Blog

In this blog we will create a quiz app. The quiz will be mcq based.

To create a quiz app we will use API from Rapid API. It has questions in MCQ format. The url of API is

The response of API is of form:

1. Object
   1. code: 200
   2. data: (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
   3. msg: "success"
   4. \_\_proto\_\_: Object

At one time we get 10 questions.

We can access the questions using data.description, options using data.options and answers using data.answers respectively.

Create a class component App.

import React from 'react';

import './App.css';

class App extends React.Component {

  data = '';

  constructor(props){

    super(props);

    this.state={data:'',loading:true,index:0,inrange:true,displayAnswer:false}

    this.nextQuestion = this.nextQuestion.bind(this);

    }

// Add nextQuestion function

//Add componentDidMount Lifecycle Method

// update the render.

  render(){

    return (<div>Quiz App</div>

      );

    }

}

export default App;

We will use componentDidMount Lifecycle method for API getting API response. In componentDidMount we will add the API Key using fetch. In fetch we get data in readable stream so we will convert data to json to access the data.

 async componentDidMount(){

   try{

   const response = await fetch("https://gre-verbal.p.rapidapi.com/api/v1/questions?subcat=TC&count=10", {

    "method": "GET",

    "headers": {

      "x-rapidapi-host": "gre-verbal.p.rapidapi.com",

      "x-rapidapi-key": "enter your key"

    }

  })

  this.data = await response.json();

  console.log(this.data);

  this.setState({loading:false, data:this.data.data[this.state.index],displayAnswer:false})

}

catch(err){

  throw err;

}

 }

We added async await because API takes time to load and fetch data, in that time we will set loading state variable to true.

In render() we will use conditional rendering to load questions.

  render(){

    return (<div>

      {

this.state.loading ? <div>Loading</div> : this.state.inrange ?<div className="App"></div> : <button className="playAgain" onClick={ ()=>window.location.reload(false)}>Play Again</button>

}

      </div>);

}

This will be the basic block then we will call the Question and Anwser class component. So our render will look like:

  render(){

    return (<div>

      {

this.state.loading ? <div>Loading</div> : this.state.inrange ?

<div className="App">

<Question data={this.state.data}/> <Answer data={this.state.data}/>

</div> : <button className="playAgain" onClick={ ()=>window.location.reload(false)}> Play Again </button>

}

      </div>);}

Here we first check if data is loaded, then we check if variable is inrange if true then we render the Question and Answer component to display the questions and answers. If 10 questions have been displayed then we will get a Play Again button with help of inrange variable.

Now, we will add two buttons Get Answer and Next. So our render function becomes:

  render(){

    return (<div>

      {

this.state.loading ? <div>Loading</div> : this.state.inrange ?

<div className="App">

<Question data={this.state.data}/> <Answer data={this.state.data}/>

<button className="getAnswer" onClick={()=>{this.setState({displayAnswer:true});}}>

Get Answer

</button>

<button className="next" onClick={this.nextQuestion}>

Next

</button>

</div> : <button className="playAgain" onClick={ ()=>window.location.reload(false)}> Play Again </button>

}

      </div>);}

then we will call GetAnswer after the conditional rendering.

{this.state.displayAnswer? <GetAnswer data={this.state.data}/>:console.log("waiting")}

For Next question button we use nextQuestion function to get the next question here we also update the state of inrange variable by checking if less than 10 questions have been rendered:

  nextQuestion(){

     {

if(this.state.index <=8){

      this.setState((state)=>{

return {index : state.index +1}},()=>{

 this.setState({loading:false,data:this.data.data[this.state.index]}); })

      this.setState({loading:true,displayAnswer:false});

     }

     else{

       this.setState({inrange:false});

     }

    }

  }

We added a callback with Setstate. As it is asynchronous, therefore it will call the function once the state is updated. If we didn’t add the callback function we won’t get error but the questions will not load properly.

