STEP ONE: Create application with TypeScript and install react-router-dom

```
npx create-react-app my-quiz-app --template typescript
cd my-quiz-app
npm install react-router-dom
```

STEP TWO: This will be the project structure — create any of these files that the scaffolding doesn't create

```
react-quiz-app/

├── public/

│   └── index.html

└── src/

    ├── App.tsx

    ├── App.scss

    ├── components/

    │   ├── Quiz.tsx

    │   └── Quiz.scss

    ├── types.ts

    ├── styles/

    │   ├── _variables.scss

    │   └── _mixins.scss

    ├── index.tsx

    └── index.css
```

STEP THREE: The types file

This file defines the TypeScript interfaces for your quiz application. Interfaces are a way to describe the shape of an object, specifying what properties it should have, and their types. Using interfaces in TypeScript helps with type safety. The TypeScript compiler will check if your code adheres to the defined types and warn you about potential errors. Typescript introduces interfaces, *which do not have to be used with classes.*

Code for `src/types.ts`:

```
export interface Question {
  questionText: string;
// The text of the question (a string)
  answerOptions: AnswerOption[];
// An array of AnswerOption objects
}

export interface AnswerOption {
  answerText: string;
// The text of the answer option (a string)
  isCorrect: boolean;
// Whether the answer option is correct (true or false)
}
```

- **`export interface Question`:** defines an interface called Question. It specifies that a Question object must have two properties:
    - `questionText`: A string representing the text of the question.
    - `answerOptions`: An array of `AnswerOption` objects, representing the possible answers to the question.
- **`export interface AnswerOption`:** defines an interface called `AnswerOption`. It specifies that an `AnswerOption` object must have two properties:
    - `answerText`: A string representing the text of the answer option.
    - `isCorrect`: A boolean value indicating whether the answer option is correct (true) or incorrect (false).
- **`export` keyword:** The export keyword makes these interfaces available to be imported and used in other files within your project (like in App.tsx and Quiz.tsx).

In your `App.tsx` file, you use the `Question` interface to define the type of the questions array:

```
const questions: Question[] = [
  // ... your questions here
];
```

This tells TypeScript that the questions array should contain objects that conform to the Question interface, meaning each object must have a questionText (string) and answerOptions (array of AnswerOption objects).

STEP FOUR: App.tsx:

1. Imports: Bring in necessary modules and components for your app to function correctly.

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link }
from 'react-router-dom';
import Quiz from './components/Quiz';
import  { Question } from './types';
import './App.scss';
```

- `react`: The core React library.
- `react-router-dom`: Provides components for routing (navigation).
- `Quiz`: Your component that handles the quiz logic and display.
- `Question`: The TypeScript interface defining the structure of a quiz question.
- `./App.scss`: Your SCSS file for styling the App component.

2. Quiz Questions Data

- Define the questions array to hold the quiz questions and their answer options. ***Note that they are all set to false; please set the correct one to true.***

```
const questions: Question[] = [
  {
    questionText: 'What is the virtual DOM?',
    answerOptions: [
      { answerText: 'An exact copy of the real DOM',
isCorrect: false },
      { answerText: 'A lightweight representation of the
real DOM', isCorrect: false },
      { answerText: 'A place where React components are
stored', isCorrect: false },
      { answerText: 'A way to style React components',
isCorrect: false },
    ],
  },
  {
    questionText: 'What is JSX?',
    answerOptions: [
      { answerText: 'A JavaScript library', isCorrect: 
false },
      { answerText: 'A templating language used to write 
HTML-like syntax in JavaScript', isCorrect: false  },
      { answerText: 'A CSS framework', isCorrect: false },
      { answerText: 'A type of React component', isCorrect: 
false },
    ],
  },

