**login.component.ts:**

import { Component, OnInit } from '@angular/core';

import {

FormBuilder,

FormControl,

FormGroup,

Validators

} from '@angular/forms';

import { Router } from '@angular/router';

import { DataserviceService } from '../dataservice.service';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css'],

})

export class LoginComponent implements OnInit {

public fullName!: string | null | undefined;

public password!: string | null | undefined;

user = { fullName: '', password: '' };

constructor(private ds: DataserviceService, private router: Router) {}

ngOnInit(): void {}

loginForm = new FormGroup({

fullName: new FormControl('', [

Validators.required,

Validators.minLength(3),

]),

password: new FormControl('', [

Validators.required,

Validators.minLength(3),

]),

});

get f() {

return this.loginForm.controls;

}

onSubmit() {

console.log("flow 3 - after the user submits or clicks the button in login.component.ts");

// console.table(this.loginForm.value);

this.fullName = this.loginForm.get('fullName')?.value;

this.password = this.loginForm.get('password')?.value;

// The error message Type 'string | null | undefined' is not assignable to type 'string'. Type 'undefined' is not assignable to type 'string'. is a TypeScript error that occurs when you try to assign a value of type string | null | undefined to a variable of type string1.

// In your code, the error is caused by the line this.user = {fullName: this.fullName, password: this.password};

// The error message suggests that the value of this.fullName or this.password might be null or undefined1.

// This will only assign the value if both this.fullName and this.password are not null or undefined.

if (this.fullName && this.password) {

this.user = { fullName: this.fullName, password: this.password };

}

console.log("flow 4 - from loigin.component.ts after getting the data from the user " + " " + this.user);

this.ds.checkLogin(this.user).subscribe({

next: (data) => {

console.log("inside the checkLogin() function in login.component.ts " + " " + data);

if (data) {

alert('Login successful!');

sessionStorage.setItem('user', this.user.fullName);

this.router.navigate(['/quizes']);

}

},

});

// if (this.fullName == 'vishnu' && this.password == '12345') {

// this.router.navigate(['/quizes']);

// } else {

// console.log('error');

// }

// }

}

}

**login.component.html:**

<div class="container">

<form [formGroup]="loginForm" (ngSubmit)="onSubmit()" class="form">

FullName:

<input

type="text"

formControlName="fullName"

placeholder="FullName"

class="form-control"

/>

<div \*ngIf="f['fullName'].touched && f['fullName'].invalid">

<p \*ngIf="f['fullName'].errors && f['fullName'].errors['required']">

Name is required

</p>

</div>

Password: <br>

<input type="password" formControlName="password" placeholder="password" class="form-control" />

<button type="submit" class="btn btn-primary">Login</button>

</form>

</div>

**Quizzes.component.ts:**

import { Component, OnInit } from '@angular/core';

import { Quiz } from '../quiz';

import { DataserviceService } from '../dataservice.service';

@Component({

selector: 'app-quizzes',

templateUrl: './quizzes.component.html',

styleUrls: ['./quizzes.component.css'],

providers: [DataserviceService],

})

export class QuizzesComponent implements OnInit {

questions!:Quiz[];

isSubmitted:boolean=false;

selectedOption!:any;

numberOfCorrectAnswers:number = 0;

resultText!:string;

isCorrect:boolean[] = [false];

selectedQuestion!: Quiz;

answered:boolean[] = [false];

answerResult:string[] = [''];

viewResult:boolean = false;

constructor(private ds:DataserviceService){}

ngOnInit(): void {

this.ds.getQuiz().subscribe({

next: data=>this.questions = data

})

}

getSelectedOption(questionIndex: number, question:Quiz){

console.log(questionIndex, question, " " + this.selectedOption);

// means that the button is clicked

this.isSubmitted = true;

// changing the selectedQuestion to the currently selected question

this.selectedQuestion = question;

// checking the result using isCorrectAnsuwer function

let result = this.isCorrectAnswer(questionIndex);

// to check if the user has answered

this.answered[questionIndex] = true;

if(result){

this.resultText = "Answer is correct";

// stores the resultText in the answerResult list at a particular index

this.answerResult[questionIndex] = this.resultText;

// stores boolean in the isCorrect list at a particular index

this.isCorrect[questionIndex] = true;

