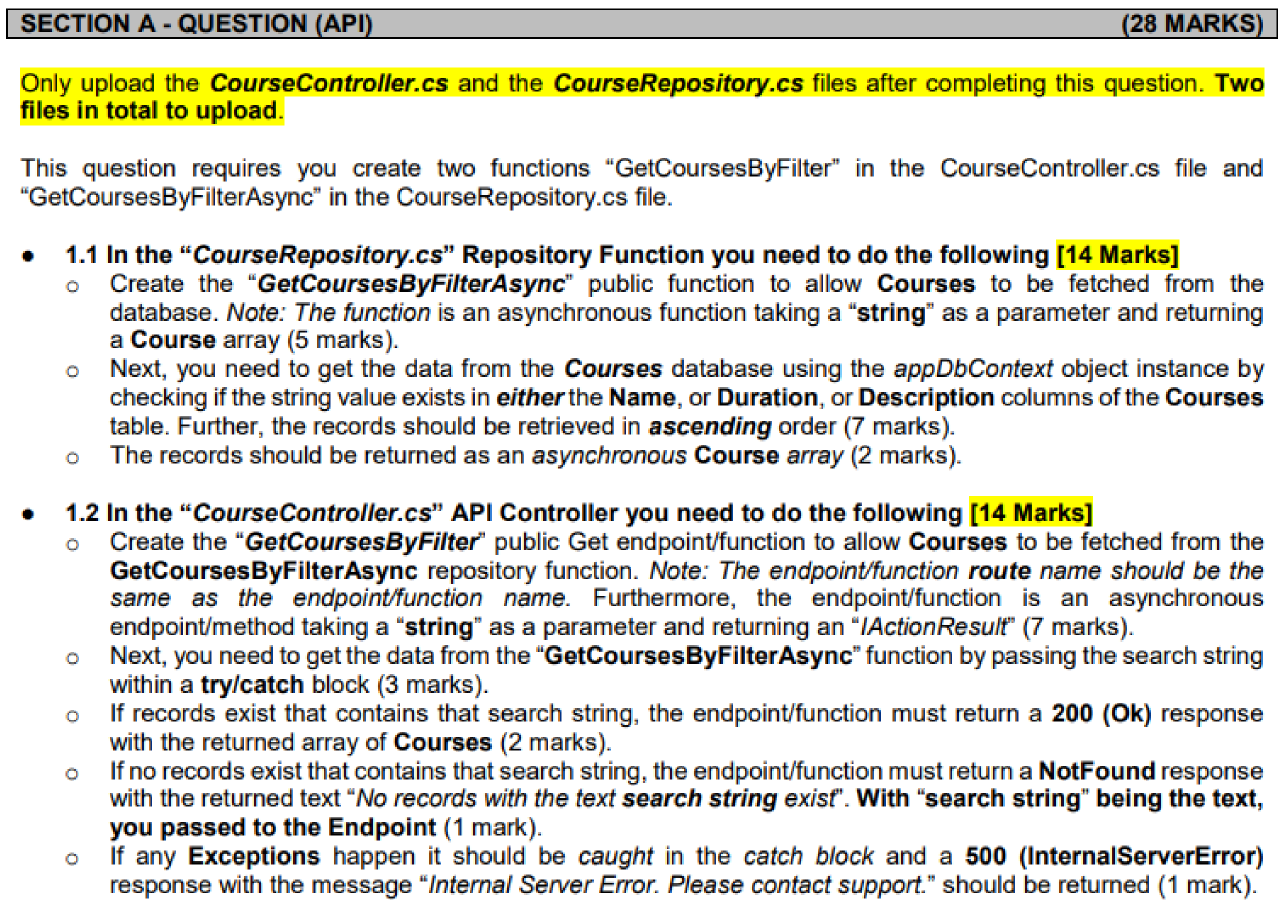
Semester Test 1 Prep + Cheat Sheet

`2023 Paper from AI:`



SECTION A - QUESTION (API) **(28 MARKS)**

Only upload the ***CourseController.cs*** and the ***CourseRepository.cs*** files after completing this question. **Two files in total to upload**.