   {
    questionText: 'What is JSX?',
    answerOptions: [
      { answerText: 'A JavaScript library', isCorrect: 
false },
      { answerText: 'A templating language used to write 
HTML-like syntax in JavaScript', isCorrect: false },
      { answerText: 'A CSS framework', isCorrect: false },
      { answerText: 'A type of React component', isCorrect: 
false },
    ],
  },
  {
    questionText: 'What is a React component?',
    answerOptions: [
      { answerText: 'A function or class that returns JSX', 
isCorrect: false },
      { answerText: 'A plain JavaScript object', isCorrect: 
false },
      { answerText: 'An HTML element', isCorrect: false },
      { answerText: 'A CSS style rule', isCorrect: false },
    ],
  },
  {
    questionText: 'What is the purpose of the `key` prop 
when rendering lists?',
    answerOptions: [
      { answerText: 'To provide a unique identifier for 
each item in the list', isCorrect: false },
      { answerText: 'To style the list items', isCorrect: 
false },
      { answerText: 'To specify the order of the list
items', isCorrect: false },
      { answerText: 'To access the index of each item in
the list', isCorrect: false },
    ],
  },

  {
    questionText: 'What is the purpose of the `useState`
hook?',
    answerOptions: [
      { answerText: 'To fetch data from an API', isCorrect:
false },
      { answerText: 'To handle user input', isCorrect:
false },
      { answerText: 'To manage side effects', isCorrect:
false },
      { answerText: 'To add state to functional
components', isCorrect: false },
    ],
  },
  {
    questionText: 'What is the purpose of the `useEffect`
hook?',
    answerOptions: [
      { answerText: 'To perform side effects in functional
components', isCorrect: false },
      { answerText: 'To update state based on props',
isCorrect: false },
      { answerText: 'To create reusable logic', isCorrect:
false },
      { answerText: 'To render components conditionally',
isCorrect: false },
    ],
  },

  {
    questionText: 'What is React Context?',
    answerOptions: [
      { answerText: 'A way to pass data through the
component tree without prop drilling', isCorrect: false },
      { answerText: 'A tool for managing global state',
isCorrect: false },
      { answerText: 'A mechanism for handling events',
isCorrect: false },
      { answerText: 'A technique for optimizing
performance', isCorrect: false },
    ],
  },

  {
    questionText: 'What is the purpose of React
Fragments?',
    answerOptions: [
      { answerText: 'To group multiple elements without
adding extra nodes to the DOM', isCorrect: false },
      { answerText: 'To define reusable components',
isCorrect: false },
      { answerText: 'To handle events in React', isCorrect:
false },
      { answerText: 'To create animations', isCorrect:
false },
    ],
  },
  {
    questionText: 'What is the difference between
controlled and uncontrolled components?',
    answerOptions: [
      { answerText: 'Controlled components store their own
state, while uncontrolled components rely on the DOM',
isCorrect: false },
      { answerText: 'Controlled components are used for
forms, while uncontrolled components are used for
everything else', isCorrect: false },
      { answerText: 'Controlled components are more
efficient than uncontrolled components', isCorrect:
false },
      { answerText: 'Controlled components have more
features than uncontrolled components', isCorrect: false },
    ],
  },
  {
    questionText: 'What is React Router?',
    answerOptions: [
      { answerText: 'A library for handling navigation and
routing in React applications', isCorrect: false },
      { answerText: 'A tool for managing state in React',
isCorrect: false },
      { answerText: 'A way to fetch data from an API',
isCorrect: false },
      { answerText: 'A testing framework for React',
isCorrect: false },
    ],
  },
];
```

- Each question is an object with `questionText` and `answerOptions`.
- `answerOptions` is an array of objects, each with `answerText` and `isCorrect`.

3. App Component Structure

- Create the main App component, which will manage routing and rendering.

```
const App: React.FC = () => {
  return (
    <Router>
      <div className="app-container">
        {/* Navigation and Routes will go here */}
      </div>
    </Router>
  );
};
```

- `React.FC` indicates this is a functional component.
- `Router` enables routing within your app.

4. Navigation

- Add a navigation bar with links to your different pages (routes).
- Link components create clickable links that navigate to the specified routes.

```
// ... inside the app-container div ...
<nav className="navbar">
  <Link to="/" className="nav-link">Home</Link>
  <Link to="/quiz" className="nav-link">Quiz</Link>
</nav>
```

5. Routes

- Define the routes using Routes and Route components.
- `Routes` manages the rendering of different components based on the URL.
- `Route` defines a specific route (path) and the component to render.
- The questions array is passed as a prop to the `Quiz` component.