}

else{

this.resultText = "Answer is wrong";

this.answerResult[questionIndex] = this.resultText;

this.isCorrect[questionIndex]=false;

}

}

isCorrectAnswer(questionIndex:number) {

// returns true if the answer of a question at a particular index answer matched the selectedOption which is two way binded in the html

const isCorrect = this.questions[questionIndex].answer === this.selectedOption;

if (isCorrect) {

this.numberOfCorrectAnswers++;

}

return isCorrect;

}

// to check the boolean at a partcular index position

isCorrect1(i:number){

return this.isCorrect[i];

}

isAnswered(i:number){

return this.answered[i];

}

getResult(){

this.viewResult = true;

}

// isAllQuestionAnswered(){

// if(this.questions.length == this.answered.length){

// return true;

// }

// else{

// return false;

// }

}

**Quizzes.component.html:**

<app-navbar></app-navbar>

<div class="container">

<!-- In the example you provided, let j=index is used to declare a local variable j and assign it the value of the current index of the loop.

The index variable is provided by the \*ngFor directive and represents the current index of the loop. -->

<div \*ngFor="let question of questions; let i = index" class="card mb-3">

<div class="card-body">

<h5 class="card-title">{{ question.question }}</h5>

<div

\*ngFor="let choice of question.choices; let j = index"

class="form-check"

>

<!-- [(ngModel)] is a two way binding from html to ts and vice versa -->

<!-- In the example you provided, id="choice{{j}}" is used to generate a unique identifier for each radio button.

The {{j}} syntax is used to interpolate the value of the j variable into the string. -->

<input

class="form-check-input"

type="radio"

name="choice"

id="choice{{ j }}"

[value]="choice"

[(ngModel)]="selectedOption"

/>

<!-- In the example you provided, for="option{{j}}" is used to associate the label with the corresponding radio button.

The for attribute specifies which form element the label is bound to. In this case, it is bound to the radio button with the id attribute of "option{{j}}". -->

<label class="form-check-label" for="option{{ j }}">

{{ choice }}

</label>

</div>

<button

class="btn btn-primary"

(click)="getSelectedOption(i, question)"

[disabled]="selectedQuestion === question || isAnswered(i)"

>

Check Answer

</button>

<!-- if the selectedQuestion(which is the 2-way binded answer) is equal to the answer at question or isAnswered(i) function is true -->

<div \*ngIf="selectedQuestion === question || isAnswered(i)">

<div [ngClass]="isCorrect1(i) ? 'correct' : 'wrong'">

{{ answerResult[i] }}

</div>

</div>

</div>

</div>

<button class="btn btn-primary mt-5 mb-3" (click)="getResult()">

Check Result

</button>

<div \*ngIf="viewResult">{{ numberOfCorrectAnswers }}</div>

</div>

**Data.service.ts:**

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { User } from 'user';

@Injectable({

providedIn: 'root'

})

export class DataserviceService {

headers!: HttpHeaders;

constructor(private http : HttpClient) {

// This line initializes an instance of the HttpHeaders class and sets the content-type header to application/json.

// The content-type header specifies the format of the request payload.

// In this case, it is set to application/json which means that the payload is in JSON format.

// The content-type header specifies the format of the request payload. In this case, it is set to application/json which means that the payload is in JSON format.

// However, you can set the content-type header to other formats such as text/plain or application/xml if you are sending data in those formats.

// The content-type header specifies the format of the request payload. It is used to indicate the format of the data that is being sent in the request body. This is important because the server needs to know how to parse the data that is being sent.

// The default content-type for an HTTP POST request in Angular is text/plain. However, it is recommended to set the content-type header to the appropriate value for the data that is being sent in the request body.

// This is important because the server needs to know how to parse the data that is being sent.

// If you are sending JSON data, then you should set the content-type header to application/json.

this.headers = new HttpHeaders().set('content-type','application/json');

}

getQuiz(): Observable<any>{

console.log("flow 1 - from getQuiz() method in service");

return this.http.get<any[]>('api/quiz');

}

checkLogin(user: User): Observable<boolean>{

console.log('flow 5 - inside checkLogin() function in dataservice ' + " " + user);

return this.http.post<boolean>('api/check-login', user);

}

}

**Server.js:**

// import express

const express = require('express');

// put express in app variable

const app = express();

// import path - The Path module provides a way of working with directories and file paths.

const path = require('path');

// import body parser - body parser converts data into json and vice verse

// Parsing is a special form of conversion that deals with getting a value from an object of type string and changing that value to another type.

