**TYPESCRIPT:**

Typescript is used for type checking. It is used to check all errors related to typescript before runtime.

It is a transpiler or transcompiler based language. It is based on javascript. When we write something in typescript. Typescript compiler turns it into js which then runs into the browser. Js runs in browser, not typescript. In typescript, we write js syntax in the form typed language. By this, all typescript identify error in compiling time. Typescript is the super set of js. It decreases development time by 1/2.

File extension -> filename.ts

**Installation:**

The below will install typescript globally in whole computer.

npm install -g typescript

**Starting:**

Create index.ts file

Then write tsc .\above file name

**Example:**

Tsc .\index.ts

This will create js file

**Create complete project:**

Npm init

Then name of the project

We can change the description entry point etc later

After that, it will create package.json

**For tsconfig.json:**

tsc –init

Like in react, we create components folder and place all of our files in that, in typescript, the best practice is to create src folder and put all src files their. So compiler compiles all the ts files in it.

**Some changes in the ts.config:**

     "outDir": "./dist",                        /\* Redirect output structure to the directory. \*/

     "rootDir": "./src",

After that create src folder. Move index.ts in it. Now we just need to write

tsc

In the vs terminal, that will compile all file in the src folder and generate dist folder containing .js files like index.js

If we want to check output through node, then

Node .\dist\index.js

We can simply create index.html file and use index.js or any .js file that was created from .ts file

**Sumarizing the starting steps:**

1. Npm init
2. tsc –init
3. Some changes in the ts.config:

     "outDir": "./dist",                        /\* Redirect output structure to the directory. \*/

     "rootDir": "./src",

1. create src folder.
2. Index.ts in it.

**To check errors:**

Write tsc in vs terminal

**To check output of .js:**

node .\dist\index.js or file.js

**any Type:**

If we want a type in typescript that can take any value, we use type “any”. Type any is used when want dynamic variable.

let b : number = 10;

// we cannot do this b = "string"

let c : any = 100;

c = "type change due to any";

//as due to any type, we can assign any type to variable c

console.log(c);

Now we can also assign constant type like this. This is like something when we want some serious strict action

let d : true = true;

// d = false; -> gives an error as we assign true type, so it will only accept true

let e : 20 = 20;

// e = 30 or any number; -> gives an error as we assign 20 type, so it will only accept 20

Symbol data type can be used to create symbols

We can also specify certain properties or key value in object data type

let user : {firstName: string, lastName: string} = {

    firstName: "Muhammad Muneeb Waseem",

    lastName: "Waseem"

}

// we can not add more key value pair or more properties in object user as we specify properties initially

console.log(user);

Way of making react app with having typescript:

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

QUIZ APP:

Install:

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

Styled components:

npm i styled-components @types/styled-components

React Dom:

npm i --save @types/react-dom

then

npm i -D @types/react

types give us the intellisense and act as bridge b/w js and tsc.

Api URL:

<https://opentdb.com/api.php?amount=10&category=9&difficulty=medium&type=multiple>

Working:

Properties for questioncard:

Like question in string type, answers are 4 so define string array as answer are in string in array, callback for promises, userAnswers is the answer selected by the user and so on

type Props = {

    question: string;

    answers: string[];

    callback: any;

    userAnswer: string;

    questionNum: number;

    totalQuestions:number;

}

Using properties in QuestionCard and destructure them to be used:

export const QuestionCard: React.FC<Props> = ({question, answers, callback, userAnswer, questionNum, totalQuestions}) => {

    return (

        <div>

            <p>

            Question: {questionNum} / {totalQuestions}

            </p>

        </div>

    )

For questions:

<p dangerouslySetInnerHTML={{\_\_html:question}}></p>

dangerouslySetInnerHTML is used to set the value of the p element using js

For answers:

            <div>

                {answers.map(answer => (

                    <div>

                        <button>

                            <span dangerouslySetInnerHTML={{\_\_html:answer}} />

                        </button>

                    </div>

                ))}

            </div>

Mapping over all answers and for each answer, creating buttons to select by the user

Again moving into App.tsx:

Creating state hooks for each of our element:

  //for loading stage, loading...

  const [loading, setLoading] = useState(false);

initially false as It will true when we load our page

  //for loading stage, loading...

  const [loading, setLoading] = useState(false);

  //for question data

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

  //for question number

  const [number, setNumber] = useState(0);//as initially 0 question

  //for answer

  const [usetAnswers, setUserAnswers] = useState([]);

  //for score

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

  //for app to know whether game is over or not

  const [gameOver, setGameOver] = useState(true)//as for now, game is over, whenuser starts, then gameover changes to false which means game is in progress

Passing properties to questionCard element:

question={questions[number].question}

as from api, we get questions in array form, so for each question we pass index number to array question, like question 0, then 1 etc.

