

PROJECT 4 TASK 2 – BOOK INFORMATION APP

API Name: OpenLibrary API

API Documentation URL: <https://openlibrary.org/dev/docs/api/search>

Description:

My application allows users to search for books by their ISBN and displays detailed information about the book. The app consists of an Android mobile application that communicates with a web service to retrieve and display book information.

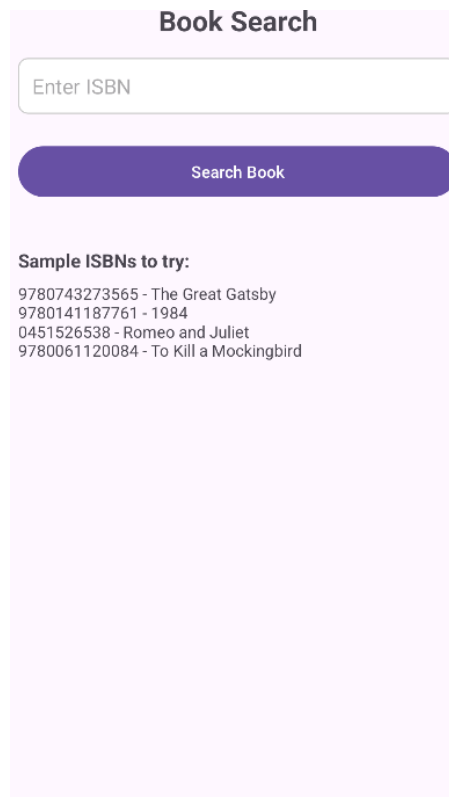
Here is how my application meets the task requirements:

1. Implement a native Android application

The name of my native Android application project in Android Studio is: Project4Task2Android

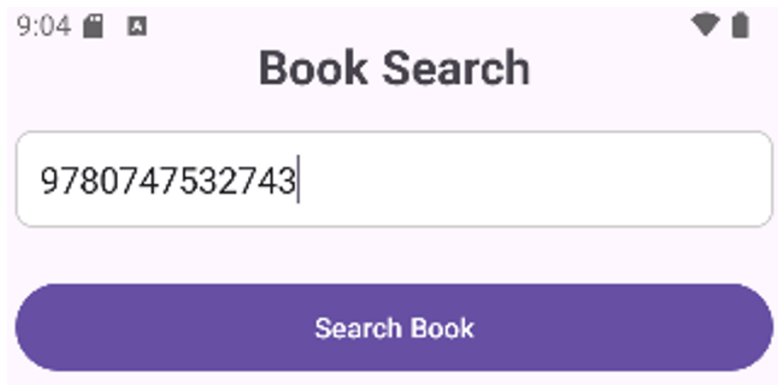
a. Has at least three different kinds of views in your Layout

My application uses TextView, EditText, Button, and LinearLayout.



The screenshot shows a mobile application interface titled "Book Search". It features a light purple background. At the top, there is a white rectangular input field with the placeholder text "Enter ISBN". Below this field is a rounded purple button with the text "Search Book" in white. Further down, the text "Sample ISBNs to try:" is displayed in a bold font. Below this text, four ISBNs are listed, each followed by the book title: "9780743273565 - The Great Gatsby", "9780141187761 - 1984", "0451526538 - Romeo and Juliet", and "9780061120084 - To Kill a Mockingbird".

