HomePageComp/insertques';
import viewQuiz from './Components/quizzesInfo/viewQuiz';
import getQuiz from './Components/quizzesInfo/getQuiz';
import startQuiz from './Components/JoinQuizComp/startQuizHome'
import Userinfo from './Components/JoinQuizComp/userinfo';
import Editques from './Components/quizzesInfo/Editques';
import Navbar from './Components/Navbar/Navbar';
import GlobalStyles from '../src/globalStyles'
import createQuiz from './Components/HomePageComp/createQuiz'
import LoginDashBoard from
'./Components/logincomponent/loginDashBoard';
import SignIn from './Components/auth/signIn';
import SignUp from './Components/auth/signUp';
import UserContext from './context/userContext';
import Axios from 'axios';
import cookie from 'js-cookie';
import hostquiz from './Components/JoinQuizComp/hostquiz';
import HostquizPage from './Components/hostQuizComponent/HostquizPage';
import QuizStats from './Components/JoinQuizComp/QuizStats';
import JoinQuizComp from './Components/partials/JoinQuizComp'

function App() {

  const [userData, setUserData] = useState({

```

```

    token: undefined,
    user: undefined
  })
  const checkLoggedIn = async () => {

    var token = cookie.get("auth-token");

    if (token === null) {
      cookie.set("auth-token", "");
      token = "";
    }

    const tokenRes = await Axios.post(
      'http://192.168.43.91:80/tokenIsValid',
      null,
      { headers: { "x-auth-token": token } }
    );

    if (tokenRes.data) {
      const userRes = await Axios.get("http://192.168.43.91:80/auth", {
        headers: { "x-auth-token": token },
      });

      setUserData({
        token,
        user: userRes.data,
      });
      // console.log(userData);
    }
  };

  useEffect(() => {

    checkLoggedIn();
  }, []);

  const defaultRoutes = () => {
    return (
      <div>
        <div className="App">
          <Navbar />

          { /* Routing for the different pages */ }
          <Switch>

            {(userData.user) ?
              // if user login then this component is available

```



```

        (<Route exact path="/" component={LoginDashBoard} />) :
        //else this
        (<Route exact path="/" component={Home} />)

    }
    <Route path="/createquiz" component={createQuiz} />
    <Route path="/getQuiz/:quiz_id" component={getQuiz} />
    <Route path="/insertques/:quiz_id" component={insertques}
/>

    <Route path="/hostquiz/:quiz_id" component={HostquizPage}
/>

    <Route path="/viewquiz" component={viewQuiz} />
    <Route path="/hostquiz" component={hostquiz} />
    <Route path="/joinquiz" component={JoinQuizComp} />
    <Route path="/quizstats" component={QuizStats} />
    <Route exact path="/login" component={SignIn} />
    <Route path="/signup" component={SignUp} />
    <Route path="/edit/:quiz_id" component={Editques} />
    <Route exact path="/Quiz/enter_info/:quiz_id"
component={Userinfo} />
    <Route exact path="/startQuiz/:quiz_id"
component={startQuiz} />
    </Switch>

    </div>

    </div>
)
}

return (
    <BrowserRouter>
        <UserContext.Provider value={{ userData, setUserData }}>
            <GlobalStyles />
            <Switch>
                <Route component={defaultRoutes} />
            </Switch>
        </UserContext.Provider>
    </BrowserRouter>
);
}

export default App;

```

QUIZOnGoing

```
import React, { Component } from 'react'
import { Redirect } from 'react-router-dom'

// importing the error component
import Directaccess from '../errComponents/DirectAccess'

import '../stylesheets/quiz.css'

import { FcAlarmClock } from 'react-icons/fc'

class QuizOngoing extends Component {

  // constructor fr initially setting state values
  constructor(props) {
    super(props)

    this.state = {

      // for kepping the normal brown background i.e., no button
      // is pressed
      backgroundClass: false,

      // setting the background green or red
      choice: false,

      // the current question to be displayed
      currentQuestion: {},

      // the next questin to be displayed
      nextQuestion: {},

      // the correct answer of the question
      answer: '',

      // Total number of questions in the quiz
      numberOfQuestions: 0,

      // the number of answered questions
      numberOfAnsweredQuestions: 0,
```

```

        // for fetching the current question from the array
        currentQuestionIndex: 0,

        // the total score
        score: 0,

        // number of wrong answers
        wrongAnswers: 0,

        // time allotted for the quiz
        time: {
            minutes: 0,
            seconds: 0
        }
    }

    // for setting time current interval is 0
    this.timeInterval = null
}

componentDidMount = () => {

    console.log("component did mount is called");
    // getting data from the location
    const Data = this.props.data
    console.log(Data.socket);

