We will use httpClient with our service.

This offloads data management from the component to a dedicated service and retrieves book data via HttpClient from a fake JSON server.

Steps:

1. **Set Up a Fake API:**
    - Use json-server with a `db.json` file containing your book data.
    - Run json-server to serve the data at a specific URL (e.g., `http://localhost:3000/books`).
2. **Book Inventory Service:**
    - Create a service (`BookInventoryService`) that uses Angular's HttpClient to GET and DELETE book data from the fake API.
3. **Book Inventory Component:**
    - Inject the service and HttpClientModule.
    - In `ngOnInit()`, subscribe to the service's `getInventory()` to retrieve data.
    - Use the service's `deleteBook()` method to remove books and refresh the list.
4. **Update main.ts:**
    - Add HttpClientModule providers.

1. Set Up a Fake JSON Server

You can use json-server to quickly simulate a REST API. Follow these steps:

a. Install json-server Globally (or as a Dev Dependency)

```
npm install -g json-server
```

Or install as a dev dependency:

```
npm install json-server --save-dev
```

b. Create a db.json File

Create a file named `db.json` in your project root with sample data. For example:

```
{
  "books": [
    {
      "ISBN": "978-1492056205",
      "title": "Angular Up & Running",
      "author": "Shyam Seshadri",
      "year": 2018,
      "price": 39.99,
      "featured": true,
      "coverImages": ["/assets/angular-up-and-running.png"]
    },
    {
      "ISBN": "978-1593279509",
      "title": "Eloquent JavaScript, 3rd Edition",
      "author": "Marijn Haverbeke",
      "year": 2018,
      "price": 29.99,
      "featured": false,
      "coverImages": ["/assets/eloquent-javascript.jpg"]
    },
    {
      "ISBN": "978-1491904244",
      "title": "You Don't Know JS Yet: Get Started",
      "author": "Kyle Simpson",
      "year": 2020,
      "price": 34.99,
      "featured": false,
      "coverImages": ["/assets/ydkjs-cover.jpg"]
    },
    {
      "ISBN": "978-1449331818",
      "title": "Learning JavaScript Design Patterns",
      "author": "Addy Osmani",
      "year": 2012,
      "price": 25.99,
      "featured": true,
      "coverImages": ["/assets/js-design-patterns.png"]
```

```
    }
  ]
}
```

c. Run the JSON Server

Start json-server to serve data from `db.json`:

```
json-server --watch db.json
```

By default, the server runs on http://localhost:3000, and your books will be available at `http://localhost:3000/books`.

2. Create/Update the Book Inventory Service

Create a service that uses HttpClient to fetch the book data.

a. Generate the Service (Optional)

If you haven't already, generate the service:

```
ng generate service book-inventory/book-inventory
```

b. Update the Service Code

File: `src/app/book-inventory/book-inventory.service.ts`

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Book } from '../book';

@Injectable({
  providedIn: 'root'
})
export class BookInventoryService {
  private apiUrl = 'http://localhost:3000/books'; // URL to
json-server

  constructor(private http: HttpClient) { }
```

```
  // Retrieve the inventory as an observable
  getInventory(): Observable<Book[]> {
    return this.http.get<Book[]>(this.apiUrl);
  }


}
```

Notes:

- We define an `apiUrl` pointing to your fake API.
- `getInventory()` returns an `Observable<Book[]>`.

3. Update the Book Inventory Component

Modify your component to use the service instead of hard-coded data.

a. Import HttpClientModule

Since HttpClient is required, ensure that `HttpClientModule` is imported in your bootstrap (for standalone components, include it in your `main.ts` or in your component's imports). For a standalone component, update the imports array.

b. Update the Component Code

File: `src/app/book-inventory/book-inventory.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { Book } from '../book';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { HoverHighlightDirective } from './hover-
highlight.directive';
import { BookFilterPipe } from '../book-filter.pipe';
import { BookInventoryService } from './book-
inventory.service';

@Component({
  selector: 'app-book-inventory',
  standalone: true,
```

```
  imports: [CommonModule, FormsModule, HttpClientModule,
HoverHighlightDirective, BookFilterPipe],
  templateUrl: './book-inventory.component.html',
  styleUrls: ['./book-inventory.component.css']
})
export class BookInventoryComponent implements OnInit {
  currentDate: Date = new Date();
  searchTerm: string = '';
  inventory: Book[] = [];

  constructor(private bookService: BookInventoryService) {}

  ngOnInit(): void {
    this.getBooks();
  }

  // Retrieve the inventory from the service
  getBooks(): void {
    this.bookService.getInventory().subscribe({
      next: (books: Book[]) => this.inventory = books,
      error: (err) => console.error('Error fetching
inventory', err)
    });
  }

  trackByISBN(index: number, book: Book): string {
    return book.ISBN;
  }

}
```

Explanation:

- **HttpClientModule:** Added to `imports` so HttpClient works in the standalone component.
- **Service Injection:** Inject `BookInventoryService` in the constructor.
- **Retrieving Data:** In `ngOnInit()`, call `getBooks()` to subscribe to the observable returned by the service.
- **Deleting a Book:** Call the service's `deleteBook()` method and refresh the inventory afterward.

4. Add the providers to main.ts:

```
// src/main.ts

import { bootstrapApplication } from '@angular/platform-
browser';

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

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

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

import { AppComponent } from './app/app.component';

import { appRoutes } from './app/app.routes';



bootstrapApplication(AppComponent, {

  providers: [

    provideRouter(appRoutes),

    importProvidersFrom(HttpClientModule) // This ensures
HttpClient is provided globally

  ]

}).catch(err => console.error(err));
```