```
// ... inside the app-container div ...
<Routes>
  <Route path="/" element={<HomePage />} />
  <Route path="/quiz" element={<Quiz
questions={questions} />} />
</Routes>
```

6. HomePage Component

- Create the `HomePage` component to be displayed at the root route (/). This component simply welcomes the user and provides a link to start the quiz.

```
const HomePage: React.FC = () => {
  return (
    <div className="home-page">
      <h1>Welcome to the React Quiz!</h1>
      <p>Test your knowledge of React concepts.</p>
      <Link to="/quiz" className="start-button">Start
Quiz</Link>
    </div>
  );
};
```

7. Export

- Export the App component to make it usable in index.tsx.

```
export default App;
```

STEP FIVE: /components/Quiz.tsx

This component displays the quiz questions, handling user answers, and showing the final score.

```
import necessary modules:

import React, { useState } from 'react';

// Import React and the useState hook

import { Question } from '../types';

// Import the Question interface

import './Quiz.scss';
// Import the SCSS file for styling
```

- `useState`: This hook is used to manage the component's state (current question, score, and whether to show the score).
- `Question`: This interface defines the structure of a quiz question, ensuring type safety.
- `./Quiz.scss`: This imports the SCSS file for styling the component.

5. **Define the QuizProps interface:**

```
interface QuizProps {

questions: Question[];

// Define questions prop as an array of Question objects


}
```

- This interface defines the props that the `Quiz` component expects to receive. In this case, it expects a questions prop, which is an array of `Question` objects.

Create the Quiz component:

```
const Quiz: React.FC<QuizProps> = ({ questions }) => {

  // ... component logic and JSX ...
};
```

- `React.FC<QuizProps>`: This indicates that `Quiz` is a functional component that accepts props of type `QuizProps`.

- `({ questions })`: This destructures the questions prop from the props object for easier access within the component.

Initialize state variables:

```
const [currentQuestion, setCurrentQuestion] =
useState(0); // Initialize current question to 0

const [showScore, setShowScore] = useState(false);

// Initialize showScore to false

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

// Initialize score to 0
```

- `useState(0)`: This initializes the currentQuestion state variable to 0, indicating that the quiz starts at the first question.
- `useState(false)`: This initializes the showScore state variable to false, indicating that the score should not be shown initially.
- `useState(0)`: This initializes the score state variable to 0.

Define the handleAnswerOptionClick function:

```
const handleAnswerOptionClick = (isCorrect: boolean) => {

  if (isCorrect) {
    setScore(score + 1); // Increment score if the answer
is correct
  }

  const nextQuestion = currentQuestion + 1;
  if (nextQuestion < questions.length) {
    setCurrentQuestion(nextQuestion);

 // Move to the next question
  } else {
    setShowScore(true); // Show the score if all questions
are answered
  }
};
```

- This function is called when the user clicks on an answer option.

  - It takes a boolean argument (`isCorrect`) indicating whether the chosen answer is correct.
  - If the answer is correct, it increments the score.
  - It then checks if there are more questions. If yes, it moves to the next question by updating currentQuestion. If no, it sets showScore to true to display the final score.

Return the JSX:

```
return (

  <div className="quiz-container">
    {showScore ? ( // Conditionally render the score or the 
questions
      <div className="score-section">
        You scored {score} out of {questions.length}
      </div>
    ) : (
      <>
        <div className="question-section">
          <div className="question-count">
            <span>Question {currentQuestion + 1}</span>/
{questions.length}
          </div>

          <div className="question-
text">{questions[currentQuestion].questionText}</div>
        </div>
        <div className="answer-section">

{questions[currentQuestion].answerOptions.map((answerOption
)  => (
            <button key={answerOption.answerText} 
onClick={() => 
handleAnswerOptionClick(answerOption.isCorrect)}>
              {answerOption.answerText}
            </button>
          ))}
        </div>
      </>
    )}
  </div>
);
```

- This code conditionally renders either the score section or the question section based on the value of `showScore`.
- If `showScore` is true, it displays the final score.
- If `showScore` is false, it displays the current question and its answer options.
- The `map` function is used to iterate over the `answerOptions` and render a button for each option.

56. **Export the Quiz component:**