// Basically its a middleware for parsing JSON, plain text, or just returning a raw Buffer object for you to deal with as you require.

const bodyParser = require('body-parser');

// importing the api.js from routes folder

// The require() function reads the file specified by the path and then executes it.

// The module then exports an object which can be used by other parts of your application

const api = require('./servers/routes/api');

console.log("flow 1 - inside server.js receiving api" + " " + api);

// It parses the incoming request body and converts it to JSON format.

app.use(bodyParser.json());

// basically tells the system whether you want to use a simple algorithm for shallow parsing (i.e. false) or complex algorithm for deep parsing that can deal with nested objects (i.e. true).

app.use(bodyParser.urlencoded({extended:false}));

// The app. use() method mounts or puts the specified middleware functions at the specified path.

// This middleware function will be executed only when the base of the requested path matches the defined path.

// /api is the starting path for api's

// In this case, it is used to mount the api middleware function at the /api path.

// This means that any request that starts with /api will use the api middleware function.

// app.use('/api', function(req, res, next){

// console.log("flow 6 - inside app.use() in server.js");

// // it passes control to the next matching route

// req.get('Something');

// next(api);

// });

app.use('/api', api);

// for static files which includes all the compiled files

// app.use(express.static(path.join(\_\_dirname, 'dist/online-test-app')));

app.use(express.static(\_\_dirname + '/dist/online-test-portal', { type: 'module' }));

// getting the index.html file

// The line app.get('\*', function(req,res){ res.sendFile(path.join(\_\_dirname,'dist/online-test-app/index.html')); }); is used to serve the index.html file of your Angular application.

//The \* is a wildcard character that matches any URL path.

// This means that this route will be used for any GET request that doesn’t match any other route.

app.get('\*', function(req,res){

res.sendFile(path.join(\_\_dirname,'dist/online-test-portal/index.html'));

});

// mentioning the port to be 5000 but it was showing already in use so using 3000

const port = process.env.PORT || 3000;

// this line starts to listen in the port 3000

app.listen(port, console.log(`flow 2 - Server started on port ${port}`));

// module.exports = function(req,res,next){

// app.use('/api', api);

// }

**App.js:**

const express = require('express');

const app = express();

const api = express.Router();

// to handle file system

const fs = require('fs');

app.use(express.json());

// api to get the quizes from the data.json file

api.get('/quiz', function(req,res){

// fs.readFile reads the data.json file from the path and it has a callback function with error and data to handle

fs.readFile(\_\_dirname + '/data.json', function(err,data){

if(err){

res.status(404).send("Error reading file");

}

else{

const jsonData = JSON.parse(data);

res.json(jsonData);

}

})

});

// api to validate user name and email

api.post('/check-login', (req,res)=>{

let fullName = req.body.fullName;

let password = req.body.password;

console.log("flow 7 - inside api.post(/check-login)" + " " +fullName + " " + password);

// \_\_dirname is an environment variable that tells you the absolute path of the directory containing the currently executing file.

fs.readFile(\_\_dirname + '/user.json',(err,data) =>{

if(err){

console.log("flow 8 - file not found");

console.log(err);

return;

}

else{

const jsonData = JSON.parse(data);

const exists = jsonData.some(obj => obj.fullName === fullName && obj.password === password);

console.log("flow 8 - if data exists? " + exists);

if (exists) {

console.log("flow 9 - inside api.js true data exists and matching with user.json");

res.status(200).send(true);

} else {

console.log("flow 9 - inside api.js false data not matching with user.json");

res.status(404).send(false);

}

}

});

})

// If the require() function is returning an empty object, it could be due to a few reasons.

//One reason could be that the path specified in the require() function is incorrect.

//Another reason could be that the module being loaded is not exporting anything

//You mentioned that you have code inside the api.js file but it is returning an empty object.

//You may want to double-check that the module is exporting something using module.exports.

//If you are still having trouble, you can try adding a .js extension to the file path in the require() function

module.exports = api;