b. Requires input from the user

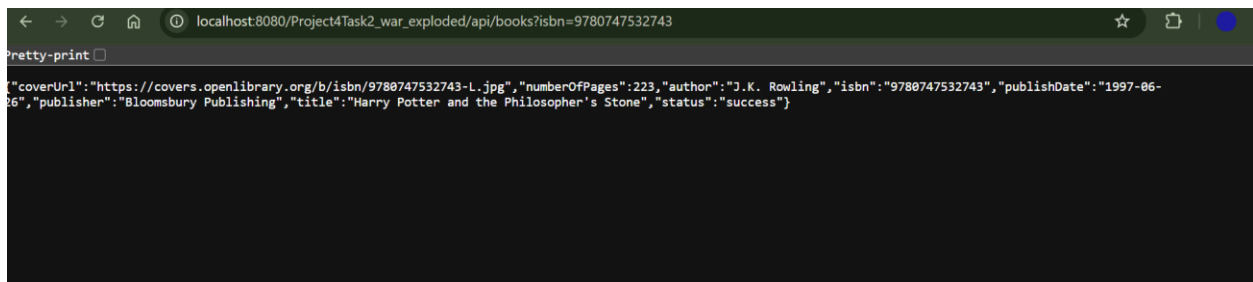


c. Makes an HTTP request (using an appropriate HTTP method) to your web service

My application does an HTTP GET request in `HttpHandler.java`. The HTTP request is:

"http://10.0.2.2:8080/Project4Task2_war_exploded/api/books?isbn="+isbn where isbn is the user's input.

The search method makes this request of my web application, parses the returned JSON, and displays the book information to the user.



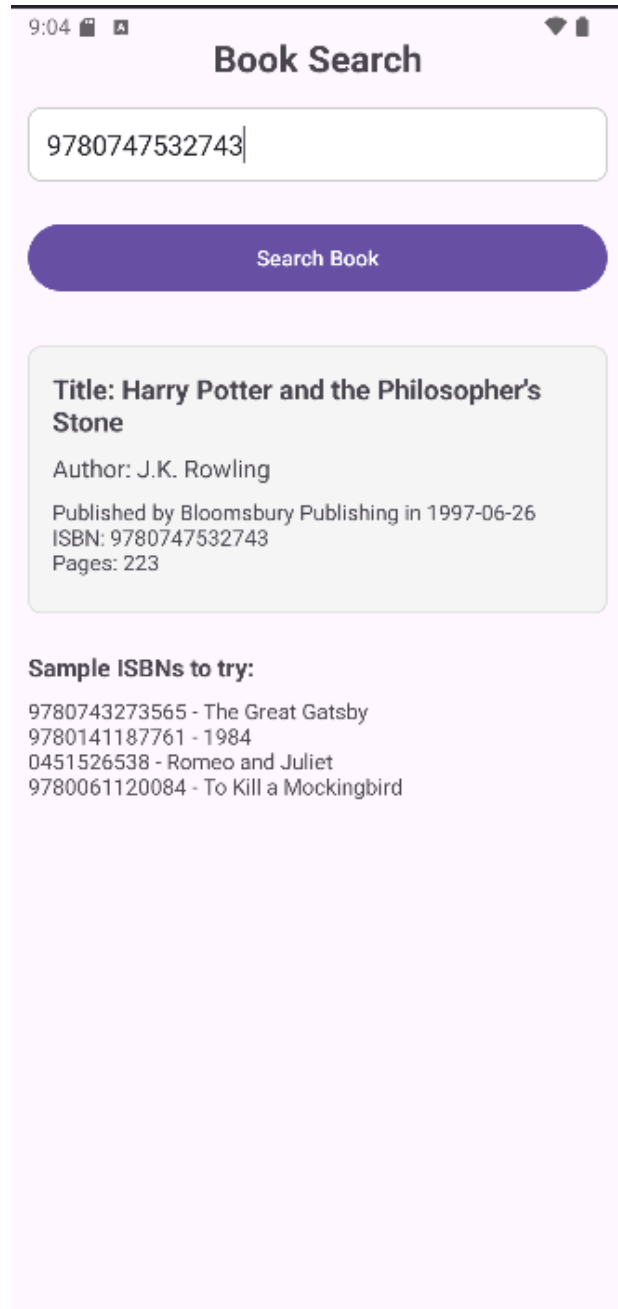
d. Receives and parses an XML or JSON formatted reply from the web service

An example of the JSON reply is:

```
{  
  "status": "success",  
  "isbn": "9780451524935",  
  "title": "1984",  
  "author": "George Orwell",  
  "publishDate": "1949",  
  "publisher": "Signet Classics",  
  "numberOfPages": 328,  
  "coverUrl": "https://covers.openlibrary.org/b/id/8575583-M.jpg"  
}
```

e. Displays new information to the user

Here is the screenshot after the book information has been returned:



f. Is repeatable (I.e. the user can repeatedly reuse the application without restarting it.)

