**Hint:**

- **Create a file called questions.json with quiz questions and answers, and put it in the /public directory. Install json-server and start this before running the app. You can use your questions.json file and add the answer to each one. The first two already have an answer. Note that there are only 19 total; you can make up your own question, choices, and answer for the last one.**
- **Create files called Home.jsx, QuizPage.jsx and ResultsPage.jsx and put them in a directory you create called /pages.**

### `Home` Component Code

**Imports**:
`useNavigate` is a hook from `react-router-dom`, which we will use to programmatically navigate between routes (pages) in your application.

```
import { useNavigate } from “react-router-dom";
```

**Create a Home component that uses it and returns a variable called `navigate` to redirect users to different pages:**

```
const Home = () => {
  const navigate = useNavigate();
}
```

**Put quiz instructions inside return - header, telling people to answer questions, etc., including a button implementing the `navigate`. Navigate to the path `/quiz`.**

```
return (


<button onClick={() => navigate(“/quiz")}>Start Quiz</
button>


);
```

**Set up routes using `react-router-dom` in your to navigate to `/quiz`. You can do this in App.jsx. Set up routing so that:**

- The app starts at the Home page ("/").
- Clicking "Start Quiz" navigates to the `/quiz` route, where your quiz will be displayed.

## `QuizPage` Component Code Breakdown

1. **Import `useNavigate`**
2. In**tialize states**:
    - **`questions`**: Stores the list of quiz questions.
    - **`currentQuestionIndex`**: Tracks which question is currently being displayed.
    - **`selectedAnswer`**: Holds the answer selected by the user for the current question.
    - **`score`**: Keeps track of the user's score (number of correct answers).
    - **`timeLeft`**: Holds the countdown timer (default is set to 7 seconds per question).

**Hint:**

```
const [questions, setQuestions] = useState([]);

const [currentQuestionIndex, setCurrentQuestionIndex] =
useState(0);
const [selectedAnswer, setSelectedAnswer] = useState(null);
const [score, setScore] = useState(0);
const [timeLeft, setTimeLeft] = useState(7); // Timer (7s
per question)
const navigate = useNavigate();
```

**Optional: add a function to shuffle question order:**

```
const shuffleArray = (array) => {

  return array.sort(() => Math.random() - 0.5);

};
```

**Load questions with `useEffect`:**

- This `useEffect` runs once when the component mounts (`[]` means it runs only once).
- It fetches the quiz questions from a JSON file (`/questions.json`), and then shuffles the questions using the `shuffleArray` function before storing them in the `questions` state.
- Optional: if there's an error (e.g., the file can't be found), it's logged to the console.

Fetch can look like:

```
  fetch("/questions.json")
    .then((res) => {
```

shuffleArray is used here. If you did not use it, leave off shuffleArray and just get data.

```
    .then((data) => setQuestions(shuffleArray(data)))
```

**Optional: Timer with `useEffect`:**

- This `useEffect` is responsible for the countdown timer.
- It checks if the timer (`timeLeft`) has reached 0. If so, it automatically moves to the next question by calling `nextQuestion()`.
- Otherwise, it sets a timeout to decrement the `timeLeft` state by 1 every second.

**Hint:**

```
    useEffect(() => {
     if (timeLeft === 0) {
      nextQuestion();
      return;
    }
    const timer = setTimeout(() => setTimeLeft(timeLeft -
1), 1000);
    return () => clearTimeout(timer);
}, [timeLeft]);
```

- 

**Loading State**:

If the questions are still being fetched, a loading message will be displayed.

**Hint:**

```
if (!questions.length) return <p>Loading...</p>;
```

**Handle answers**:

This function sets the selected answer when the user clicks on an answer option.

**Hint:**

```
const handleAnswer = (answer) => {

  setSelectedAnswer(answer);
};
```

**Navigate to the next question**:

- This function checks if the selected answer is correct. If correct, the score is incremented.
- After the user selects an answer, it resets the selected answer and the timer for the next question.
- It moves to the next question if there are more questions; otherwise, it navigates to the `/results` page and passes the `score` and `total` questions as state.

**Hint:**

```
const nextQuestion = () => {

  if (selectedAnswer ===
questions[currentQuestionIndex].answer) {
    setScore(score + 1);
  }
  setSelectedAnswer(null);
  setTimeLeft(7); // Reset timer

  if (currentQuestionIndex + 1 < questions.length) {
    setCurrentQuestionIndex(currentQuestionIndex + 1);
  } else {
```

```
    navigate("/results", { state: { score, total:
questions.length } });
  }
};
```

**Optional progress bar:**

Calculates the progress percentage based on the current question index and the total number of questions.

```
const progress = ((currentQuestionIndex + 1) /
questions.length) * 100;
```

**Render the UI**:

**Hint:**

```
return (

  <div style={{ padding: "20px", maxWidth: "500px", margin:
"auto" }}>
    {/* Progress Bar */}
    <div style={{ width: "100%", backgroundColor: "#ccc",
height: "10px", marginBottom: "10px" }}>
      <div style={{ width: `${progress}%`, backgroundColor:
"#4caf50", height: "10px" }}></div>
    </div>

    {/* Timer */}
    <h3>Time Left: {timeLeft}s</h3>

    <h2>Question {currentQuestionIndex + 1}</h2>
    <p>{questions[currentQuestionIndex].question}</p>

    {questions[currentQuestionIndex].options.map((option)
=> (
      <button
        key={option}
        onClick={() => handleAnswer(option)}
```

```
        style={{
          backgroundColor: selectedAnswer === option ?
"gray" : "",
          display: "block",
          width: "100%",
          margin: "5px 0",
          padding: "10px",
        }}
      >
        {option}
      </button>
    ))}

    <br />
    <button onClick={nextQuestion} disabled={!
selectedAnswer} style={{ marginTop: "10px" }}>
      Next
    </button>
  </div>
);


};
```

## `ResultsPage` Component

1. After completing the quiz, the user is redirected to this `ResultsPage` with the quiz score and total questions passed as state from the quiz page.
2. The user is shown their score on the `ResultsPage`.
3. They can click the "Try Again" button to go back to the home page and start the quiz over.

1. **Imports**:
    - `useLocation` is a hook from `react-router-dom` that allows you to access the current location (e.g., passed state from the previous page).
    - `useNavigate` is another hook from `react-router-dom` used to navigate to different pages programmatically.

    -

**Setup state**:

- `location`: This hook retrieves the current location object. It's used to access state passed between pages (in this case, the `score` and `total`).
- `navigate`: This hook allows you to programmatically navigate to different routes/pages.
- The `score` and `total` are destructured from the `location.state`. If `state` is not available (i.e., the user comes directly to this page without a score or total), it defaults to `{ score: 0, total: 0 }`.

**Hint:**

```
const location = useLocation();
const navigate = useNavigate();
const { score, total } = location.state || { score: 0, total: 0 };
```

**Rendering the Results**:

- The user will see a message stating "Quiz Completed!".
- It displays the score the user achieved and the total number of questions in the quiz.
- The "Try Again" button triggers navigation back to the home page (`/`) when clicked. The `navigate("/")` function is used to go back to the home page.

**Hint:**

```
return (
  <div>
    <h1>Quiz Completed!</h1>
    <p>Your Score: {score} / {total}</p>
    <button onClick={() => navigate("/")}>Try Again</button>
  </div>
);
```