**Quiz App for Ethiopia Ministry takers**

* **The main idea(purpose) of the app:**

It is a forum that asks you whether you are in grade 6, 8 or 12. Then after you choose your grade it will allow you to access model questions that have been added to the website. The question will be listed from the latest to the oldest. The new books will be at the top of the webpage. After clicking the question link it will redirect or take you to the page and will ask you the questions turn by turn or as a slide show at the top it will show you the questions you have done out of all. Then it will show your results as follow first the question you get it out of all, second score out of hundred (100%), third wrong answers out of all and finally calculate his IQ point(point he get from the quiz by dividing the score out of all and multiplying the result by 100 and divide it again by 20).

* Create a Quiz App for ministry takers in Ethiopia. It must contain:

**Backend**

A backend made from MySQL to accumulate or contain the following data:

1. There must be two tables’ questions and user tables.
2. A sign in table which asks you: your email, registration number (It must be 7 numbers), name (All letters of yours, fathers and grand father’s name must be capitalized), Woreda number(It must be 2 digits), school name (All letters must be capitalized), city and sub-city name, age(it must be below 90 and above 5 to access), gender(Whether you are male or female) and account password(It must be more than 8 character, contain special characters and numbers).
3. A login table which asks you: your email, registration number (It must be 7 numbers) and password.
4. A private route (admin panel) which allows the Admin to update, delete and upload questions in a very easy system. It must be protected by a login page which appears when the website is loaded.
5. Create a config file to create tables and connect with the server.
6. Env file to store secret code.

**Frontend**

A frontend made from React to serve or display the following pages:

1. A sign in page which asks you: your email, registration number (It must be 7 numbers), name (All letters of yours, fathers and grand father’s name must be capitalized), Woreda number(It must be 2 digits), school name (All letters must be capitalized), city and sub-city name, age(it must be below 90 and above 5 to access), gender(Whether you are male or female) and account password(It must be more than 8 character, contain special characters and numbers).
2. A login page which asks you: your email, registration number (It must be 7 numbers) and password.
3. A home page that displays all questions listed in the database. (From oldest to latest)
4. A question page which appears when you click the question link in the home page.
5. At the header the logo or icon of the user must be displayed.
6. The IQ point or the points gained from doing models must be displayed at the left top corner. It must be calculated by dividing the score of the question you do out of all and multiplying it by hundred (100%) then divide it again to 20, finally add it to the current IQ point.

**Admin panel (Private route)**

1. A login page which asks only for the password of the admin account. It must be the landing page when you load the website.
2. A home page which allows you to delete a model booklet.
3. An uploading route to upload model questions. In the private route (admin panel) you must take categories (subject), questions, choice…

* The language you can use is:

**For the backend:**

* **MySQL**: To create connection with the server.
* **PhpMyAdmin**: To store data.
* **Express**: To create a local server.
* **NodeJS**: To run your backend.
* **Dotenv**: To protect your secret code.

**For the frontend:**

* **React:** For efficiency and speed.
* **Axios:** To fetch data from the backend.
* **React-hook-form:** To create an uploading system and register or login form.
* **React-router**-**dom:** To route over different pages.

**For the private route (admin panel):**

* **React:** For efficiency and speed.
* **Axios:** To fetch data from the backend.
* **React-hook-form:** To create an uploading system and register or login form.
* **React-router**-**dom:** To route over different pages.

**Bonus**

> The IQ point of the user must be saved in the database, to remember when the user comes back.

>You must deploy the app using AWS.

* Then you will follow:
* First create the tips or notes I must follow.
* Second, add the note I need to understand the system of private routes.
* Third, add the note I need to understand the system of frontend and backend.
* Fourth display the file structure of the frontend, backend and private route (admin panel).
* Fifth display the code for the config file which creates the tables in it once.
* Sixth, do the bonus part.

\*\*Tips/Notes:\*\*

1. \*\*Database Security:\*\* Ensure that your MySQL database is securely configured, and use parameterized queries or prepared statements to prevent SQL injection attacks.