This question requires you create two functions "GetCoursesByFilter" in the CourseController.cs file and "GetCoursesByFilterAsync" in the CourseRepository.cs file.

- **1.1 In the "*CourseRepository.cs*" Repository Function you need to do the following [14 Marks]**
  - Create the "***GetCoursesByFilterAsync***" public function to allow **Courses** to be fetched from the database. *Note: The function* is an asynchronous function taking a "**string**" as a parameter and returning a **Course** array (5 marks).
  - Next, you need to get the data from the ***Courses*** database using the *appDbContext* object instance by checking if the string value exists in ***either*** the **Name**, or **Duration**, or **Description** columns of the **Courses** table. Further, the records should be retrieved in ***ascending*** order (7 marks).
  - The records should be returned as an *asynchronous* **Course** *array* (2 marks).

- **1.2 In the "*CourseController.cs*" API Controller you need to do the following [14 Marks]**
  - Create the "***GetCoursesByFilter***" public Get endpoint/function to allow **Courses** to be fetched from the **GetCoursesByFilterAsync** repository function. *Note: The endpoint/function **route** name should be the same as the endpoint/function name.* Furthermore, the endpoint/function is an asynchronous endpoint/method taking a "**string**" as a parameter and returning an "*IActionResult*" (7 marks).
  - Next, you need to get the data from the "**GetCoursesByFilterAsync**" function by passing the search string within a **try/catch** block (3 marks).
  - If records exist that contains that search string, the endpoint/function must return a **200 (Ok)** response with the returned array of **Courses** (2 marks).
  - If no records exist that contains that search string, the endpoint/function must return a **NotFound** response with the returned text "*No records with the text **search string** exist*". **With "search string" being the text, you passed to the Endpoint** (1 mark).
  - If any **Exceptions** happen it should be *caught* in the *catch block* and a **500 (InternalServerError)** response with the message "*Internal Server Error. Please contact support.*" should be returned (1 mark).

- 1.1:

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

public class CourseRepository
{
    private readonly AppDbContext _appDbContext;

    public CourseRepository(AppDbContext appDbContext)
    {
        _appDbContext = appDbContext;
    }

    public async Task<Course[]> GetCoursesByFilterAsync(string search)
    {
        return await _appDbContext.Courses
            .Where(c => c.Name.Contains(search) ||
                        c.Duration.Contains(search) ||
                        c.Description.Contains(search))
            .OrderBy(c => c.Name)
            .ToArrayAsync();
    }
}
```

- 1.2:

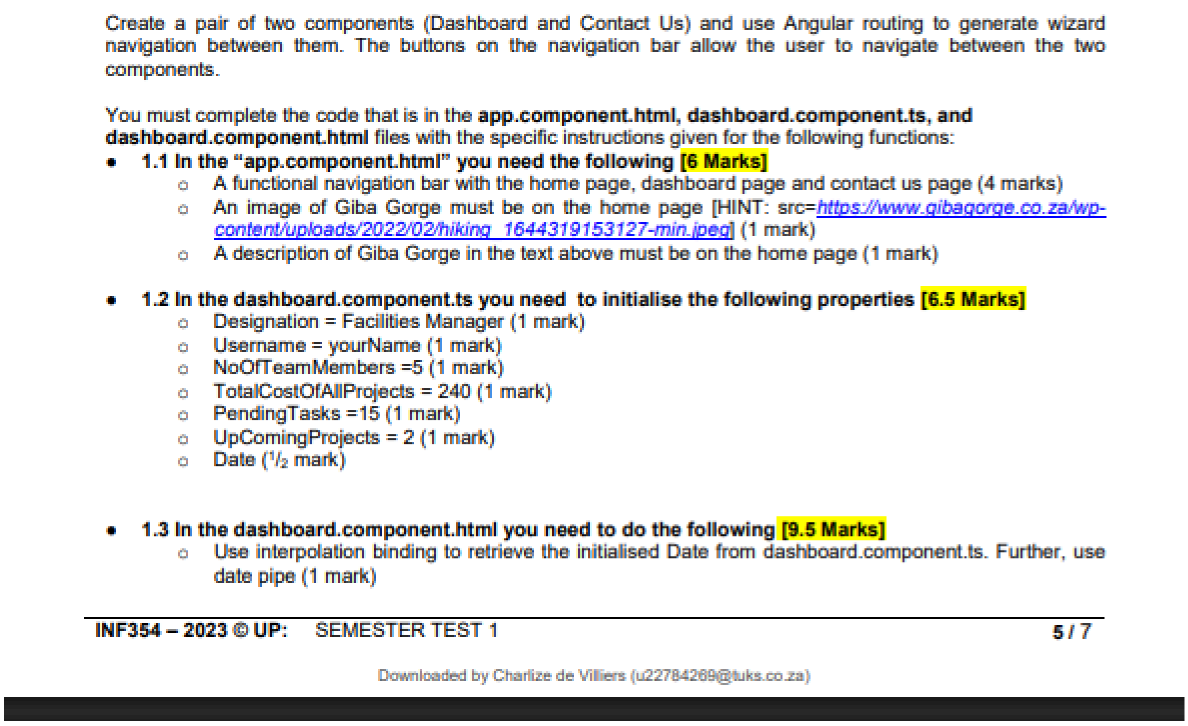
```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("api/[controller]")]
public class CourseController : ControllerBase
{
    private readonly CourseRepository _courseRepository;

