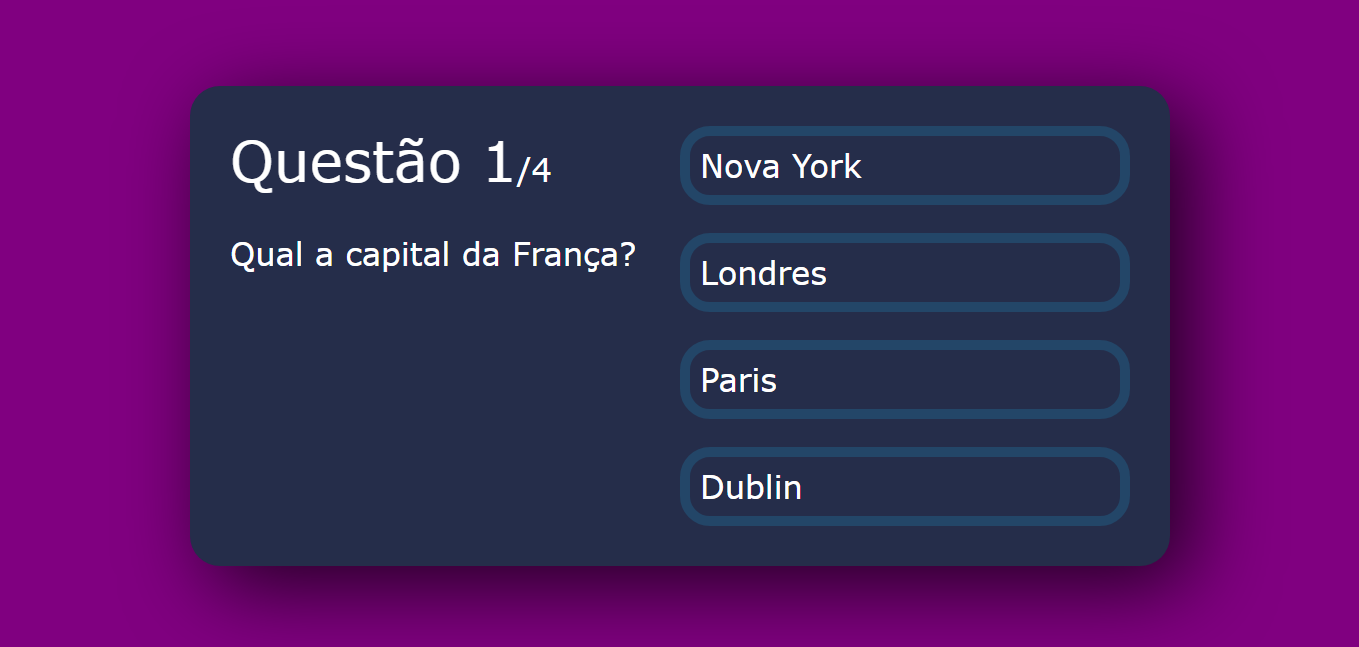
**Codi Ao Vivo: React Quiz**

Parte 1: Introdução

Neste tutorial React para iniciantes, vamos construir um aplicativo de quiz. Trabalharemos com objetos de estado complexos, como lidar com diferentes ganchos de estado e renderizar coisas com base no estado.



O aplicativo terá as seguintes funcionalidades:

* Quando o usuário clica em um botão, a próxima pergunta deve mostrar
* Se o usuário acertar a pergunta, ele deve incrementar sua pontuação
* Quando o usuário chega ao final do questionário, sua pontuação total deve ser mostrada

Comece criando um novo aplicativo react no Visual Studio:

**npm init react-app quiz**

Crie 3 arquivos:

* index.js
* index.css
* App.js

No arquivo index.css copie o código:

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

\* {

font-family: "Verdana", cursive, sans-serif;

color: #ffffff;