The user can type in another ISBN and hit the "Search Book" button. The app provides sample ISBNs to try, making it easy for users to test different searches.

2. Implement a web application, deployed to GitHub Codespaces

I implemented and deployed my web service using GitHub Codespaces.

The project database name is Project4Task2.

The collection name is bookdata.

a. Using an HttpServlet to implement a simple (can be a single path) API

In my web app project:

- Model: LogEntry.java
- Controller: BookServlet.java
- View: Dashboard.jsp

b. Receives an HTTP request from the native Android application

BookServlet.java receives the HTTP GET request with the argument "isbn". It passes this search string to the model to retrieve book data.

c. Executes business logic appropriate to your application

BookServlet.java processes the ISBN parameter and uses the fetchBookData method to retrieve book information. It then formats a response to send back to the Android application.

d. Replies to the Android application with an XML or JSON formatted response.

The response to the Android app is in JSON format:

```
{  
  "status": "success",  
  "isbn": "9780451524935",  
  "title": "1984",  
  "author": "George Orwell",  
  "publishDate": "1949",  
  "publisher": "Signet Classics",  
  "numberOfPages": 328,  
  "coverUrl": "https://covers.openlibrary.org/b/id/8575583-M.jpg"  
}
```

3. Handle error conditions

My application handles several error conditions:

- Missing ISBN parameter: Returns an error message to the user
- Book not found: Displays a proper error message
- Network errors: Informs the user about connection issues
- JSON parsing errors: Handles parsing failures gracefully

All errors are displayed to the user with clear and helpful messages, making it easy to understand what went wrong.

4. Log useful information

I log the following information for each request:

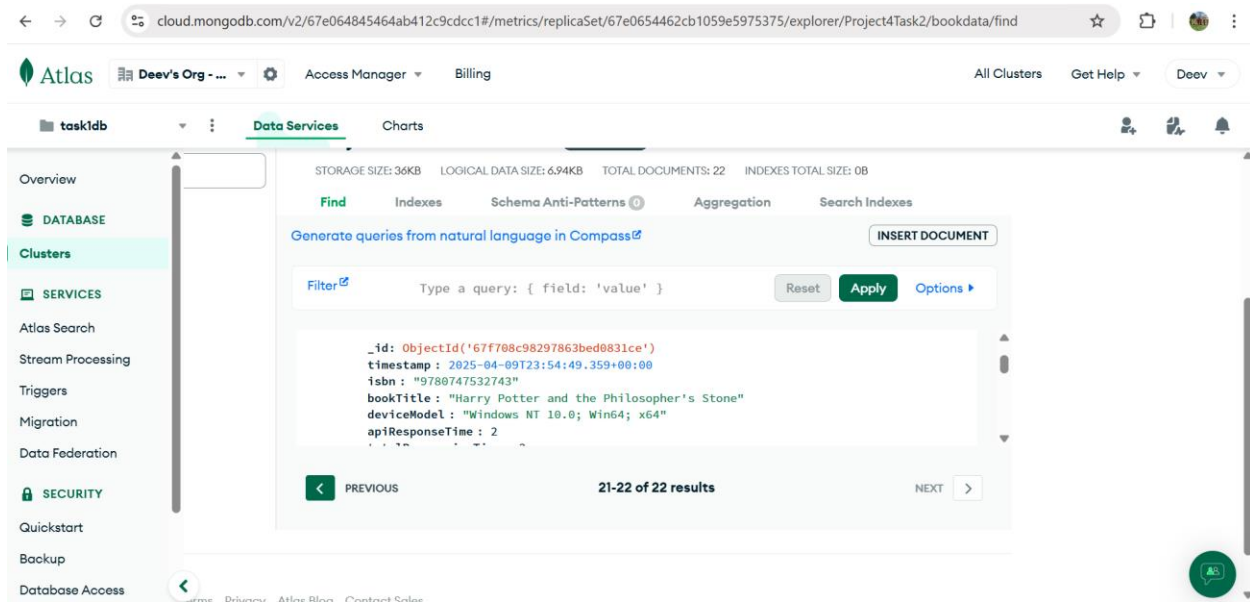
- Request timestamp: When the request was received
- ISBN searched: The parameter provided by the user
- Book title: The result returned (if successful)
- Device model: Extracted from user agent
- API request/response timestamps: To measure API performance
- API success/failure: Whether the request was successful
- Client IP address: For tracking user locations
- User agent: For device analytics
- API response time: To monitor performance
- Total processing time: End-to-end request handling time

