**BACKEND FIX (C# Web API)**

Here’s the **given code (buggy)**:

[HttpPost]public IActionResult AddJob(Job job) {     if (string.IsNullOrWhiteSpace(job.Title))     {         return BadRequest("Title is required");     }     jobs.Add(new Job     {         Id = nextId++,         Title = job.Title,         Description = job.Description,         Location = job.Location,         Type = job.Type,         ClosingDate = job.ClosingDate     });     return Ok(job); } [HttpDelete("{id}")]public IActionResult DeleteJob(int id) {     jobs.RemoveAll(j => j.Id == id);     return Ok("Job deleted"); }

**Problems:**

**No validation for ClosingDate** (it can be in the past).

**DeleteJob** doesn’t check if the job exists — it always says "Job deleted" even if the ID is wrong.

**GetJobs** doesn’t support filtering.

**Fixed Code**

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

namespace JobSystemAPI.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class JobsController : ControllerBase

{

private static List<Job> jobs = new List<Job>();

private static int nextId = 1;

// Get all jobs with optional filtering

[HttpGet]

public IActionResult GetJobs([FromQuery] string? type, [FromQuery] string? location)

{

var result = jobs.AsEnumerable();

if (!string.IsNullOrEmpty(type))

result = result.Where(j => j.Type?.Equals(type, StringComparison.OrdinalIgnoreCase) == true);

if (!string.IsNullOrEmpty(location))

result = result.Where(j => j.Location?.Equals(location, StringComparison.OrdinalIgnoreCase) == true);

return Ok(result);

}

// Add new job with validation

[HttpPost]

public IActionResult AddJob(Job job)

{

if (string.IsNullOrWhiteSpace(job.Title))

{

return BadRequest("Title is required");

}

if (job.ClosingDate <= DateTime.Now)

{

return BadRequest("Closing date must be in the future");

}

jobs.Add(new Job

{

Id = nextId++,

Title = job.Title,

Description = job.Description,

Location = job.Location,

Type = job.Type,

ClosingDate = job.ClosingDate

});

return Ok(job);

}

// Delete job with existence check

[HttpDelete("{id}")]

public IActionResult DeleteJob(int id)

{

var existing = jobs.FirstOrDefault(j => j.Id == id);

if (existing == null)

{

return NotFound("Job not found");

}

jobs.Remove(existing);

return Ok("Job deleted");

}

}

// Job model

public class Job

{

public int Id { get; set; }

public string Title { get; set; } = string.Empty;

public string? Description { get; set; }

public string? Location { get; set; }

public string? Type { get; set; }

public DateTime ClosingDate { get; set; }

}

}

**FRONTEND FIX (Angular)**

Here’s the **given buggy code**:

loadJobs() { this.http.get('/jobs').subscribe((data: any) => { this.jobs = data; this.filteredJobs = this.jobs; }); } applyFilter() { this.filteredJobs = this.jobs; }

**Problems:**

Wrong API endpoint — should be /api/jobs, not /jobs.

Doesn’t actually filter jobs by type — applyFilter() just copies the list.

The component doesn’t handle asynchronous timing or filter changes properly.

**Fixed Angular Code**

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

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

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

@Component({

selector: 'app-job-list',

standalone: true,

imports: [CommonModule, FormsModule],

template: `

<div class="filter-section">

<input

[(ngModel)]="filterType"

placeholder="Filter by type"

(input)="applyFilter()"

class="filter-input"

>

<input

[(ngModel)]="filterLocation"

placeholder="Filter by location"

(input)="applyFilter()"

class="filter-input"

>

<button (click)="clearFilters()" class="clear-btn">Clear Filters</button>

</div>

<div \*ngIf="loading" class="loading">Loading jobs...</div>

<div \*ngIf="error" class="error">

Error loading jobs: {{ error }}

</div>

<ul \*ngIf="!loading && !error" class="job-list">

<li \*ngFor="let job of filteredJobs" class="job-item">

<h3>{{ job.title }}</h3>

<p><strong>Type:</strong> {{ job.type }}</p>

<p><strong>Location:</strong> {{ job.location }}</p>

<p><strong>Description:</strong> {{ job.description }}</p>

<p><strong>Closes:</strong> {{ job.closingDate | date:'mediumDate' }}</p>

</li>

</ul>

<div \*ngIf="!loading && filteredJobs.length === 0" class="no-jobs">

No jobs found matching your criteria.