}

body {

background-color: purple;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

}

.app {

background-color: #252d4a;

width: 450px;

min-height: 200px;

height: min-content;

border-radius: 15px;

padding: 20px;

box-shadow: 10px 10px 42px 0px rgba(0, 0, 0, 0.75);

display: flex;

justify-content: space-evenly;

}

.score-section {

display: flex;

font-size: 24px;

align-items: center;

}

/\* QUESTION/TIMER/LEFT SECTION \*/

.question-section {

width: 100%;

position: relative;

}

.question-count {

margin-bottom: 20px;

}

.question-count span {

font-size: 28px;

}

.question-text {

margin-bottom: 12px;

}

.timer-text {

background: rgb(230, 153, 12);

padding: 15px;

margin-top: 20px;

margin-right: 20px;

border: 5px solid rgb(255, 189, 67);

border-radius: 15px;

text-align: center;

}

/\* ANSWERS/RIGHT SECTION \*/

.answer-section {

width: 100%;

display: flex;

flex-direction: column;

justify-content: space-between;

}

button {

width: 100%;

font-size: 16px;

color: #ffffff;

background-color: #252d4a;

border-radius: 15px;

display: flex;

padding: 5px;

justify-content: flex-start;

align-items: center;

border: 5px solid #234668;

cursor: pointer;

}

.correct {

background-color: #2f922f;

}

.incorrect {

background-color: #ff3333;

}

button:hover {

background-color: #555e7d;

}

button:focus {

outline: none;

}

button svg {

margin-right: 5px;

}

No arquivo index.js vamos definir a renderização do aplicativo e importar os demais arquivos:

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

Parte 2: App.js

Agora vamos criar a lógica do aplicativo no Componentes App.js, que mostra as questões e dá o resultado.

Não copie pedaços de códigos para o App.js, copie o código base inicial ao fim desse documento, e altere-o a partir da Parte 3 desse roteiro.

Começamos importando o useState:

import React, { useState } from 'react';

Na sequência iniciamos a função do componente, de agora em diante todos os códigos estarão dentro dessa função.

export default function App() {}

Dentro da função, começamos criando um array com as perguntas e as alternativas, repare que cada pergunta é um objeto com o atributo “título” e tem um array de objetos interno com as alternativas, onde cada objeto tem um título e a propriedade “isCorrect”, a questão correta recebe true e as demais, false.

const questions = [

{

questionText: 'What is the capital of France?',

answerOptions: [

{ answerText: 'New York', isCorrect: false },

{ answerText: 'London', isCorrect: false },

{ answerText: 'Paris', isCorrect: true },

{ answerText: 'Dublin', isCorrect: false },

],

},

{

questionText: 'Who is CEO of Tesla?',

answerOptions: [

{ answerText: 'Jeff Bezos', isCorrect: false },

{ answerText: 'Elon Musk', isCorrect: true },

{ answerText: 'Bill Gates', isCorrect: false },

{ answerText: 'Tony Stark', isCorrect: false },

],

},

{

questionText: 'The iPhone was created by which company?',

answerOptions: [

{ answerText: 'Apple', isCorrect: true },

{ answerText: 'Intel', isCorrect: false },

{ answerText: 'Amazon', isCorrect: false },

{ answerText: 'Microsoft', isCorrect: false },

],

},

{

questionText: 'How many Harry Potter books are there?',

answerOptions: [

{ answerText: '1', isCorrect: false },

{ answerText: '4', isCorrect: false },

{ answerText: '6', isCorrect: false },

{ answerText: '7', isCorrect: true },

],

},

];

Para o retorno da função, utilize o trecho:

Repare que existem duas coisas que podem ser exibidas:

* O placar;
* As perguntas;

E o trecho seguinte organiza esse fluxo:

{false ? (

<div className='score-section'>Você acertou 1 de {questions.length} questões</div> ) : (

Com o segundo fluxo, começamos declarando a pergunta:

( // fluxo 2 "jogando"