```
export default Quiz;
```

STEP SIX: All SCSS files.

React can read SCSS. Change all .css extensions to .scss except for index.css.

Reference for SASS/SCSS details: https://sass-lang.com/guide/

- The SCSS files store all the SCSS variables for your project, so you can define values (like colors, font sizes, etc.) in one place and reuse them throughout your stylesheets. This maintains consistency throughout global styles.
- &:hover is like `:hover` but the context is:
  - **CSS Preprocessors (like Sass)**
  - **Parent Selector:** The ampersand (`&`) represents the parent selector, making it clear that the hover style is directly linked to the element being defined.
  - **Nesting:** It's often used within nested selectors to apply the hover effect specifically to the parent element.

SCSS Code: /styles/_variables.scss

```
$primary-color: #2196F3;  // Blue shade for primary
elements (buttons)

$secondary-color: #1976D2;
```

```
// Slightly darker blue for hover effects
$background-color: #f0f0f0;
// Light gray background color
$text-color: #333;
// Dark gray text color
$font-size-base: 16px;
// Base font size for the application
$border-radius-base: 5px;
// Standard border-radius for buttons, etc.
```

SCSS Code: _mixins.scss

- This file defines SCSS mixins. Mixins are reusable blocks of styles that can be included in other stylesheets. They help you avoid code duplication and keep your styles organized. Note the underscore character.

SCSS Code for /styles/_mixins.scss:

```
@mixin button-style {

  padding: 10px 20px;
// Standard padding for buttons
  border: none;
  border-radius: $border-radius-base;
// Use the variable for border-radius
  color: white;
  font-size: $font-size-base;
// Use the variable for font size
  cursor: pointer;
// Show pointer cursor on hover
  transition: background-color 0.3s ease;
// Smooth transition for background color on hover

  &:hover {
    background-color: $secondary-color;
// Use the secondary color on hover
  }
}
```

SCSS Code: /components/Quiz.scss

- This file styles the Quiz component, which displays the quiz questions and answers.

```
@import '../styles/variables'; // Import the variables file

@import '../styles/mixins';
// Import the mixins file

.quiz-container {
  display: flex;
  flex-direction: column;
// Arrange elements vertically
  align-items: center;
// Center horizontally
  justify-content: center;
// Center vertically
  height: 100vh;
// Take full viewport height
  background-color: $background-color;
// Use the variable for background color
  color: $text-color;
// Use the variable for text color
  font-family: sans-serif;
  .question-section {
    text-align: center;
// Center the question text
    margin-bottom: 20px;
// Add space below the question
    .question-count {
      font-size: $font-size-base + 2;
// Slightly larger font size for question count
      margin-bottom: 10px;
    }
    .question-text {
      font-size: $font-size-base + 8;
// Larger font size for question text
      font-weight: bold;

    }
  }

  .answer-section {
    display: flex;
    flex-direction: column;
// Arrange answer buttons vertically
    width: 80%;
// Set width of the answer section

    button {
      @include button-style;
// Use the button-style mixin
      margin-bottom: 10px;

      background-color: $primary-color;
// Use the primary color for buttons
    }
  }

  .score-section {
    font-size: $font-size-base + 8;
// Larger font size for the score
    font-weight: bold;

  }
}
```

SCSS Code: App.scss

- This file styles the main App component, including the navigation bar and the home page.

SCSS Code for /App.scss:

```
@import './styles/variables';

// Import the variables file

@import './styles/mixins';
// Import the mixins file
```

```
.app-container {
  text-align: center;
// Center text within the app container

  .navbar {
    background-color: #333;
    padding: 10px;
    .nav-link {
      color: white;
      margin: 0 15px;
      text-decoration: none;

    }
  }
  .home-page {
    display: flex;
    flex-direction: column;
// Arrange elements vertically
    align-items: center;
// Center horizontally
    justify-content: center;
// Center vertically
    height: 100vh;
// Take full viewport height
    font-family: sans-serif;
    h1 {
      font-size: 3rem;
// Large font size for the heading
      margin-bottom: 1rem;
// Add space below the heading
      color: $text-color;
// Use the variable for text color
    }
    .start-button {
      @include button-style;
// Use the button-style mixin
      display: inline-block;
// Display button as inline-block
      background-color: $primary-color;
// Use the primary color for the button
      text-decoration: none;
    }
```

```
        }
    }
```