**React Component: ViewQuizzes.js - Quick Interview Refresher**

**🌟 Purpose:**

This component allows instructors to:

* View all quizzes under a selected course
* Create new quizzes
* View questions inside a quiz
* Add questions to a selected quiz

**📊 State Variables:**

| **State** | **Description** |
| --- | --- |
| quizzes | Stores quizzes fetched for a course |
| loading | Shows if data is being fetched |
| error | Error messages for the user |
| message | Info messages like success or no questions found |
| selectedQuizId | Tracks which quiz is selected to view questions |
| questions | Stores questions for the selected quiz |
| viewingQuestions | true = viewing questions; false = viewing quiz list |
| previousPage | Stores referrer to enable back navigation |
| currentQuizTotalMarks | Total marks for the current quiz |
| showCreateQuizForm | Boolean for showing/hiding the quiz creation form |

**🔗 Local Storage Used:**

* email: used to get token and instructorId
* selectedCourseTitle: course name (shown on UI)
* {email}\_token: for API authorization
* {email}\_uuid: instructor's unique ID

**⚙️ Key Functions:**

**1. fetchQuizzes()**

* API: /api/quiz/viewAll?courseId=...
* Fetches all quizzes for the course
* Handles loading and error states

**2. handleQuizSelect(quizId, totalMarks)**

* Sets selected quiz
* Switches view to questions list
* Calls fetchQuestions()

**3. fetchQuestions(quizId)**

* API: /api/question/view?quizId=...
* Fetches questions for a quiz
* Sets questions array or message if empty

**4. handleQuestionAdded()**

* Re-fetches questions after a new one is added

**5. handleBack()**

* Navigates back to quiz list or course page depending on current view

**6. handleCreateQuizButtonClick()**

* Opens quiz creation modal

**7. handleQuizCreated()**

* After quiz creation, refetches quiz list
* Closes the modal

**8. handleCloseCreateQuizForm()**

* Just closes the quiz form without creating

**📄 JSX UI Breakdown:**

* If loading: shows "Loading..."
* If error: shows error message
* If showCreateQuizForm: shows modal with CreateQuizForm
* If viewingQuestions: shows
  + Quiz title
  + Questions list (QuestionList)
  + AddQuestionForm
* Else: shows QuizList of all quizzes
* Always shows a Back button

**📁 Child Components:**

* CreateQuizForm: Handles creating a new quiz
* QuizList: Displays quizzes, lets user select one
* QuestionList: Shows list of questions in a quiz
* AddQuestionForm: Form to add new question to quiz

**🚀 Tips for Interview:**

* Understand how **state** is used to control views.
* Know how **API calls** are handled with Axios.
* Be able to explain how child components work together.
* Know how React Router hooks like useParams and useNavigate are used.

**if (response.data && response.data.success && response.data.data)**

1. **The response exists**
2. **success is true**
3. **The data array exists**

**🔍 document.referrer — What is it?**

* document.referrer is a **built-in browser property**.
* It tells you **which page the user came from** (the previous URL) before landing on the current one.

**React Component: QuestionsDisplay.js - Interview Refresher**

**🌟 Purpose:**

This component allows a **student** to:

* View quiz instructions
* Take a quiz (one question at a time)
* Track remaining time using a timer
* Submit quiz responses
* View results (score and percentage)

**📊 State Variables:**

| **State** | **Description** |
| --- | --- |
| questions | Stores all questions for the quiz |
| loading | True when data is being fetched |
| error | Holds error messages if API calls fail |
| selectedAnswers | Stores selected options per question (key = questionId) |
| score | Stores user's final score after submission |
| percentage | Stores final percentage result |
| quizDetails | Quiz metadata like title, total marks, etc. |
| isTimerRunning | Boolean to control the timer state |
| showQuizInfoPopup | True initially to show quiz instructions before starting |
| totalQuestionsCount | Total number of questions in quiz |
| currentQuestionIndex | Index of the currently visible question |

**🔹 Local Storage Used:**

* email: gets user's token & UUID
* {email}\_token: auth token for headers
* {email}\_uuid: user ID for submission

**⚙️ Key Functions:**

**1. fetchInitialData()**

* Calls two APIs:
  + /api/quiz/view?quizId=... for quiz info
  + /api/question/view?quizId=... for questions
* Sets state for quizDetails, questions, and totalQuestionsCount

**2. calculateQuizDuration()**

* Returns quiz duration: 60 sec \* number of questions

**3. handleOptionChange(questionId, option)**

* Updates selectedAnswers with user's selected option for a question

**4. handleSubmit()**

* Creates a payload of selected answers and sends a PUT request to /api/squiz/submit
* Sets score and percentage based on response