<>

<div className='question-section'>

<div className='question-count'>

<span>Questão 1</span>/{questions.length}

</div>

<div className='question-text'>É aqui que a pergunta vai estar</div>

</div>

Em seguida, mostramos as alternativas e encerramos a função:

<div className='answer-section'>

<button>Alternativa 1</button>

<button>Alternativa 2</button>

<button>Alternativa 3</button>

<button>Alternativa 4</button>

</div>

</>

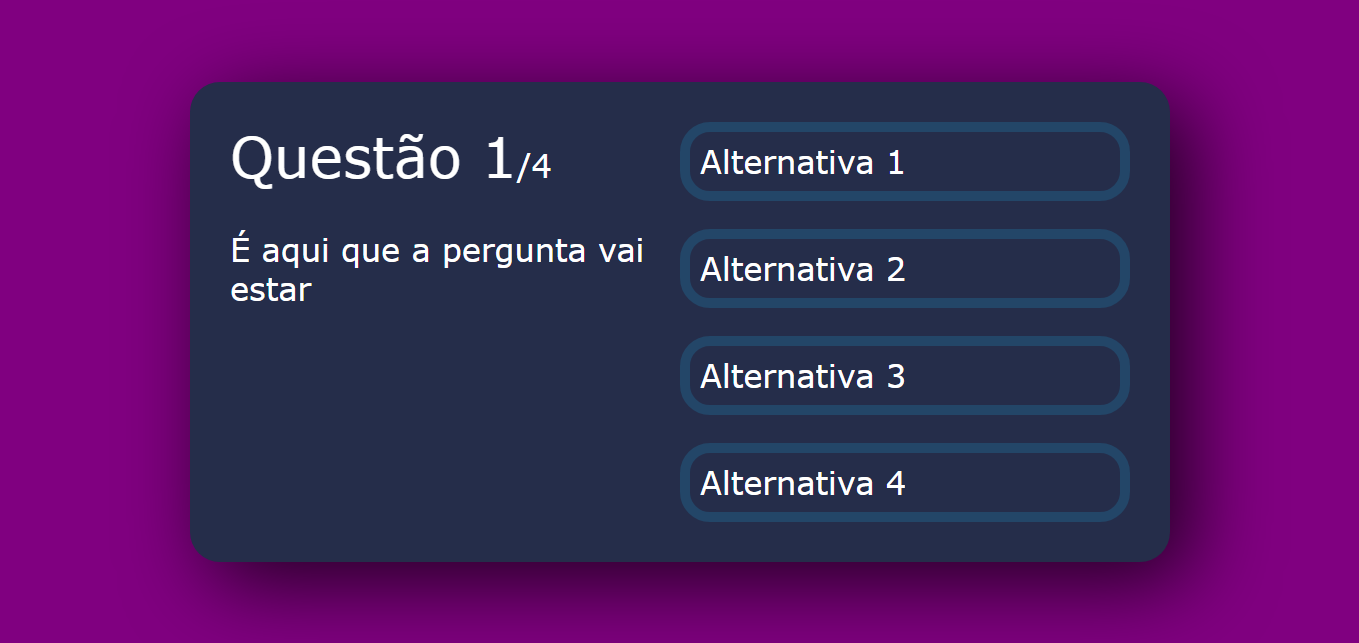
)}

</div>

);

}

Teremos então o seguinte resultado estático:



Parte 3: Perguntas e Respostas

Nosso primeiro objetivo é pegar os dados da pergunta do array e exibi-los na tela. Vamos remover o texto codificado e pegar os dados da primeira pergunta por enquanto, apenas para fazer as coisas acontecerem. Vamos nos preocupar com a troca de perguntas mais tarde.

Em nosso JSX, remova o texto “É aqui que a pergunta vai estar” e digite {questions[0]}para obter o primeiro item (ou pergunta) em nossa matriz de perguntas.

A primeira questão é um objeto, então podemos usar “notação de ponto” para ter acesso às propriedades, faremos apenas {question[0].questionText} para ter acesso ao texto da pergunta para este objeto:

<div className='question-text'>{questions[0].questionText}</div>

Salve e execute o aplicativo.

