Lifecycle Hooks for Component Initialization and Destruction

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1 Project Abstract

Component lifecycle management is crucial in Angular applications for handling tasks such as initialization, updates, and cleanup. This assignment focuses on the use of **Lifecycle Hooks**, particularly `ngOnInit` and `ngOnDestroy`, to properly initialize data when a component loads and handle cleanup when the component is destroyed.

The objective is to build a **Dynamic List Single Page Application (SPA)** where:

- A list of items is displayed.
- A filter input allows real-time filtering display.
- Lifecycle hooks control data initialization and destruction logging.

2 Problem Statement

You are required to develop a **Dynamic List SPA** using Angular 15.

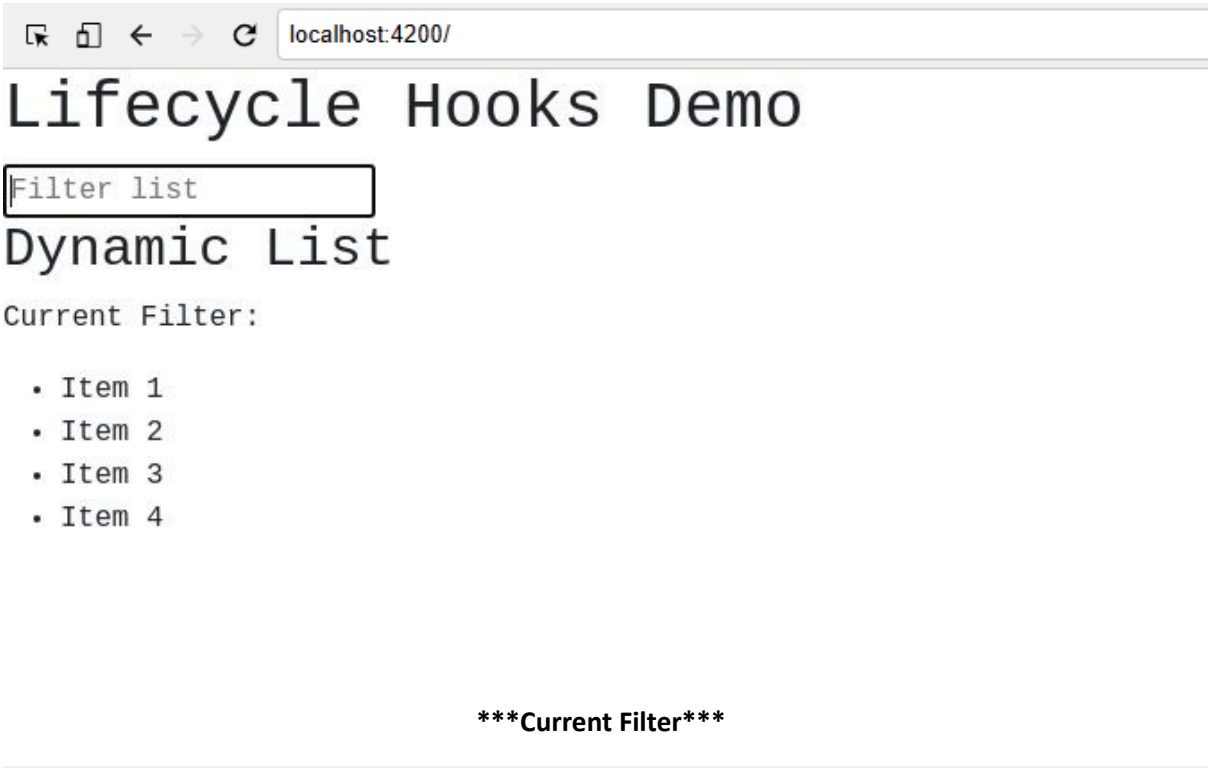
The application should:

- Initialize a predefined list of items when the component is loaded.
- Display a filter input field to update the list display dynamically.
- Log messages on component initialization and destruction using `ngOnInit` and `ngOnDestroy`.
- Pass filter data from parent to child component.