    public CourseController(CourseRepository courseRepository)
    {
        _courseRepository = courseRepository;
    }

    [HttpGet("GetCoursesByFilter")]
    public async Task<IActionResult> GetCoursesByFilter(string search)
    {
        try
        {
            var courses = await
_courseRepository.GetCoursesByFilterAsync(search);

            if (courses.Length > 0)
            {
                return Ok(courses);
            }
            else
            {
                return NotFound($"No records with the text '{search}'
exist.");
            }
        }
        catch (Exception)
        {
            return StatusCode(500, "Internal Server Error. Please contact
support.");
        }
    }
}
```



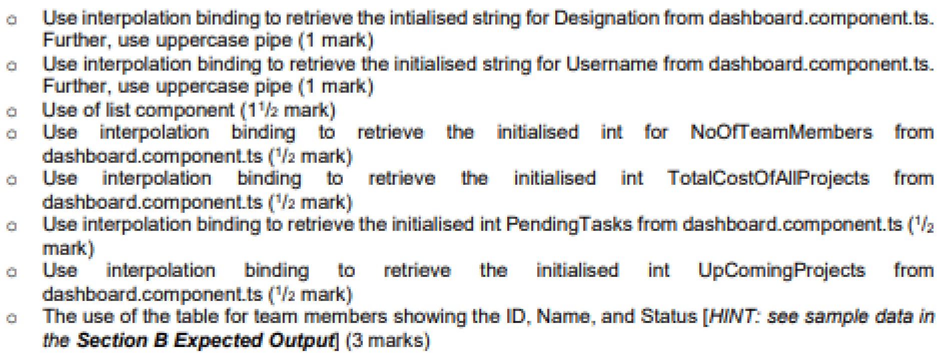
Create a pair of two components (Dashboard and Contact Us) and use Angular routing to generate wizard navigation between them. The buttons on the navigation bar allow the user to navigate between the two components.

You must complete the code that is in the **app.component.html, dashboard.component.ts, and dashboard.component.html** files with the specific instructions given for the following functions:

- **1.1 In the "app.component.html" you need the following [6 Marks]**
  - A functional navigation bar with the home page, dashboard page and contact us page (4 marks)
  - An image of Giba Gorge must be on the home page [HINT: src=https://www.gibagorge.co.za/wp-content/uploads/2022/02/hiking_1644319153127-min.jpeg] (1 mark)
  - A description of Giba Gorge in the text above must be on the home page (1 mark)

- **1.2 In the dashboard.component.ts you need to initialise the following properties [6.5 Marks]**
  - Designation = Facilities Manager (1 mark)
  - Username = yourName (1 mark)
  - NoOfTeamMembers =5 (1 mark)
  - TotalCostOfAllProjects = 240 (1 mark)
  - PendingTasks =15 (1 mark)
  - UpComingProjects = 2 (1 mark)
  - Date ($^1/_2$ mark)

- **1.3 In the dashboard.component.html you need to do the following [9.5 Marks]**
  - Use interpolation binding to retrieve the initialised Date from dashboard.component.ts. Further, use date pipe (1 mark)



  - Use interpolation binding to retrieve the intialised string for Designation from dashboard.component.ts. Further, use uppercase pipe (1 mark)
  - Use interpolation binding to retrieve the initialised string for Username from dashboard.component.ts. Further, use uppercase pipe (1 mark)
  - Use of list component (1$^1/_2$ mark)
  - Use interpolation binding to retrieve the initialised int for NoOfTeamMembers from dashboard.component.ts ($^1/_2$ mark)
  - Use interpolation binding to retrieve the initialised int TotalCostOfAllProjects from dashboard.component.ts ($^1/_2$ mark)
  - Use interpolation binding to retrieve the initialised int PendingTasks from dashboard.component.ts ($^1/_2$ mark)
  - Use interpolation binding to retrieve the initialised int UpComingProjects from dashboard.component.ts ($^1/_2$ mark)
  - The use of the table for team members showing the ID, Name, and Status [*HINT: see sample data in the **Section B Expected Output***] (3 marks)

- setup:

```
//app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';
import { ContactUsComponent } from './contact-us/contact-us.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'contact-us', component: ContactUsComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

- 1.1:

```
<!--app.component.html-->
<nav>
  <ul>
    <li><a routerLink="/">Home</a></li>
    <li><a routerLink="/dashboard">Dashboard</a></li>
    <li><a routerLink="/contact-us">Contact Us</a></li>
  </ul>
</nav>

<div class="content">
  <router-outlet></router-outlet>
</div>

<!-- Home Page Content -->
<div *ngIf="!currentRoute">
  <h1>Welcome to Giba Gorge</h1>
  <p>
    Giba Gorge Adventure Park lies in the Giba Valley alongside the N3
freeway, just outside Pinetown.
    Here you will find a haven for outdoor enthusiasts with something
suitable for all abilities and ages.
    Whether it's the kids scooting around the BMX track, the avid mountain
bikers & trail runners making use
    of the single-track trails, the adrenaline-fueled downhillers, or the
families and couples choosing to picnic
    on the lawn or relax in the outdoor restaurant, there is something to
appeal to everyone!
  </p>
  <img src="https://www.gibagorge.co.za/wp-
content/uploads/2022/02/hiking_1644319153127-min.jpeg" alt="Giba Gorge">
</div>
```

- 1.2:

```
//dashboard.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent {
  designation: string = "Facilities Manager";
  username: string = "yourName";
  noOfTeamMembers: number = 5;
  totalCostOfAllProjects: number = 240;
  pendingTasks: number = 15;
  upcomingProjects: number = 2;
  date: Date = new Date();

  teamMembers = [
    { id: 1, name: 'John Doe', status: 'Active' },
    { id: 2, name: 'Jane Smith', status: 'Inactive' },
    { id: 3, name: 'Emily Johnson', status: 'Active' },
    { id: 4, name: 'Michael Brown', status: 'Active' },
    { id: 5, name: 'Sarah Davis', status: 'Inactive' }
  ];
}
```