Observe como o texto é atualizado. Lembre-se de que estamos apenas pegando o primeiro texto da pergunta do primeiro objeto em nosso array de perguntas. Usaremos uma abordagem semelhante para as opções de resposta.

Remova os botões com as alternativas estáticas e usaremos a função map para percorrer as opções de resposta para uma determinada pergunta.

Lembre-se de que a função map faz um loop sobre o array e nos dá o item atual em que o loop está atualmente, na forma de uma variável.

Substitua a div "answer-section" pelo seguinte:

<div className='answer-section'>

{questions[0].answerOptions.map((answerOption, index) => (

<button>{answerOption.answerText}</button>))}

</div>

Salve e execute o aplicativo.

Observe como quatro botões de resposta aparecem e o texto é renderizado dinamicamente.

Vamos recapitular:

* Estamos recebendo a primeira pergunta da matriz de perguntas:questions[0]
* A primeira pergunta é um objeto, que contém uma matriz de answerOptions.
* Podemos chegar a esse array usando a notação de ponto: questions[0].answerOptions
* Como o answerOptionsé um array, podemos mapear sobre isso: questions[0].answerOptions.map
* Dentro da função map, renderizamos um botão para cada answerOption e exibimos o texto

Agora vamos voltar ao nosso JSX. Observe como, se mudarmos questions[0]para questions[1], ou questions, a interface do usuário será atualizada. Isso ocorre porque está obtendo os dados de diferentes perguntas em nossa matriz de perguntas, dependendo do índice.

O que queremos fazer é usar um objeto de estado para manter em qual pergunta o usuário está atualmente e atualizar isso quando um botão de resposta for clicado.

Vá em frente e adicione um objeto de estado após a criação do array em App, que conterá o número da pergunta atual em que o usuário está. Isso será inicializado com 0 para que o questionário receba a primeira pergunta da matriz:

const [currentQuestion, setCurrentQuestion] = useState(0);

Agora queremos substituir o '0' codificado em nosso JSX por esta variável. Primeiro para o texto da pergunta:

<div className='question-text'>{questions[**currentQuestion**].questionText}

</div>

E também para a seção de perguntas:

<div className='answer-section'>

{questions[**currentQuestion**].answerOptions.map((answerOption, index) => (

<button>{answerOption.answerText}</button>

))}

</div>

Agora, se você inicializar a currentQuestion com algo diferente de 0, por exemplo 1 ou 2, a interface do usuário será atualizada para mostrar a pergunta e as respostas para essa pergunta específica.

Vamos adicionar um código para que, quando clicarmos em uma resposta, incrementamos o valor currentQuestion para nos levar para a próxima pergunta.

Crie uma nova função chamada handleAnswerButtonClick (após a criação do objeto de estado) . Isso é o que será chamado quando o usuário clicar em uma resposta. Vamos incrementar o valor da pergunta atual em um, salvá-lo em uma nova variável e definir essa nova variável no estado:

const handleAnswerButtonClick = (answerOption) => {

const nextQuestion = currentQuestion + 1;

setCurrentQuestion(nextQuestion);

};

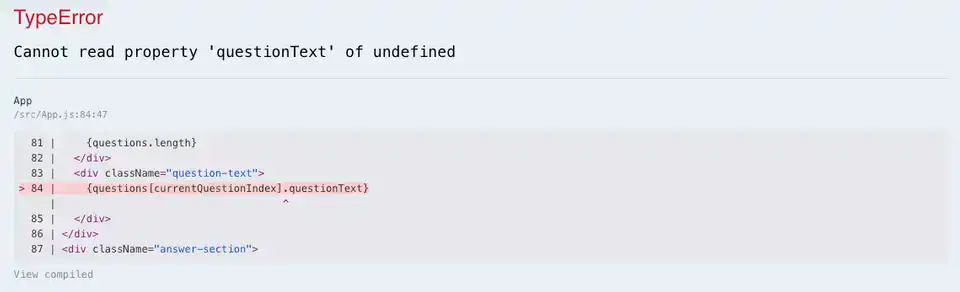
Em seguida, adicione um evento onClick ao nosso botão assim:

<button **onClick={() => handleAnswerButtonClick()}**>

{answerOption.answerText}

</button>

Se tentarmos isso, você verá que funciona, até chegarmos ao final:



Então o que está acontecendo? Bem, em nossa função handleAnswerButtonClick , estamos incrementando o número e definindo-o como estado. Isso está ok.

Mas lembre-se que usamos este número para acessar um array, a fim de obter as opções de pergunta e resposta. Quando chegarmos a 5, ele quebrará, pois não há 5º elemento!.

Vamos fazer uma verificação para garantir que não ultrapassem o limite. Em nossa função handleAnswerButtonClick vamos adicionar a seguinte condição:

const handleAnswerButtonClick = (answerOption) => {

const nextQuestion = currentQuestion + 1;

**if (nextQuestion < questions.length) {**

**setCurrentQuestion(nextQuestion);**

**} else {**

**alert('you reached the end of the quiz');**

**}**

};

Isso basicamente diz que se o número da próxima pergunta for menor que o número total de perguntas, atualize o estado para a próxima pergunta. Caso contrário, chegamos ao final do teste, então mostre um alerta por enquanto.

Parte 4: Tela de Pontuação

Em vez de mostrar um alerta, o que queremos fazer é mostrar a tela de “pontuação”.

Se olharmos para o JSX, você notará que eu coloquei a marcação aqui para você, só precisamos substituir “false” pela lógica.

Então, como vamos fazer isso? Bem, isso é a coisa perfeita para colocar no estado!

Adicione outro objeto de estado que armazenará se queremos mostrar a tela de pontuação ou não:

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

E substitua false por showScore em nosso JSX:

{**showScore** ? ( // fluxo 1 "fim de jogo"

<div className='score-section'>Você acertou 1 de {questions.length} questões</div>

)

Nada mudará, mas se alterarmos o valor do estado para true, a div da pontuação será exibida. Isso ocorre porque tudo está envolto em um ternário, ou seja:

“Se showScore for true, renderize a marcação da seção de pontuação, caso contrário, renderize a marcação de pergunta/resposta do questionário”

Agora, queremos atualizar essa variável de estado quando o usuário chegar ao final do questionário. Já escrevemos a lógica para isso em nossa função handleAnswerButtonClick.

Tudo o que precisamos fazer é substituir a lógica de alerta que atualiza a variável showScore para ser verdadeira:

const handleAnswerButtonClick = (answerOption) => {

const nextQuestion = currentQuestion + 1;

setCurrentQuestion(nextQuestion);

if (nextQuestion < questions.length) {

setCurrentQuestion(nextQuestion);

} else {

**setShowScore(true);**

}

};

Se clicarmos nas respostas do questionário, ele mostrará a seção de pontuação quando chegarmos ao final. No momento, o texto e a pontuação mostrados são uma string codificada, portanto, devemos torná-lo dinâmico.

Também vamos atualizar a contagem de questões na tela:

<span>Question {currentQuestion + 1}</span>/{questions.length}

Parte 5: Salvando a pontuação

Nossa próxima tarefa é manter uma pontuação em algum lugar em nosso aplicativo e incrementar esse valor se o usuário selecionar a opção correta.

O lugar lógico para fazer isso é dentro da função “handleAnswerButtonClick”.

Lembre-se que quando iteramos sobre o answerOptions , a função map nos dá um objeto para cada um que inclui o questionText e um valor booleano mostrando se essa resposta está correta ou não. Este booleano é o que usaremos para nos ajudar a incrementar nossa pontuação.

