QUESTION 1:

Smart Office Room Management Application

Create an Angular application to manage office rooms, with functionalities for adding, listing, and deleting rooms. Use HttpClient for mock data communication with a JSON server.

Components:

AddRoomComponent (/add-room)

Form Fields:

name (text input, placeholder: "Enter room name")

floor (select input, options: "1st Floor", "2nd Floor", "3rd Floor", "4th Floor")

capacity (number input, placeholder: "Enter room capacity")

lastInspection (date input, placeholder: "Enter last inspection date")

status (radio buttons: "Available" and "Under Maintenance")

Submit Button:

Text: "Add Room"

Use FormGroup for form validation to ensure all fields are filled correctly.

RoomListComponent (/rooms)

Display a table of rooms with headers:

Name, Floor, Capacity, Last Inspection, Status, Action

Functionalities:

Delete: Implement a delete button for each room entry to remove it from the list.

Sort: Add buttons to sort the table by name, floor, and capacity.

Model (room.model.ts):

Define the following model:

```
export interface Room {
  id?: number;
  name: string;
  floor: string;
  capacity: number;
  lastInspection: Date;
  status: string;
}
```

App Component (app.component.ts)

Ensure the title property is set to "Smart-Office-Room-Management" for testing.

Service (room.service.ts):

Define methods in the service to handle HTTP requests:

getRooms(): GET request

addRoom(): POST request

deleteRoom(): DELETE request

Use a variable named backendUrl to store the API URL to align with test case requirements.

Requirements:

Inputs: Use specified types and placeholders for each input in the AddRoomComponent form.

Sorting: Implement sortRooms() method in the RoomListComponent to sort by name, floor, or capacity.

Device List Page Structure (room-list.component.html):

Sort Buttons: Located above the table to allow sorting by name, floor, or capacity.

Table Headers:

Name, Floor, Capacity, Last Inspection, Status, Location, Action

Table Body:

Each row displays room information. Conditionally style the Status field:

Red if status is "Under Maintenance"

Green if status is "Available"

Delete Action: The last column includes a "Delete" button that calls deleteRoom(room.id) to remove the room.

Room List Page:

Display a list of rooms in a table format with columns for Name, Floor, Capacity, Last Inspection, Status, and an Action column with a Delete button for each row. Sort buttons for Name, Floor, and Capacity appear above the table.

Add Room Page:

A form with inputs for room details as described, with an "Add Room" button.

QUESTION 2:

Product Inventory Management System

Create an Angular application to manage products using HttpClient for mock data communication.

Components:

1. AddProductComponent (/add-product)

Form with fields:

productName (text input, placeholder: "Enter product name")

category (dropdown, options: "Electronics", "Clothing", "Groceries", "Home Appliances")

price (number input, placeholder: "Enter price")

stockQuantity (number input, placeholder: "Enter stock quantity")

supplier (text input, placeholder: "Enter supplier name")

manufactureDate (date input)

Submit button with exact text: "Add Product"

Use FormGroup for form validation with the following validators:

productName: Required, minLength(3), maxLength(100)

category: Required

price: Required, min(1)

stockQuantity: Required, min(1), max(10000)

supplier: Required, minLength(3), maxLength(50)

manufactureDate: Required

2. ProductListComponent (/products)

Display a table with headers:

Product Name, Category, Price, Stock Quantity, Supplier, Manufacture Date, Action

Implement edit functionality

Add sort buttons for price

Implement filter functionality for product name and category

3. EditProductComponent (/edit-product/:id)

Similar form to AddProductComponent, but pre-populated with product data

Submit button with exact text: "Update Product"

4. App Component (app.component.ts)

Ensure title property is set to "Product-Inventory-System" to pass test cases.

Model (product.model.ts):

```
export interface Product {
  id?: number;
  productName: string;
  category: string;
  price: number;
  stockQuantity: number;
  supplier: string;
  manufactureDate: Date;
}
```

Service (product.service.ts):

getProducts(): GET request

addProduct(): POST request

updateProduct(): PUT request

Ensure all input types and placeholders are correctly implemented in the AddProductComponent and EditProductComponent templates.

The API URL should be stored in a variable named backendUrl in product.service.ts to align with the test case requirements.

Additional Information:

Use component names, class names, and method names as specified to avoid errors.

Import and export of components, model, and services file should be included.

Use placeholders and text content as mentioned.

QUESTION 3:

Event Management Application

Create an Angular application to manage event schedules using HttpClient for mock data communication.

Components:

AddEventComponent (/add)

Form with fields:

title (text input, placeholder: "Enter event title")

date (date input, placeholder: "Select event date")

startTime (time input, placeholder: "Enter start time")

endTime (time input, placeholder: "Enter end time")

location (text input, placeholder: "Enter location")

maxAttendees (number input, placeholder: "Enter maximum attendees")

Submit button with the exact text: "Add Event"

Use FormGroup for form validation

EventListComponent (/events)

Display table with headers: Title, Date, Start Time, End Time, Location, Max Attendees, Action

Implement delete functionality

Add a sort button for maxAttendees

Model (event.model.ts)

```
export interface Event {

  id?: number;

  title: string;
  date: string;

  startTime: string;

  endTime: string;

  location: string;

  maxAttendees: number;

}
```

Service (event.service.ts)

Define the following methods:

getEvents(): GET request to fetch all events

addEvent(event: Event): POST request to add a new event

deleteEvent(id: number): DELETE request to remove an event by ID

Additional Requirements

Ensure all input types and placeholders are correctly implemented in the AddEventComponent template.

Use the text content and placeholder names.

Import and export all required components, models, and services.