- 1.3:

```
<!--dashboard.component.html-->
<div class="dashboard">
  <h2>Dashboard</h2>
  <p>Date: {{ date | date:'fullDate' }}</p> <!-- (1 mark) -->

  <p>Designation: {{ designation | uppercase }}</p> <!-- (1 mark) -->
  <p>Username: {{ username | uppercase }}</p> <!-- (1 mark) -->

  <ul> <!-- (1.5 marks) -->
    <li>Number of Team Members: {{ noOfTeamMembers }}</li> <!-- (0.5 mark) -
->
    <li>Total Cost of All Projects: ${{ totalCostOfAllProjects }}</li> <!--
(0.5 mark) -->
    <li>Pending Tasks: {{ pendingTasks }}</li> <!-- (0.5 mark) -->
    <li>Upcoming Projects: {{ upcomingProjects }}</li> <!-- (0.5 mark) -->
  </ul>

  <h3>Team Members</h3>
  <table border="1"> <!-- (3 marks) -->
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Status</th>
    </tr>
    <tr *ngFor="let member of teamMembers">
      <td>{{ member.id }}</td>
      <td>{{ member.name }}</td>
      <td>{{ member.status }}</td>
    </tr>
  </table>
</div>
```

- final steps:
- run these commands:

```
ng generate component dashboard
ng generate component contact-us
```

- add components:

```
//app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { ContactUsComponent } from './contact-us/contact-us.component';

@NgModule({
  declarations: [
    AppComponent,
    DashboardComponent,
    ContactUsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

`Linq SQL stuff:`

Equivalent SQL Query

```
SELECT *
FROM Courses
WHERE Name LIKE '%search%'
   OR Duration LIKE '%search%'
   OR Description LIKE '%search%'
ORDER BY Name;
```

Explanation:

- `SELECT * FROM Courses` : Retrieves all columns from the `Courses` table.
- `WHERE Name LIKE '%search%' OR Duration LIKE '%search%' OR Description LIKE '%search%'` : Filters records where any column contains the search text.
- `ORDER BY Name` : Sorts results alphabetically by `Name` .

Handling Objects (c) in LINQ vs. SQL

Example: Filtering and Sorting

LINQ (C#)

```
var result = await _appDbContext.Courses
    .Where(c => c.Duration == "5 weeks" && c.TotalCost > 100)
    .OrderByDescending(c => c.TotalCost)
    .ToListAsync();
```

Equivalent SQL

```
SELECT *
FROM Courses
WHERE Duration = '5 weeks'
  AND TotalCost > 100
ORDER BY TotalCost DESC;
```

- Filters courses where `Duration` is exactly "5 weeks" and `TotalCost` is greater than 100.
- Sorts results in descending order of `TotalCost` .

Different Query Types in LINQ and SQL

LINQ Query (C#)	SQL Equivalent
`context.Courses.ToList();`	`SELECT * FROM Courses;`
`context.Courses.Where(c => c.Name == "SQL Basics");`	`SELECT * FROM Courses WHERE Name = 'SQL Basics';`
`context.Courses.Where(c => c.TotalCost > 100);`	`SELECT * FROM Courses WHERE TotalCost > 100;`

LINQ Query (C#)	SQL Equivalent
`context.Courses.OrderBy(c => c.Name);`	`SELECT * FROM Courses ORDER BY Name ASC;`
`context.Courses.OrderByDescending(c => c.TotalCost);`	`SELECT * FROM Courses ORDER BY TotalCost DESC;`
`context.Courses.FirstOrDefault(c => c.Name == "SQL Basics");`	`SELECT TOP 1 * FROM Courses WHERE Name = 'SQL Basics';`
`context.Courses.Count();`	`SELECT COUNT(*) FROM Courses;`
`context.Courses.Select(c => new { c.Name, c.Duration });`	`SELECT Name, Duration FROM Courses;`
`context.Courses.Any(c => c.Duration == "4 weeks");`	`SELECT CASE WHEN EXISTS (SELECT 1 FROM Courses WHERE Duration = '4 weeks') THEN 1 ELSE 0 END;`

Handling Complex Queries

4.1 Searching Multiple Columns

LINQ

```
var result = await _appDbContext.Courses
    .Where(c => c.Name.Contains(search) ||
                c.Duration.Contains(search) ||
                c.Description.Contains(search))
    .OrderBy(c => c.Name)
    .ToListAsync();
```

SQL