    // checking if the data is undefined (in case of direct access)

    if (Data !== undefined) {

        // getting quiz from the state if is not undefined
        const quiz = Data.quiz
        const username = Data.username
        const socket = Data.socket

        // setting state of the component with the details of the
quiz fetched
        this.setState({
            quiz: quiz,
            username: username,
            questions: quiz.questions,
            socket : socket

```

```

        },

        // callback function that is executed after the state
is set
        () => {

            // getting the values from state as the function
is called after the state is set so this will not be undefined
            const { questions, currentQuestion, nextQuestion }
= this.state;

            // calling the display question option
            this.displayQuestions(questions, currentQuestion,
nextQuestion)

        })

        // starting the timer
        this.startTimer(quiz.questions.length);
    }
}

// function to display questions in the quiz takes four arguments
displayQuestions =

// the questions array that has all the questions of the quiz
(questions = this.state.questions,

// the current question to be displayed
currentQuestion,

// the next question to be displayed
nextQuestion,

// the index of the current question for transition
currentQuestionIndex = 0) => {

    // if the index is equal to the length i.e., all the
questions are answered
    if (currentQuestionIndex === (questions.length)) {

        // end the quiz

```

```

        this.endQuiz();
    }

    // if the current index is less i.e, questions are left to
be answered
    if (currentQuestionIndex <= questions.length - 1) {

        // fetching the current question index
        // this step is necassry because we will keep updating
the current index for next question
        currentQuestion = questions[currentQuestionIndex];

        // getting next question
        nextQuestion = questions[currentQuestionIndex + 1];

        // storing the correct answer
        const answer = currentQuestion.correct;

        // setting the total number of questions as it is to be
displayed as a info
        const numberOfQuestions = questions.length

        // setting the state with the details
        this.setState({
            currentQuestionIndex,
            currentQuestion,
            nextQuestion,
            answer,
            numberOfQuestions

        })
    }

}

// start timer function takes total no of questions as input
startTimer = (numberOfQuestions) => {

    // setting the timer according to number of questions
    const countDown = Date.now() + (numberOfQuestions * 60) * 500

    // 2 sec more to compunsate the loading time
    + 2000;

```

```

        // setting the time interval that will keep executing again and
again after a certain time limit
        this.timeInterval = setInterval(() => {
            const now = new Date();
            const distance = countdown - now

            // getting left minutes and seconds
            const minutes = Math.floor((distance) % (1000 * 60 * 60) /
(1000 * 60))
            const seconds = Math.floor((distance) % (1000 * 60) /
(1000))

            // if the time is up
            if (distance < 0) {

                // clear the interval
                clearInterval(this.innerHTML);

                // setting time to zero
                this.setState({
                    time: {
                        minutes: 0,
                        seconds: 0
                    }
                },

                // end the quiz
                () => {
                    this.endQuiz();
                })
            }

            // if time is left
            else {

                // setting state every 1 sec
                this.setState({
                    time: {

                        minutes,
                        seconds

                    }
                })
            }
        })

```

```

    }

    },

    // time interval of 1 sec
    1000)

}

// a function to end the quiz
endQuiz = () => {

    // alerting that the quiz has ended
    alert('QUIZ ENDED')

    const quizdata = {
        score : this.state.score,
        total : this.state.numberOfQuestions
    }
    var socket = this.state.socket
    socket.emit('quiz_ended',quizdata)

    // after a while
    setTimeout(() => {
        this.setState({
            redirect: true
        })
    },

    // half second wait
    500)

}

// when any option is clicked
handleSubmit = (e) => {

    // comparing the clicked optin with the answer of the question

    // if the answer is correct
    if (e.target.innerHTML === this.state.answer) {

        // execute the right option function
        this.rightChosen()
    }
}

```

```

        // if the answer is wrong
        else {

            // execute the wrong option function
            this.wrongChosen()
        }

    }

    // if the option is correct
    rightChosen = () => {

        // updating the values of state

        // currentState is the current values of state
        this.setState(currentState => ({

            // setting background classs to false as the background
needs to change
            backgroundClass: true,

            // setting choice true for green color
            choice: true,

            // increasing no of answered question by 1
            numberOfAnsweredQuestions:
currentState.numberOfAnsweredQuestions + 1,

            // increasing the score as the option chosen is right
            score: currentState.score + 1,
        })),

        // called when the state is updated
        () => {

            // time out function to show the background color for a
while
            setTimeout(

                // a self executing function
                function () {

                    // getting the updated values of state

```



```

const {

    // the array of questions
    questions

    // the current question that is answered
    , currentQuestion,

    // the next question to be displayed
    nextQuestion,

    // the updated index for the next question
as this will update the current question
    currentQuestionIndex }

    = this.state;

    // calling the display question function to
update the current displayed question
    this.displayQuestions(questions,
currentQuestion, nextQuestion, currentQuestionIndex + 1)