Sample Screenshots:

Add Event Page

Event List Page

Edit Event Page

QUESTION 4:

Book Management System Application

Create an Angular application to manage a collection of books using HttpClient for mock data communication.

Components:

AddBookComponent (/add-book)

Form with fields:

title: (text input, placeholder: "Enter book title")

author: (text input, placeholder: "Enter author name")

genre: (text input, placeholder: "Enter genre")

publishedDate: (date input)

pages: (number input, placeholder: "Enter number of pages")

Submit button with exact text: "Add Book"

Use FormGroup for form validation with the following validators:

title: Required, minLength(1), maxLength(150)

author: Required, minLength(1), maxLength(70)

genre: Required, minLength(1), maxLength(50)

publishedDate: Required

pages: Required, min(1), max(2000)

BookListComponent (/books)

Display a table with headers:

Title, Author, Genre, Published Date, Pages, Action

Implement edit functionality.

Add sort buttons for title, author, and pages.

Implement filter functionality for title and author.

EditBookComponent (/edit-book/:id)

Similar form to AddBookComponent, but pre-populated with book data.

Submit button with exact text: "Update Book"

App Component (app.component.ts)

Ensure title property is set to "Book-Management-System" to pass test cases.

Model (book.model.ts):

```
export interface Book {
  id?: number;
  title: string;
  author: string;
  genre: string;
  publishedDate: string;
  pages: number;
}
```

Service (book.service.ts):

getBooks(): GET request

addBook(): POST request

updateBook(): PUT request

Ensure all input types and placeholders are correctly implemented in the AddBookComponent and EditBookComponent templates. The API URL should be stored in a variable named backendUrl in the service to align with the test case requirements.

Sample Screenshots:

[Book List Page Screenshot]

[Add Book Page Screenshot]

[Edit Book Page Screenshot]

[Sort by Pages Screenshots]

Book List HTML (book-list.component.html):

This component's template displays a dynamic table of books, structured as follows:

Filter Input: At the top of the table, an input field is provided to filter books by title or author.

Sort Buttons: Below the filter input, buttons are provided to trigger the sortBooks() method, allowing users to sort by title, author, or pages in ascending or descending order.

Table Structure:

Table Headers (<thead>): The headers are defined within a <thead> section, containing the columns mentioned above.

Table Body (<tbody>): The body of the table is defined within a <tbody> section, which dynamically generates rows for each book in the filteredBooks array using the ngFor directive.

QUESTION 5:
Fitness Class Management System

Create an Angular application to manage fitness classes using HttpClient for mock data communication.

Components:

1. AddClassComponent (/add-class)

Form with fields:

name (text input, placeholder: "Enter class name")

description (textarea, placeholder: "Enter class description")

instructor (text input, placeholder: "Enter instructor name")

date (date input)

time (time input)

duration (number input, placeholder: "Enter duration in minutes")

capacity (number input, placeholder: "Enter class capacity")

availableSlots (number input, placeholder: "Enter available slots")

Submit button with exact text: "Add Class"

Use FormGroup for form validation with the following validators:

name: Required, minLength(3), maxLength(100)

description: Required, minLength(10), maxLength(500)

instructor: Required, minLength(3), maxLength(50)

date: Required

time: Required

duration: Required, min(15), max(180)

capacity: Required, min(1), max(100)

availableSlots: Required, min(0), max(100)

2. ClassListComponent (/classes)

Display table with headers:

Name, Instructor, Date, Time, Duration, Available Slots, Action

Implement edit functionality

Add sort buttons for name, date, and availableSlots

Implement filter functionality for name and instructor

3. EditClassComponent (/edit-class/:id)

Similar form to AddClassComponent, but pre-populated with class data

Submit button with exact text: "Update Class"

Model (fitness-class.model.ts):

```
export interface FitnessClass {
  id?: number;
  name: string;
  description: string;
  instructor: string;
  date: string;
  time: string;
  duration: number;
  capacity: number;
  availableSlots: number;
}
```

Service (fitness.service.ts):

getClasses(): GET request

addClass(): POST request

updateClass(): PUT request

getClassById(): GET request

Ensure all input types and placeholders are correctly implemented in the AddClassComponent and EditClassComponent templates.

Sample Screenshots:

Fitness Class List Page:

[A table showing a list of fitness classes with columns for Name, Instructor, Date, Time, Duration, Available Slots, and an Edit button for each row. Sort buttons for Name, Date, and Available Slots are displayed above the table. A filter input is present above the table.]

Add Fitness Class Page:

[A form with input fields for Class Name, Description, Instructor, Date, Time, Duration, Capacity, and Available Slots. An "Add Class" button is at the bottom of the form.]

Edit Fitness Class Page:

[Similar to the Add Fitness Class page, but with pre-populated data and an "Update Class" button.]

Sort by Available Slots sample screenshots:

[Two screenshots showing the list sorted by available slots in ascending and descending order]

Class List HTML (class-list.component.html):

This component's template displays a dynamic table of fitness classes, structured as follows:

Filter Input: At the top of the table, an input field is provided to filter classes by name or instructor.

Sort Buttons: Below the filter input, buttons are provided to trigger the sortClasses() method, allowing users to sort by name, date, or available slots in ascending or descending order.

Table Structure:

The table is constructed using standard HTML table elements:

Table Headers (<thead>): The headers are defined within a <thead> section, containing the columns mentioned above.

Table Body (<tbody>): The body of the table is defined within a <tbody> section, which dynamically generates rows for each class in the filteredClasses array using the ngFor directive.

This structure provides a clear and interactive way for users to view, sort, filter, and manage their fitness classes.