```
SELECT *
FROM Courses
WHERE Name LIKE '%search%'
   OR Duration LIKE '%search%'
   OR Description LIKE '%search%'
ORDER BY Name;
```

4.2 Filtering by Date

LINQ

```
var result = await _appDbContext.Courses
    .Where(c => c.StartDate >= DateTime.Now)
    .ToListAsync();
```

SQL

```
SELECT *
FROM Courses
WHERE StartDate >= GETDATE();
```

- `GETDATE()` retrieves the current date in SQL.

4.3 Aggregation Queries

LINQ

```
var totalCost = await _appDbContext.Courses.SumAsync(c => c.TotalCost);
```

SQL

```
SELECT SUM(TotalCost) FROM Courses;
```

- `SUM(TotalCost)` calculates the total cost of all courses.

4.4 Grouping Data

LINQ

```
var groupedCourses = await _appDbContext.Courses
    .GroupBy(c => c.Duration)
    .Select(g => new { Duration = g.Key, Count = g.Count() })
    .ToListAsync();
```

SQL

```
SELECT Duration, COUNT(*) AS Count
FROM Courses
GROUP BY Duration;
```

- Groups courses by `Duration` and counts how many exist in each group.

4.5 JOIN Queries

LINQ

```
var coursesWithInstructors = await _appDbContext.Courses
    .Join(_appDbContext.Instructors,
          course => course.InstructorID,
          instructor => instructor.ID,
          (course, instructor) => new { course.Name, Instructor =
instructor.Name })
    .ToListAsync();
```

SQL

```
SELECT c.Name, i.Name AS Instructor
FROM Courses c
INNER JOIN Instructors i ON c.InstructorID = i.ID;
```

- Joins the `Courses` and `Instructors` tables on `InstructorID` to get the instructor's name for each course.

Summary Table

LINQ Expression	SQL Equivalent
`.Where(c => c.Name == "SQL Basics")`	`WHERE Name = 'SQL Basics'`
`.OrderBy(c => c.TotalCost)`	`ORDER BY TotalCost ASC`
`.OrderByDescending(c => c.TotalCost)`	`ORDER BY TotalCost DESC`
`.Count()`	`SELECT COUNT(*) FROM Courses`
`.Sum(c => c.TotalCost)`	`SELECT SUM(TotalCost) FROM Courses`
`.FirstOrDefault()`	`SELECT TOP 1 * FROM Courses`
`.Join()`	`INNER JOIN`
`.GroupBy(c => c.Duration)`	`GROUP BY Duration`

Other Filtering Operators:

Operator	Description	Example
`=`	Exact match	`WHERE Name = 'SQL Basics'`
`LIKE '%search%'`	Contains search term	`WHERE Name LIKE '%SQL%'`

Operator	Description	Example
`LIKE 'SQL%'`	Starts with 'SQL'	`WHERE Name LIKE 'SQL%'`
`LIKE '%SQL'`	Ends with 'SQL'	`WHERE Name LIKE '%SQL'`
`>`	Greater than	`WHERE Duration > 10`
`<`	Less than	`WHERE Duration < 5`
`>=`	Greater than or equal to	`WHERE Duration >= 8`
`<=`	Less than or equal to	`WHERE Duration <= 6`
`IN`	Matches any in list	`WHERE Name IN ('SQL Basics', 'Advanced SQL')`
`BETWEEN`	Between two values	`WHERE Duration BETWEEN 5 AND 10`

Entity Framework (EF) to SQL Comparison Table

EF Core LINQ	Equivalent SQL
`context.Courses.ToArrayAsync();`	`SELECT * FROM Courses;`
`context.Courses.Where(c => c.Name.Contains("SQL"))`	`SELECT * FROM Courses WHERE Name LIKE '%SQL%';`
`context.Courses.OrderBy(c => c.Name)`	`SELECT * FROM Courses ORDER BY Name;`
`context.Courses.Count()`	`SELECT COUNT(*) FROM Courses;`
`context.Courses.FirstOrDefault(c => c.Name == "SQL Basics")`	`SELECT TOP 1 * FROM Courses WHERE Name = 'SQL Basics';`