Em nosso botão, atualize a função assim:

<button

**onClick={()=> handleAnswerButtonClick(answerOption.isCorrect)}>**

{answerOption.answerText}</button>

Em seguida, atualize a função para aceitar este parâmetro:

const handleAnswerButtonClick = (**isCorrect**) => {}

Agora podemos adicionar alguma lógica aqui em nossa função. Por enquanto queremos dizer “se isCorrect for true, queremos mostrar um alerta”:

const handleAnswerButtonClick = (isCorrect) => {

if (isCorrect) {

alert("the answer is correct!")

}

Este é o mesmo que if(isCorrect === true), apenas uma versão abreviada.

Agora, se tentarmos isso, você verá que recebemos um alerta quando clicamos na resposta correta.

Pra recapitular até agora:

* Quando iteramos sobre os botões, passamos o valor isCorrect booleano desse botão para a função handleAnswerButtonClick
* Na função verificamos se este valor é verdadeiro e mostramos um alerta se for.

Em seguida, queremos realmente salvar a pontuação.

Como você acha que fazemos isso? Se você disse valor do estado, está correto! Vá em frente e adicione outro valor de estado chamado “score”.

Lembre-se de prefixar a função para alterar o valor com “set” para que seja setScore. Inicialize-o para 0:

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

Em seguida, em vez de mostrar um alerta, queremos atualizar nossa pontuação em 1 se o usuário acertar a resposta.

Em nossa função handleAnswerButtonClick , remova o alerta e incremente nossa pontuação em um:

const handleAnswerButtonClick = (isCorrect) => {

if (isCorrect) {

setScore(score + 1);

}

Parte 6: Mostrando os pontos

Para mostrar a pontuação, basta fazer uma pequena alteração no nosso código de renderização. Em nosso JSX, remova a string codificada na seção de pontuação e adicione esta nova variável:

<div className='app'>

{}

{showScore ? ( // fluxo 1 "fim de jogo"

<div className='score-section'>

**Você acertou {score} de um total de {questions.length}**

</div>

Agora, se percorrermos as respostas, a pontuação é dinâmica e será exibida corretamente no final!

Código inicial:

import React, { useState } from 'react';

export default function App() {

const questions = [

{

questionText: 'What is the capital of France?',

answerOptions: [

{ answerText: 'New York', isCorrect: false },

{ answerText: 'London', isCorrect: false },

{ answerText: 'Paris', isCorrect: true },

{ answerText: 'Dublin', isCorrect: false },

],

},

{

questionText: 'Who is CEO of Tesla?',

answerOptions: [

{ answerText: 'Jeff Bezos', isCorrect: false },

{ answerText: 'Elon Musk', isCorrect: true },

{ answerText: 'Bill Gates', isCorrect: false },

{ answerText: 'Tony Stark', isCorrect: false },

],

},

{

questionText: 'The iPhone was created by which company?',

answerOptions: [

{ answerText: 'Apple', isCorrect: true },

{ answerText: 'Intel', isCorrect: false },

{ answerText: 'Amazon', isCorrect: false },

{ answerText: 'Microsoft', isCorrect: false },

],

},

{

questionText: 'How many Harry Potter books are there?',

answerOptions: [

{ answerText: '1', isCorrect: false },

{ answerText: '4', isCorrect: false },

{ answerText: '6', isCorrect: false },

{ answerText: '7', isCorrect: true },

],

},

];

return (

<div className='app'>

{/\* HINT: substitua "falso" pela lógica para exibir o

pontuação quando o usuário respondeu a todas as perguntas

\*/}

{false ? (

<div className='score-section'>Você acertou 1 de {questions.length}</div>

) : (

<>

<div className='question-section'>

<div className='question-count'>

<span>Question 1</span>/{questions.length}

</div>

<div className='question-text'>É aqui que a pergunta vai estar!</div>

</div>

<div className='answer-section'>

<button>Alternativa 1</button>

<button>Alternativa 2</button>

<button>Alternativa 3</button>

<button>Alternativa 4</button>

</div>

</>

)}

</div>

);

}