Angular CRUD Application Explained (Layman's Terms)

This document explains a simple Angular CRUD (Create, Read, Update, Delete) application with each component clearly described, along with Observables, subscribe(), and RxJS usage.

# 1. Define the Book Model

Create a file named book.model.ts to define the structure of a book. This acts like a blueprint.

export interface Book {  
 id: number;  
 title: string;  
 author: string;  
 price: number;  
}

# 2. Create Book Service (book.service.ts)

This service handles all HTTP requests (GET, POST, PUT, DELETE) to the backend API.

@Injectable({ providedIn: 'root' })  
export class BookService {  
 private apiUrl = 'http://localhost:8080/api/books';  
 constructor(private http: HttpClient) {}  
  
 getBooks(): Observable<Book[]> {  
 return this.http.get<Book[]>(this.apiUrl);  
 }  
  
 addBook(book: Book): Observable<Book> {  
 return this.http.post<Book>(this.apiUrl, book);  
 }  
  
 updateBook(book: Book): Observable<Book> {  
 return this.http.put<Book>(`${this.apiUrl}/${book.id}`, book);  
 }  
  
 deleteBook(id: number): Observable<void> {  
 return this.http.delete<void>(`${this.apiUrl}/${id}`);  
 }  
}

# 3. Display Book List (book-list.component.ts)

This component displays the list of books and handles deletion.

export class BookListComponent implements OnInit {  
 books: Book[] = [];  
 constructor(private bookService: BookService) {}  
  
 ngOnInit() {  
 this.getBooks();  
 }  
  
 getBooks() {  
 this.bookService.getBooks().subscribe({  
 next: data => this.books = data,  
 error: err => console.error('Error', err)  
 });  
 }  
  
 deleteBook(id: number) {  
 this.bookService.deleteBook(id).subscribe(() => this.getBooks());  
 }  
}

# 4. Book Form to Add/Edit (book-form.component.ts)

Used to add or edit books.

export class BookFormComponent implements OnInit {  
 @Input() book: Book = { id: 0, title: '', author: '', price: 0 };  
 constructor(private bookService: BookService) {}  
 ngOnInit() {}  
  
 saveBook() {  
 if (this.book.id) {  
 this.bookService.updateBook(this.book).subscribe();  
 } else {  
 this.bookService.addBook(this.book).subscribe();  
 }  
 }  
}

# 5. Example HTML for List and Form

These are the UI components written in Angular HTML templates.

<!-- book-list.component.html -->  
<table>  
 <tr \*ngFor="let book of books">  
 <td>{{ book.title }}</td>  
 <td>{{ book.author }}</td>  
 <td>{{ book.price }}</td>  
 <td>  
 <button (click)="editBook(book)">Edit</button>  
 <button (click)="deleteBook(book.id)">Delete</button>  
 </td>  
 </tr>  
</table>  
  
<!-- book-form.component.html -->  
<form (ngSubmit)="saveBook()">  
 <input [(ngModel)]="book.title" name="title" required>  
 <input [(ngModel)]="book.author" name="author" required>  
 <input [(ngModel)]="book.price" name="price" required>  
 <button type="submit">Save</button>  
</form>

# 6. Observables and subscribe() Explained

Observables are data streams. Components use subscribe() to respond to these streams.

this.bookService.getBooks().subscribe({  
 next: data => this.books = data,  
 error: err => console.error('Error:', err),  
 complete: () => console.log('Completed')  
});

# 7. RxJS Operators (Used in Angular)

- map(): Transform the data (e.g., extract field)  
- filter(): Filter items (e.g., price > 100)  
- tap(): Side effect (e.g., log)  
- catchError(): Handle errors  
- switchMap(): Cancel previous call and switch to latest