</div>

`

})

export class JobListComponent implements OnInit {

jobs: any[] = [];

filteredJobs: any[] = [];

filterType: string = "";

filterLocation: string = "";

loading: boolean = false;

error: string = "";

constructor(private http: HttpClient) {}

ngOnInit() {

this.loadJobs();

}

// Fixed: Correct API endpoint and proper async handling

loadJobs() {

this.loading = true;

this.error = "";

this.http.get<any[]>('/api/jobs').subscribe({

next: (data: any) => {

this.jobs = data;

this.applyFilter(); // Apply any existing filters after loading

this.loading = false;

},

error: (err) => {

console.error('Error loading jobs:', err);

this.error = err.message || 'Failed to load jobs';

this.loading = false;

}

});

}

// Fixed: Proper filtering by type and location

applyFilter() {

if (!this.filterType && !this.filterLocation) {

this.filteredJobs = [...this.jobs];

return;

}

this.filteredJobs = this.jobs.filter(job => {

const typeMatch = !this.filterType ||

job.type?.toLowerCase().includes(this.filterType.toLowerCase());

const locationMatch = !this.filterLocation ||

job.location?.toLowerCase().includes(this.filterLocation.toLowerCase());

return typeMatch && locationMatch;

});

}

// Added: Clear filters functionality

clearFilters() {

this.filterType = "";

this.filterLocation = "";

this.applyFilter();

}

}

**Summary of What Was Fixed**

| **Problem** | **Fixed** |
| --- | --- |
| Backend: Missing closing date validation | Added validation |
| Backend: Delete always says “Job deleted” | Now checks if job exists |
| Frontend: Async bug & wrong endpoint | Fixed endpoint + added proper subscribe handling |
| Frontend: Filter didn’t work | Implemented real filter logic |

**PART 4 – Testing**

**Backend Unit Test**

Test in JobPostingAPI.Tests project:

[Fact]public void AddJob\_ShouldRejectPastClosingDate()  
{  
    var controller = new JobsController();  
    var result = controller.Add(new Job  
    {  
        Title = "Test",  
        Description = "Desc",  
        Type = "Full-Time",  
        Location = "Remote",  
        ClosingDate = DateTime.Now.AddDays(-1)  
    });  
  
    Assert.IsType<BadRequestObjectResult>(result);  
}

**Frontend Test (simplified)**

In job-list.component.spec.ts:

it('should filter jobs correctly', () => {  
  component.jobs = [  
    { title: 'A', type: 'Full-Time', location: 'NY' },  
    { title: 'B', type: 'Part-Time', location: 'LA' }  
  ];  
  component.filterType = 'Full-Time';  
  component.applyFilter();  
  expect(component.jobs.some(j => j.type !== 'Full-Time')).toBeFalse();  
});

**PART 5 – General Development Questions**

| **#** | **Question** | **Answer** |
| --- | --- | --- |
| 1 | **Synchronous vs Asynchronous** | Synchronous code runs one step at a time (blocks the thread). Asynchronous code allows other tasks to run while waiting for something (like an API call) to complete. |
| 2 | **Database Normalization** | Organizing data into related tables to reduce duplication and maintain consistency. |
| 3 | **Dependency Injection (DI)** | A way to give an object the dependencies it needs instead of creating them itself. It makes code easier to test and maintain. |
| 4 | **Secure API Endpoints** | Use authentication (JWT tokens), authorization, HTTPS, input validation, and rate limiting. |
| 5 | **Unit vs Integration Testing** | Unit tests test one small part (e.g., one function). Integration tests check if multiple parts work together properly. |