same for the answers

answers={questions[number].answers}

As userAnswer is the answer selected by the user, so we want to save that answer to check correct or not, so we select which answer user has select, otherwise undefined

userAnswer={userAnswers ? userAnswers[number] : undefined}

for answer checking:

callback={checkAnswer}

const checkAnswer = (e: React.MouseEvent<HTMLButtonElement>)=> {}

What does checkAnswer do is that it will be functional when user click on some of the option, so for that we use event e which works on mouse event and works when user click on the html button

Now data from api:

We are creating dynamic api, all would be same, just mae no of answers dynamic, difficulty dynamic etc

export const fetchQuestions = async(amount: number, difficulty: Difficulty) => {

    const endpoint = `https://opentdb.com/api.php?amount=${amount}&difficulty=${difficulty}&type=multiple`;

}

For difficulty, we used enum data type

export enum Difficulty {

    EASY = "easy",

    MEDIUM = "medium",

    HARD = "hard",

}

Now getting data from the above created api and convert it to json

    const data = await(await fetch(endpoint)).json();

Importing data from api in app.tsx:

import { fetchQuestions, Difficulty} from './API';

 console.log(fetchQuestions(TOTAL\_QUESTIONS, Difficulty.EASY));

As from api, we are getting data In the form

1. results: Array(10)
   1. 0:
      1. category: "Geography"
      2. correct\_answer: "Lion"
      3. difficulty: "easy"
      4. incorrect\_answers: (3) ["Bull", "Horse", "Dog"]
      5. question: "The body of the Egyptian Sphinx was based on which animal?"
      6. type: "multiple"

so we have set type for each of these, to use in our question

export type Question = {

    category: string;

    correct\_answer: string;

    incorrect\_answers: string[];

    question: string;

    type: string;

}

Now for our app, we want correct answer and incorrect answer in single array, so for that we create another type

export type QuestionState = Question & {answers: string[]};

Creating shuffle and sorted array in utilities.tsx file:

//sorting and shuffling the data in array so that each time, correct answer postion from all of the answers changes

export const shuffleArray = (array: any[]) =>

    [...array].sort(() => Math.random() - 0.5)

Here […array] means for whole array which consist correct answer and the rest of the incorrect answers

These three dots is a spread operator, that takes existing all data and creating a new array

Now we going to use this API.tsx to shuffle our data

export const fetchQuestions = async(amount: number, difficulty: Difficulty) => {

    const endpoint = `https://opentdb.com/api.php?amount=${amount}&difficulty=${difficulty}&type=multiple`;

    const data = await(await fetch(endpoint)).json();

    return data.results.map((question: Question) => (

        {

            ...question,

            answers: shuffleArray([...question.incorrect\_answers, question.correct\_answer])

        }

    ))

}

What happening in …

    return data.results.map((question: Question) => (

        {

            ...question,

            answers: shuffleArray([...question.incorrect\_answers, question.correct\_answer])

        }

Is that our data is in results object, so we map over it using data.results.map, that we pass Question data type for each question, and for all questions, we have answers that contain both incorrect and correct answers together and shuffle them using shuffleArray which we have created in utilities.tsx.

Now we want to use this question answer data into our main app to display, so for that

We import

import { fetchQuestions, Difficulty, QuestionState} from './API';

Here QuestionState contains question and answers

Now update our question and answer part in the app.

So we get question in the type QuestionState

  const [questions, setQuestions] = useState<QuestionState[]>([]);

And for userAnswer;

  const [userAnswers, setUserAnswers] = useState<AnswerObject[]>([]);

where AnswerObeject is

type AnswerObject = {

  question: string;

  answer: string;

  correct: boolean;

  correctAnswer: string;

}

So its describing type like correct is either true or false, and correctAnswer selected by user is of string type

Now lets make our begin quiz button functionable:

  const startQuiz = async() => {

    setLoading(true);//initially loading

    setGameOver(false);//as by clicking on the begin quiz, quiz has started

    const newQuestions = await fetchQuestions(TOTAL\_QUESTIONS, Difficulty.EASY);//fetching new questions from the api

    setQuestions(newQuestions);//update questions

    setScore(0);//again setscore 0 removing prevously played score

    setUserAnswers([]);//store user answer

    setNumber(0);//again setting number 0 from previeously played quiz

    setLoading(false);//as questions has been fetchd, so setLoading to false

  };

Now we only want to show begin quiz button when user has completed the quiz(gameOver) or it completed all of the questions(TOTALQUESTIONS) and if that then show begin quiz, otherwise if it doing the quiz, then don’t show the begin quiz button

      {gameOver || userAnswers.length === TOTAL\_QUESTIONS ? (

        <button className='start' onClick={startQuiz}>

        Begin Quiz

        </button>) : null}

Where || is called or operator

Now we want to show the score till game is running(!gameOver), if it is ended (gameOver) then don’t show the score.