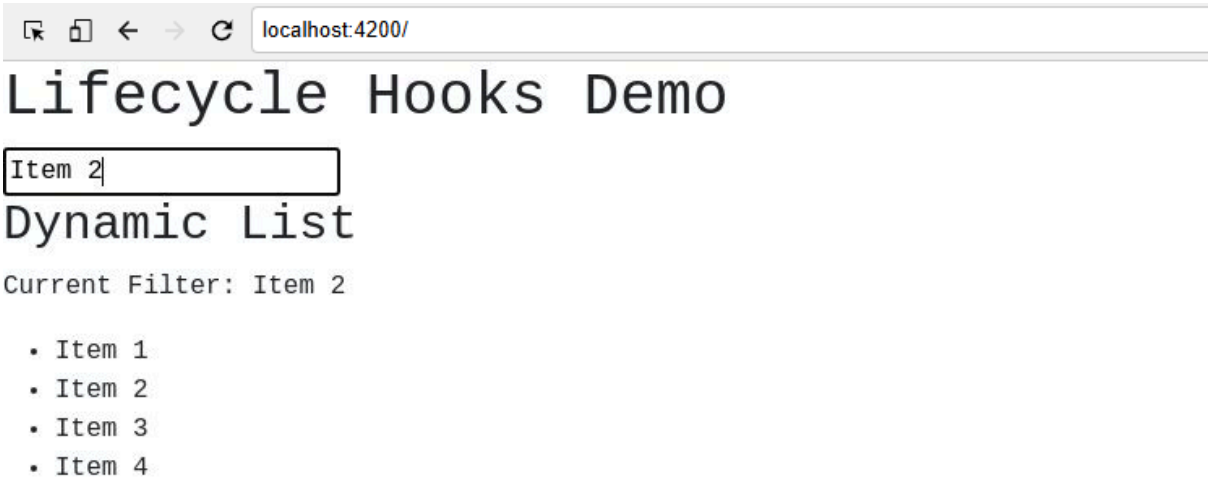
3 Proposed Lifecycle Hooks for Component Initialization and Destruction Application Wireframe

UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 ScreenShots



Current Filter



Among the given values its not filtering any

4 BUSINESS-REQUIREMENT:

As an application developer, develop the Lifecycle Hooks for Component Initialization and Destruction (Single Page App) with below guidelines:

User Story #	User Story Name	User Story

US_01	Welcome Page	As a user I should be able to visit the welcome page as the default page. Acceptance criteria: **AppComponent:** 1. Display the heading **"Lifecycle Hooks Demo"** in an **h1** tag. Define a string property in the component class named **`filter`**. ● This will store the value entered in the input field. ● Initialize it as an empty string. Create a method named **`updateFilter(value: string)`**. This method should: ● Accept the latest string value from the input. ● Update the `filter` property with this value. Add a **text input field**: ● Bind it to the `filter` property using **two-way data binding** (`[(ngModel)]`). ● Listen for changes using `(ngModelChange)` and call the `updateFilter()` method with the new value. ● Add a placeholder like **"Filter list"** for user clarity. Include a child component named **`app-dynamic-list`**: ● Pass the `filter` value to it using **property binding** (`[filter]="filter"`). **DynamicListComponent:** 2. Display the heading **"Dynamic List"** in an **h2** tag. Import and implement the following Angular lifecycle interfaces: ● `OnInit` ● `OnChanges` ● `DoCheck` ● `OnDestroy` Inject a service named **`DynamicListService`** to fetch the data 3. Define an `@Input()` property named **`filter`**. 4. Type: `string` 5. This property will receive a filter value from the parent component. 6. Initialize it as an empty string (`''`).

		Define a property named **`items`**, which is an array of strings. ● This will store the list of items fetched from the service. **1. Constructor** ● Log a message like `"DynamicListComponent constructor"` to the console. ● Inject `DynamicListService` using dependency injection. **2. ngOnInit()** ● Log `"ngOnInit called"` to the console. ● Call a method (e.g., `getItems()`) from the injected service to retrieve a list of items. ● Subscribe to the observable and assign the data to the `items` array. ● Log the fetched data to the console. **3. ngOnChanges(changes: SimpleChanges)** ● Log `"ngOnChanges called"` and the `changes` object. ● Check if the `filter` input has changed. ○ If so, log the updated filter value. ● This hook runs whenever the parent changes the `filter` input. **4. ngDoCheck()** ● Log `"ngDoCheck called"` to the console. ● Use this for **custom change detection logic** if needed. **5. ngOnDestroy()** ● Log `"ngOnDestroy called"` to the console. ● If you used a subscription in `ngOnInit()`, make sure to **unsubscribe** from it here to prevent memory leaks. Show the current value of the `filter` input using interpolation (`{{ filter }}`). Display the list of items using `*ngFor`: ● Loop over the `items` array and show each item inside an `<li>` tag. **** Kindly refer to the screenshots for any clarifications. ****

5 CONSTRAINTS

1. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

6 MANDATORY ASSESSMENT GUIDELINES

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser
6. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min.**
   b. **npm run start -> To compile and deploy the project in browser. You can press the <Ctrl> key while clicking on localhost:4200 to open the project in the browser -> takes 2 to 3 min.**
   c. **npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**
7. **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.**