**5. handleTimeUp()**

* Triggered by Timer component when time ends
* Calls handleSubmit()

**6. handleStartQuiz()**

* Hides instruction popup and starts the timer

**7. goToNextQuestion() / goToPreviousQuestion()**

* Navigate between questions using currentQuestionIndex

**📄 JSX UI Breakdown:**

* **Loading:** shows loading text
* **Error:** shows error message
* **Before Start:** shows QuizInfoPopup with quiz details
* **During Quiz:**
  + Header shows title and Timer
  + Displays one QuizQuestion at a time
  + Navigation: Previous / Next buttons
  + Submit button (disabled until all questions answered)
* **After Submission:** shows QuizResults

**📁 Child Components:**

* QuizInfoPopup: Popup with quiz metadata before starting
* QuizQuestion: Displays one question and options
* QuizResults: Displays score and percentage after quiz
* Timer: Countdown timer for quiz duration

**🧠 Tips for Interview:**

* Know how to manage multiple states (quiz, answers,gt
* timer)
* Understand how selected answers are mapped and submitted
* Be able to explain conditional rendering based on quiz state
* Know how useEffect fetches data when component mounts
* Bonus: selectedAnswers is dynamically built with question IDs as keys!

the selected options are temporarily held in memory **until** the user clicks the **"Submit Quiz"** button — that’s when the following happens:

1. A payload is built with all selected answers.
2. The axios.put() request sends the data to the backend (/api/squiz/submit).
3. Backend processes and stores it in the DB.

**✅ Why not just use useState?**

Because:

* If you store timer ID in useState, updating it would cause a re-render (which we don't want).
* But useRef lets us **store it safely without re-rendering**.

**🤔 What is useRef?**

In React, useRef() is a special hook that gives you a **persistent container** that:

* Does **not** trigger a re-render when it changes.
* Can **store a value** (like a variable) across component renders.

**timerIdRef.current = setInterval(() => {**

**setTimeLeft(prevTime => prevTime - 1);**

**}, 1000);**

**This line is saying:**

"Hey browser, run this function every 1 second. Inside that function, reduce the remaining time (timeLeft) by 1."

**What is react modal??**

**When you're building a web app, sometimes you need a pop-up window (called a "modal") to show information or ask the user to do something (like filling out a form, confirming a choice, etc.). A modal is a special type of window that appears on top of your current page, blocking the rest of the content until the user interacts with it.**

**Why use the spread operator?**

* **Immutability: React requires you to update state immutably (i.e., without changing the original state directly). The spread operator helps you create a new object with the updated values rather than modifying the existing one.**
* **Merging Objects: The spread operator allows you to merge two objects. If there are any overlapping properties (e.g., if both prev and response.data.data have the same username key), the properties from response.data.data will overwrite the ones from prev.**

**Example:**

**Let's assume:**

* **prev = { 1: 'Alice', 2: 'Bob' }**
* **response.data.data = { 3: 'Charlie', 4: 'David' }**

**Using the spread operator:**

**js**

**const newUsernames = { ...prev, ...response.data.data };**

**This will result in:**

**jsnewUsernames = { 1: 'Alice', 2: 'Bob', 3: 'Charlie', 4: 'David'};**

**🧾 JSON and JavaScript – What’s the deal?**

**JSON (JavaScript Object Notation) is a format used to exchange data between a server and a web app. It looks like JavaScript objects but it's actually just a string.**

**So you need to:**

* **🔁 Convert JSON string ↔ JavaScript object when working with APIs.**

**🔄 Convert JSON to JavaScript object (parse)**

**This means you're taking a JSON string and turning it into a real usable object in JS.**

**js**

**CopyEdit**

**const jsonString = '{"name": "Asma", "age": 22}';**

**const jsObject = JSON.parse(jsonString);**

**console.log(jsObject.name); // Output: Asma**

* **✅ JSON.parse() → Takes a string, gives you a usable JS object.**

**🔄 Convert JavaScript object to JSON (stringify)**

**This is when you're sending data to a server, like in a POST request. Servers want it as JSON string, not JS object.**

**js**

**CopyEdit**

**const jsObject = { name: "Asma", age: 22 };**

**const jsonString = JSON.stringify(jsObject);**

**console.log(jsonString); // Output: '{"name":"Asma","age":22}'**

* **✅ JSON.stringify() → Takes an object, gives you a JSON string.**