      {!gameOver ? (

      <p className='score'>

        Score:

        </p>): null}

      <p>Loading</p>

Similarly, if data is loading, then show loading para other wise don’t show the loading para

      {loading? (

      <p>

        Loading

        </p>): null}

Similarly, if not loading(!loading) and game is running and not over(!gamOver) then continue to show question card, it loading and gameOver then don’t show it

      {!loading && !gameOver ? (

      <QuestionCard

        questionNum={number + 1}

        totalQuestions = {TOTAL\_QUESTIONS}

        question={questions[number].question}

        answers={questions[number].answers}

        userAnswer={userAnswers ? userAnswers[number] : undefined}

        callback={checkAnswer}

      />): null}

Similarly, if game is running(!gameOver) and page is not loading(!loading) and users answers are equal to current question number and current question number is not equal to TOTALQUESTIONS(we pass -1 as number is an array so it starts from 0, so make TOTALQUESTIONS 1 less to match array number) then show the next button

      {!gameOver && !loading && userAnswers.length === number + 1 && number !== TOTAL\_QUESTIONS - 1 ? (

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

        Next

      </button>): null}

Now upon clicking the any of the options, we need to check whether the selected option is correct or not. If game is running, then store the selected answer the the const answer. Then compare whether api correct answer is same as the user answer(answer). If correct, then update previous score by 1.

  const checkAnswer = (e: React.MouseEvent<HTMLButtonElement>)=> {

    if (!gameOver){

      const answer = e.currentTarget.value;

      const correct = questions[number].correct\_answer === answer;

      if (correct) setScore(prev => prev + 1)

    }

  };

Now updating the answer selected by the user

      const answerObject = {

        question: questions[number].question,

        answer,

        correct,

        correctAnswer: questions[number].correct\_answer

       }

      setUserAnswers(prev => [...prev, answerObject])

    }

Where answerObject is of api data like question from api, answer by user, correct answer. By this, we can take history of all the users selected answers.

Updating Next question function:

It will work as we click on next button to show next question. When we click on next button, our question number will be updated by 1. If we are on the tenth question, then after that, game will be over so we need no next button for that as no questions left, else move on to next question

  const nextQuestion = async() => {

    const nextQuestion = number + 1;

    if (nextQuestion === TOTAL\_QUESTIONS){

      setGameOver(true);

    }

    else{

      setNumber(nextQuestion);

    }

  };

Now to update score when we select any option;

      {!gameOver ? (

      <p className='score'>

        Score: {score}

        </p>): null}

So everything done here.

Updating UI:

Create App.styles.ts and place all css code from adil altaf App.styles.ts their.

Then call Global style in App.tsx

import { GlobalStyle } from './App.styles'

first place GlobalStyle component(self closing tag) above all, as it will apply on whole app. Then apply wrapper inside it.

    <>

    <GlobalStyle />

    <Wrapper>

      <h1>Quiz App</h1>

      {gameOver || userAnswers.length === TOTAL\_QUESTIONS ? (

        <button className='start' onClick={startQuiz}>

        Begin Quiz

        </button>) : null}

      {!gameOver ? (

      <p className='score'>

        Score: {score}

        </p>): null}

      {loading? (

      <p>

        Loading

        </p>): null}

      {!loading && !gameOver ? (

      <QuestionCard

        questionNum={number + 1}

        totalQuestions = {TOTAL\_QUESTIONS}

        question={questions[number].question}

        answers={questions[number].answers}

        userAnswer={userAnswers ? userAnswers[number] : undefined}

        callback={checkAnswer}

      />): null}

      {!gameOver && !loading && userAnswers.length === number + 1 && number !== TOTAL\_QUESTIONS - 1 ? (

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

        Next

      </button>): null}

    </Wrapper>

    </>

Now this is to style our app.

Now to style question card:

Create QuestionCard.styles.tsx and paste code their from sir adil altaf QuestionCard.styles.tsx. It contain two main component. One is for whole QuestionCard called wrapper. So we will place it at the top. Second is for button called Button Wrapper, which is of two types. One is which is correct from the api, second is what user clicked. So correct one which is from api will always be green as it is the correct one, where as the userClicked can be green (if user select correct answer) or can be red (if user select wrong answer)

So import these two components in the QuestionCard.tsx and place them in the return statement

   return (

        <Wrapper>

            <p>

                Question: {questionNum} / {totalQuestions}

            </p>

            <p dangerouslySetInnerHTML={{ \_\_html: question}} />

            <div>

                {answers.map(answer => (

                    <ButtonWrapper

                        key={answer}

                        correct = {userAnswer?.correctAnswer === answer}

                        userClicked = { userAnswer?.answer === answer }

                    >

                        <button disabled={userAnswer} value={answer} onClick={callback}>

                            <span dangerouslySetInnerHTML={{ \_\_html: answer }} />

                        </button>

                    </ButtonWrapper>

                ))}

            </div>

        </Wrapper>

    )