}

    // binding this with the timeout function this
keyword is used in the function
    .bind(this),

    // the time interval for the timeout function
    1100
);

// another timeout function to set the background back
to normal

setTimeout
(

    function () {

        this.setState({
            backgroundClass: false
        })

        }.bind(this), 1000

```

```

        );

    })

}

// if the answer chosen is wrong
wrongChosen = () => {

    this.setState(currentState => ({
        backgroundClass: true,

        // choice set to false for red color
        choice: false,

        numberOfAnsweredQuestions:
currentState.numberOfAnsweredQuestions + 1,

        // increasing the wrong answers
        wrongAnswers: currentState.wrongAnswers + 1,
    })), () => {

        // time out functions same as above
        setTimeout(
            function () {

                const { questions, currentQuestion, nextQuestion,
currentQuestionIndex } = this.state;
                this.displayQuestions(questions, currentQuestion,
nextQuestion, currentQuestionIndex + 1)

            }
            .bind(this),
            1100
        );
        setTimeout(
            function () {

                this.setState({
                    backgroundClass: false
                })
            }
            .bind(this),
            1000
        );
    }
}

```

```

    })
  }

  render() {

    if(this.state.redirect){
      const quizdata = {
        score : this.state.score,
        total : this.state.numberOfQuestions
      }
      return(
        <Redirect to = {{
          pathname : '/quizstats',
          state: quizdata
        }}/>
      )
    }

    // if the state is not defined (direct acess)

    if (this.state.quiz === undefined) {

      // render the error component
      return (
        <Directaccess />
      )
    }

    // if the quiz is valid
    else {

      // getting details for rendering

      const {

        // to be displayed
        currentQuestion,

        // to be displayed as information
        numberOfQuestions,

        // for current number of question

```

```

        currentQuestionIndex,

        // time allotted for quiz
        time,
        backgroundClass,
        choice
    } = this.state

    // if question is loading because of any issue
    if (currentQuestion === {}) {

        return (

            // just display loading message
            <h1>Loading</h1>

        )
    }

    // if fetched
    else {

        return (

            // setting class of the background dyanamically

            <div

                // checking if the the normal background
                className={backgroundClass ?

                    // if background is to be changed i.e, any
one option is selected
                    (choice ?
                        // if choice is true add green to class
                        ' questions green' :

                        // if option is wrong add red to class
                        'questions red ') :

                    // if background is normal no color is set
                    'questions'}>

                <div className="details-container">
                    <div className="timer">

```

```

        <div
className="time_text"><FcAlarmClock/>TimeLeft{/* getting time from the
state */}</div>

        <div className="time">{time.minutes}
:
{time.seconds}
</div>
</div >
{ /*<span className="shiftBelow"> Score:
{this.state.score}</span>*/}
        <span className="shiftBelow"> Myname:
{this.state.username}</span>

        <p>
            <span>
                <span className="question-no">
                    {currentQuestionIndex + 1} of
{numberOfQuestions}
                </span>
            </span>
        </p>

    </div>
    <hr></hr>
    <div className="questionString">

        {/* Displaying current question string */}
        <p>{currentQuestion.questionString}</p>
    </div>
    <div className="options-container">

        {/* Displaying option */}
        <p onClick={this.handleSubmit}
className="option">{currentQuestion.option1}</p>
        <p onClick={this.handleSubmit}
className="option">{currentQuestion.option2}</p>

    </div>
    <div className="options-container">
        <p onClick={this.handleSubmit}
className="option">{currentQuestion.option3}</p>
        <p onClick={this.handleSubmit}
className="option">{currentQuestion.option4}</p>

    </div>

```

```

        <div className="btn-container">
            <button onClick={this.endQuiz}
className="quit-btn">Quit</button>
        </div>
    </div>
    )
    }
}

}

}

export default QuizOngoing

```

References

- <https://www.geeksforgeeks.org/mern-stack/>
- <https://docs.mongodb.com/>
- <http://expressjs.com/>
- <https://reactjs.org/>
- <https://nodejs.org/>
- <https://www.tutorialspoint.com/socket.io/index.htm>
- <https://socket.io/get-started/>
- <https://learnredux.com/>
- <https://www.tutorialspoint.com/redux/index.htm>
- <https://www.youtube.com/watch?v=OxIDLw0M-m0&list=PL4cUxeGkcC9ij8CfkAY2RAGb-tmkNwQHG>
- https://www.youtube.com/watch?v=yZ0f1Apb5CU&list=PL4cUxeGkcC9i0_2FF-WhfRifIJ1IXITZR
- <https://www.youtube.com/watch?v=w-7RQ46RgxU&list=PL4cUxeGkcC9gcy9IrvMJ75z9maRw4byYp>