2. \*\*Password Security:\*\* Hash passwords before storing them in the database using a strong hashing algorithm (e.g., bcrypt). Never store plain text passwords.

3. \*\*Backend Security:\*\* Implement user authentication and authorization. Use middleware to protect routes and validate user roles.

4. \*\*Environment Variables:\*\* Utilize the dotenv package to store sensitive information such as database credentials and API keys securely.

5. \*\*Code Organization:\*\* Follow a modular and organized code structure to enhance maintainability and scalability.

6. \*\*React Components:\*\* Break down your React components into smaller, reusable components for better code organization and readability.

7. \*\*React Routing:\*\* Use React Router for client-side routing in your React application. Define routes for different pages.

8. \*\*State Management:\*\* Consider using React state or a state management library (e.g., Redux) to manage the state of your application.

9. \*\*File Uploads:\*\* Implement a secure file upload system for the admin panel. Validate file types and sizes to prevent potential security risks.

10. \*\*AWS Deployment:\*\* Follow AWS best practices for deployment. Use AWS services like EC2 for hosting and RDS for database hosting. Configure security groups and IAM roles appropriately.

\*\*Understanding Private Routes:\*\*

- Private routes are routes that require authentication to access. In your case, the admin panel should be a private route accessible only after successful admin login.

- Use a higher-order component (HOC) or React context to create a wrapper around your admin-related components. This wrapper should check if the user is authenticated (logged in as admin) before rendering the component.

- Redirect unauthenticated users to the login page or another appropriate page.

\*\*Understanding Frontend and Backend:\*\*

- The frontend is responsible for presenting the user interface and interacting with users. It communicates with the backend to fetch or send data.

- The backend handles business logic, interacts with the database, and serves data to the frontend through APIs (Application Programming Interfaces).

- Use Axios in your React frontend to make HTTP requests to your Node.js/Express backend.

- React-hook-form can be used for form handling, validation, and submission.

\*\*File Structure:\*\*

1. \*\*Frontend:\*\*

```

/src

/components

- SignInForm.jsx

- LoginForm.jsx

- HomePage.jsx

- QuestionPage.jsx

- AdminPanel.jsx

/pages

- SignInPage.jsx

- LoginPage.jsx

- Home.jsx

- Question.jsx

/utils

- api.js (Axios instance)

/App.jsx

/index.jsx

```

2. \*\*Backend:\*\*

```

/backend

/config

- database.js

- server.js

/controllers

- userController.js

- questionController.js

/middlewares

- authMiddleware.js

/routes

- userRoutes.js

- questionRoutes.js

- adminRoutes.js (private)

/uploads

- uploadedFiles...

```

3. \*\*Admin Panel (Private Route):\*\*

```

/admin

/components

- AdminLoginForm.jsx

- AdminHomePage.jsx

- UploadForm.jsx

/pages

- AdminLogin.jsx

- AdminHome.jsx

- Upload.jsx

/utils

- adminApi.js (Axios instance for admin)

/AdminApp.jsx

/AdminIndex.jsx

```

\*\*Config File (backend/config/database.js):\*\*

```javascript

const mysql = require('mysql');

const dotenv = require('dotenv');

dotenv.config();

const connection = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

});

connection.connect((err) => {

if (err) {

console.error('Database connection failed: ', err);

} else {

console.log('Connected to the database');

}

});

module.exports = connection;

```

\*\*Bonus:\*\*

1. \*\*Saving IQ Points to Database:\*\*

- Create a new table (e.g., user\_stats) to store user-related statistics, including IQ points.

- Update the user table to include a foreign key relationship with the user\_stats table.

- When a user completes a quiz, update the corresponding IQ points in the user\_stats table.

2. \*\*AWS Deployment:\*\*

- Set up an EC2 instance for your backend and React frontend.

- Use RDS for your MySQL database.

- Configure security groups to allow traffic only from necessary sources.

- Set up an S3 bucket for file storage (e.g., question uploads).

- Use Elastic Beanstalk or other AWS services for easier deployment management.

Remember to handle errors gracefully and regularly backup your database during deployment and updates.