Implement CRUD Methods Using RxJS Operators

4.1 GET – Retrieve All Books

Implement a method to get the list of books:

```
getInventory(): Observable<Book[]> {
  return this.http.get<Book[]>(this.apiUrl).pipe(
    retry(2), // Retry the HTTP GET request up to 2 times
in case of transient errors
    tap(books => console.log('Fetched books:', books)),
    catchError(this.handleError<Book[]>('getInventory',
[]))
  );
}
```

Explanation:

- **http.get<Book[]>(this.apiUrl):** Sends a GET request and expects an array of Book objects.
- **retry(2):** Automatically retries the request if it fails.
- **tap(...):** Logs the retrieved books.
- **catchError(...):** Catches errors and passes them to a generic error handler.

4.2 POST – Create a New Book

Implement a method to add a new book. If the new book does not include an `id`, assign it using the ISBN:

```
createBook(book: Book): Observable<Book> {
  if (!book.id) {
    book.id = book.ISBN; // Set id to ISBN if not provided
  }
  return this.http.post<Book>(this.apiUrl, book).pipe(
    tap(newBook => console.log('Created book:', newBook)),
    catchError(this.handleError<Book>('createBook'))
  );
}
```

Explanation:

- **http.post<Book>(this.apiUrl, book):** Sends a POST request with the book data.
- **tap:** Logs the newly created book.
- **catchError:** Handles any errors.

4.3 PUT – Update an Existing Book

Implement a method to update an existing book. In this example, we're updating using the ISBN in the URL:

```
updateBook(book: Book): Observable<Book> {
  return this.http.put<Book>(`${this.apiUrl}/${book.ISBN}`,
book).pipe(
    tap(updatedBook => console.log('Updated book:',
updatedBook)),
    catchError(this.handleError<Book>('updateBook'))
  );
}
```

Explanation:

- **http.put<Book>(`${this.apiUrl}/${book.ISBN}`, book):** Sends a PUT request to update the book at the URL corresponding to its ISBN.
- **tap:** Logs the updated book.
- **catchError:** Handles errors.

4.4 DELETE – Remove a Book

Implement a method to delete a book. Note that json-server expects an `id` property, so we use `book.id`:

```
deleteBook(book: Book): Observable<Book> {
  const url = `${this.apiUrl}/${book.id}`;
  return this.http.delete<Book>(url).pipe(
    tap(() => console.log(`Deleted book with id: ${book.id}
`)),
    catchError(this.handleError<Book>('deleteBook'))
  );
}
```

Explanation:

- **http.delete<Book>(url):** Sends a DELETE request to remove the book.
- **tap:** Logs the deletion.
- **catchError:** Catches and handles errors.

Step 5: Create a Generic Error Handler

Implement a helper method to handle HTTP errors:

```
private handleError<T>(operation = 'operation', result?: T)
{
  return (error: HttpErrorResponse): Observable<T> => {
```

```
    // Log the error to the console (or send to remote
logging)
    console.error(`${operation} failed: ${error.message}`);
    // Return an observable with a user-facing error
message
    return throwError(() => error);
  };
}
```

Explanation:

- **handleError:**
  This method logs the error and returns an observable that emits the error. It's used by catchError in each HTTP method.

Complete Service Code

Putting it all together, your `book-inventory.service.ts` should look like this:

```
// src/app/book-inventory/book-inventory.service.ts
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/
common/http';
import { Observable, throwError } from 'rxjs';
import { Book } from '../book';
import { catchError, retry, tap } from 'rxjs/operators';

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

  constructor(private http: HttpClient) { }

  getInventory(): Observable<Book[]> {
    return this.http.get<Book[]>(this.apiUrl).pipe(
      retry(2),
      tap(books => console.log('Fetched books:', books)),
      catchError(this.handleError<Book[]>('getInventory',
[]))
```

```
    );
  }

  getBook(isbn: string): Observable<Book> {
    return this.http.get<Book>(`${this.apiUrl}/${isbn}
`).pipe(
      retry(2),
      tap(book => console.log(`Fetched book with ISBN $
{isbn}:`, book)),
      catchError(this.handleError<Book>('getBook'))
    );
  }

  createBook(book: Book): Observable<Book> {
    if (!book.id) {
      book.id = book.ISBN;
    }
    return this.http.post<Book>(this.apiUrl, book).pipe(
      tap(newBook => console.log('Created book:',
newBook)),
      catchError(this.handleError<Book>('createBook'))
    );
  }

  updateBook(book: Book): Observable<Book> {
    return this.http.put<Book>(`${this.apiUrl}/${book.ISBN}
`, book).pipe(
      tap(updatedBook => console.log('Updated book:',
updatedBook)),
      catchError(this.handleError<Book>('updateBook'))
    );
  }

  deleteBook(book: Book): Observable<Book> {
    const url = `${this.apiUrl}/${book.id}`;
    return this.http.delete<Book>(url).pipe(
      tap(() => console.log(`Deleted book with id: $
{book.id}`)),
      catchError(this.handleError<Book>('deleteBook'))
    );
```

```
  }

  private handleError<T>(operation = 'operation', result?:
T) {
    return (error: HttpErrorResponse): Observable<T> => {
      console.error(`${operation} failed: ${error.message}
`);
      return throwError(() => error);
    };
  }
}
```