So our App component will look like:

class App extends React.Component {

  data = '';

  constructor(props){

    super(props);

    this.state={data:'',loading:true,index:0,inrange:true,displayAnswer:false}

    this.nextQuestion = this.nextQuestion.bind(this);

    }

// Add nextQuestion function

  nextQuestion(){

     {

if(this.state.index <=8){

      this.setState((state)=>{

return {index : state.index +1}},()=>{

 this.setState({loading:false,data:this.data.data[this.state.index]}); })

      this.setState({loading:true,displayAnswer:false});

     }

     else{

       this.setState({inrange:false});

     }

    }

  }

//Add componentDidMount Lifecycle Method

 async componentDidMount(){

   try{

   const response = await fetch("https://gre-verbal.p.rapidapi.com/api/v1/questions?subcat=TC&count=10", {

    "method": "GET",

    "headers": {

      "x-rapidapi-host": "gre-verbal.p.rapidapi.com",

      "x-rapidapi-key": "enter your key"

    }

  })

  this.data = await response.json();

  console.log(this.data);

  this.setState({loading:false, data:this.data.data[this.state.index],displayAnswer:false})

}

catch(err){

  throw err;

}

 }

// update the render.

    render(){

    return (<div>

      {

this.state.loading ? <div>Loading</div> : this.state.inrange ?

<div className="App">

<Question data={this.state.data}/> <Answer data={this.state.data}/>

<button className="getAnswer" onClick={()=>{this.setState({displayAnswer:true});}}>

Get Answer

</button>

<button className="next" onClick={this.nextQuestion}>

Next

</button>

</div> : <button className="playAgain" onClick={ ()=>window.location.reload(false)}> Play Again </button>

}

{this.state.displayAnswer? <GetAnswer data={this.state.data}/>:console.log("waiting")}

      </div>);}

export default App;

Now we need to write three more components: Question,Answer and GetAnswer.

Lets first complete the Question component which will show the questions

The Question component can be written as:

class Question extends React.Component{

  constructor(props){

    super(props)

    this.state = {data : this.props.data}

  }

  componentDidUpdate(nextProps) {

    // You don't have to do this check first, but it can help prevent an unneeded render

    if (nextProps.data !== this.state.data) {

      this.setState({ data: nextProps.data });

    }

  }

  render(){

  return(

        <div className="question">{this.state.data.description}</div>

        )}

}

To display the option we use Answer component which is written as:

class Answer extends React.Component{

  constructor(props){

    super(props);

    this.state={data:this.props.data,activeButton:[]}

    this.changeColor = this.changeColor.bind(this);

  }

/\*

We use ActiveButton array to store the value of active button, i.e button clicked by

the user so that we can change its color. In checkAnswer function we add that item

clicked to activeButton.

activeButton.some(val => val === item ) is used to check if one of the value in array

is equal to that of item.

\*/

  changeColor(item){

    this.setState(prevState=>({activeButton: [...prevState.activeButton , item.item]}));

    }

  render(){

    const optns = this.state.data.options;

    return(

    <div>

      {Object.keys(optns).map((optn) => {

          return <div key={optn}>

            {

              optns[optn].map((item,index)=>{

                  return <button className="optn-button" style={{backgroundColor: this.state.activeButton.some(val => val === item ) ? 'red':'blue'}} key={index}

                   onClick={()=> {

                    this.changeColor({item});

                  }

                }>{item}</button>

              })

            }

          </div>

        })}

    </div>

    )

  }

}

To display the correct answer GetAnswer function is written as:

// This Class component is used to show the correct answer

class GetAnswer extends React.Component

{

 constructor(props){

   super(props);

   this.state = {data:this.props.data}

  }

 render(){

   const arr=[];  // arr array is used to storeindex of correct answer

   const final =[]  //It is used to store value of corect answer

    const answers=this.state.data.answers;

    const optns = this.state.data.options;

    /\*

      Object.values(object\_name) gives array of values of given object. Here it will be of form array of array.

    \*/

    const optn = Object.values(optns);

    // Here we implement the logic to store correct ans index in arr. Since answers is of form array of array

    // we use nested maps.

    answers.map((ans)=>{

      ans.map((item)=>{

        arr.push(item)

      })

    });

    //Store correct answer value in final array

    for(var i=0;i<arr.length;i++){

          final.push(optn[i][arr[i]]);

    }

      const ansfinal = final.map((ans)=>{

        return <div key={ans}>

          <div className="ans">{ans}</div>

        </div>

      })

      return <div className="showAnswer">

              <div>Answers:</div>

              {ansfinal}

            </div>

 }

}

So here we complete our Quiz App.