

Fun React Quiz App→ A React quiz application related to trivia knowledge which generates random questions, developed using React and TypeScript along with Styled-Components scaffolding components using React and TypeScript documentation.

Functionalities/Demo:

[fun react quiz app overview-1.PNG](#)

[fun react quiz app overview-2.PNG](#)

- Each quiz question will display 4 choices for users to choose.
- The correct/incorrect answer will be revealed after the user clicks on one choice from four.
- The quiz will move on to next question after the user is done answering the current one.

Developing Tools:

Create React App adding TypeScript Docs <https://create-react-app.dev/docs/adding-typescript/>

Vscode 1.72 https://code.visualstudio.com/updates/v1_72

Styled-Components Dependency for React <https://styled-components.com/>

Unsplash Image Gallery https://unsplash.com/photos/3FA80_d8rHo

Google Fonts-Catamaran Bold700 <https://fonts.google.com/specimen/Catamaran?query=catam>

Trivia API https://opentdb.com/api_config.php

vscode-styled-components extension v1.7.5

<https://marketplace.visualstudio.com/items?itemName=styled-components.vscode-styled-components>

Prerequisites & Setup:

Install all TypeScript documentations within Create React App in local Console:

```
npx create-react-app fun-react-quiz-app --template typescript
```

Navigate into the Fun React Quiz App folder:

```
cd fun-react-quiz-app
```

Open the project with Vscode and remove 6 unnecessary files:

```
setupTests.ts serviceWorker.ts logo.svg index.css App.test.tsx App.css
```

Install the Styled-Components Dependency for React in Console:

```
npm i styled-components @types/styled-components
```

Installed the style-components library first, then installed the styled-component itself.

Start localhost web server with:

```
npm start
```

Import Catamaran font into index.html:

```
<link
href="https://fonts.googleapis.com/css2?family=Catamaran:wght@700&display=swap" rel="stylesheet">
```

Generate a Multiple Choice API from Trivia API to get JSON response.

Synchronous Developing Notes:

Implement Logics:

Create `API.ts` to create logic for fetching data from API.

Create `utils.ts` to randomize the answers to the quiz questions.

Implement core components in `App.tsx`:

```
<div className="App">
  <h1>Fun React Quiz</h1>
```

```

    <button className = "start" onClick = {startTrivia}>
      Start
    </button>
    <p className = "score">Score:</p>
    <p>Loading Questions...</p>

```

Button to keep the next question in QuestionCard:

```

    <button className='next' onClick={nextQuestion}>
      NextQuestion
    </button>

```

Create QuestionCards components:

In QuestionCard.tsx, create props for question cards components:

```

type Props = {
  question: string;
  answers: string[];
  callback: any;
  userAnswer: any;
  questionNr: number;
  totalQuestions: number; }

```

Create different use states in App.tsx:

```

const App = () => {
  const [loading, setLoading] = useState(false);
  const [questions, setQuestions] = useState([]);
  const [number, setNumber] = useState(0);
  const [userAnswers, setUserAnswers] = useState([]);
  const [score, setScore] = useState(0);
  const [gameOver, setGameOver] = useState(true);

```

Create the function that grabs the data from API in API.ts:

```

export const fetchQuizQuestions = async (amount: number, difficulty:
Difficulty) => {
  const endpoint = `...`
  const data = await (await fetch(endpoint)).json();
  console.log(data);

```

Now the initial page of React app in localhost server looks like:

quiz app initial page.PNG

Specify the type in App.ts:

```

export type Question = {
  category: string;
  correct_answer: string;
  difficulty: string;
  incorrect_answer: string[];
  question: string;
  type: string; }

```

Shuffle arrays and functions to export in `utils.ts`:

```
export const shuffleArray = (array: any[]) =>
  [...array].sort(() => Math.random() - 0.5);
```

Now reload the web server we have Promises showing up on the JS console inspect:

Promise shows up in Console Inspecting.PNG

Implement the `startTrivia` function to start the game in `App.tsx`:

```
const startTrivia = async () => {
  setLoading(true);
  setGameOver(false);
  const newQuestions = await fetchQuizQuestions(
    TOTAL_QUESTIONS,
    Difficulty.EASY
  );
  setQuestions(newQuestions);
  setScore(0);
  setUserAnswers([]);
  setNumber(0);
  setLoading(false);
};
```

Game over if finished 10 questions in `App.tsx`:

```
{gameOver || userAnswers.length === TOTAL_QUESTIONS ? (
  <button className="start" onClick={startTrivia}>
    Start
  </button>
) : null}
{!gameOver ? <p className="score">Score:</p> : null}
```

Now we can see that the quiz questions loading is showing:

loading is showing.PNG

If it's not loading and not game over, show the question card:

```
{!gameOver ? <p className="score">Score:</p> : null}
{loading && <p>Loading Questions...</p>}
{!loading && !gameOver && (
  <QuestionCard
    questionNr={number + 1}
    totalQuestions={TOTAL_QUESTIONS}
    question={questions[number].question}
    answers={questions[number].answers}
    userAnswer={userAnswers ? userAnswers[number] : undefined}
    callback={checkAnswer}
  /> )}
<button className='next' onClick={nextQuestion}>
  NextQuestion
```

```

        </button>
      </div>
    );}

```

Now random questions are generated:

random questions are generated.PNG

Implement the CheckAnswer function:

In App.tsx:

```

const checkAnswer = (e: React.MouseEvent<HTMLButtonElement>) => {
  if (!gameOver) {
    //users answer
    const answer = e.currentTarget.value;
    //check answer against correct answer
    const correct = questions[number].correct_answer === answer;
    //add score if answer is correct
    if (correct) setScore((prev) => prev + 1);
    //save answer in the array for user answers
    const answerObject = {
      question: questions[number].question,
      answer,
      correct,
      correctAnswer: questions[number].correct_answer,
    };
    setUserAnswers((prev) => [...prev, answerObject]);
  }
};

```

Now clicking on start, the answer is stored and the next question option is showing:

answer stored and next question shows.PNG

Implement the NextQuestion function:

In App.tsx:

```

const nextQuestion = () => {
  //move on to the next question if not the last question
  const nextQuestion = number + 1;

  if (nextQuestion === TOTAL_QUESTIONS) {
    setGameOver(true);
  } else {
    setNumber(nextQuestion);
  }
}

```

Now the next question answers are stored when completed all questions:

answers are stored.PNG

Styling:

Error: Score not showing while user answering the questions. DEBUGGING: In App.tsx:
`{!gameOver ? <p className="score">Score: {score}</p> : null}` Now the score shows: **score shows.PNG**

To make the background image display:

Create a new file named App.styles.ts in source, import background image:

```
import styled, { createGlobalStyle } from 'styled-components';  
import BGImage from './images/mypic.jpg';  
export const GlobalStyle = createGlobalStyle`
```

Import into App.tsx:

```
import { GlobalStyle, Wrapper } from './App.styles';
```

background image is displayed.PNG

Create a new file App.styles.ts and create some styles:

```
font-family: Fascinate Inline;  
background-image: linear-gradient(180deg, #fff, #87f1ff);  
font-weight: 400;  
background-size: 100%;  
background-clip: text;  
-webkit-background-clip: text;  
-webkit-text-fill-color: transparent;  
-moz-background-clip: text;  
-moz-text-fill-color: transparent;  
filter: drop-shadow(2px 2px #0085a3);  
font-size: 70px;  
text-align: center;  
margin: 20px;
```