This information is valuable for monitoring usage patterns, identifying performance issues, and troubleshooting errors.

5. Store the log information in a database

I'm using MongoDB Atlas as my database. Here's my connection string with the three shards:
mongodb+srv://deevp:Patel1234@task1db.h914b.mongodb.net/?retryWrites=true&w=majority&appName=task1db

The MongoClient class manages all database interactions, including logging request information and retrieving analytics data for the dashboard.



6. Display operations analytics and full logs on a web-based dashboard

The dashboard.jsp page provides a comprehensive view of the application's usage, including:

- Top searched ISBNs
- Average API response time
- Top device models used to access the service
- A full log of all requests with timestamps, results, and performance metrics

Book API Operations Dashboard

Refresh Data

Top Searched ISBNs

ISBN	Count
9780451524935	5
9780061120084	3
9780142437247	2

API Performance

Average API Response Time: 85.50 ms

Top Device Models

Device Model	Count
iPhone 14	8
Samsung Galaxy S21	6
Google Pixel 7	4

Request Logs

Timestamp	ISBN	Book Title	Device Model	API Response Time	Total Processing Time	Status	Client IP
2025-04-10 00:41:55	9780747532743	Harry Potter and the Philosopher's Stone	Android 16	114.0 ms	120.0 ms	Success	127.0.0.1
2025-04-10 00:41:13	9780061120084	N/A	Android 16	107.0 ms	109.0 ms	Failed	127.0.0.1
2025-04-10 00:40:28	9780141187781	N/A	Android 16	114.0 ms	115.0 ms	Failed	127.0.0.1
2025-04-10 00:40:03	9780743273565	N/A	Android 16	111.0 ms	111.0 ms	Failed	127.0.0.1
2025-04-10 00:39:04	9780060850524	N/A	Android 16	113.0 ms	114.0 ms	Failed	127.0.0.1
2025-04-10 00:38:44	9780747532743	Harry Potter and the Philosopher's Stone	Android 16	115.0 ms	125.0 ms	Success	127.0.0.1
2025-04-10 00:37:58	0451528538	N/A	Android 16	113.0 ms	128.0 ms	Failed	127.0.0.1
2025-04-10 00:34:48	9781400033416	N/A	Windows Desktop	107.0 ms	108.0 ms	Failed	0.0.0.0.0.0.1
2025-04-10 00:32:49	9780060850524	N/A	Windows Desktop	112.0 ms	112.0 ms	Failed	0.0.0.0.0.0.1
2025-04-10 00:32:33	0451528538	N/A	Windows Desktop	108.0 ms	108.0 ms	Failed	0.0.0.0.0.0.1
2025-04-10 00:32:05	9780451524935	1984	Windows Desktop	105.0 ms	105.0 ms	Success	0.0.0.0.0.0.1
2025-04-10 00:31:31	9780747532743	Harry Potter and the Philosopher's Stone	Windows Desktop	110.0 ms	112.0 ms	Success	0.0.0.0.0.0.1
2025-04-10 19:24:19	N/A	N/A	Windows NT 10.0; Win64; x64	0.0 ms	0.0 ms	Failed	0.0.0.0.0.0.1
2025-04-10 19:24:19	9780747532743	Harry Potter and the Philosopher's Stone	Linux; U; Android 16; sdx_gphone64_x86_64 Build/PP22.250221.010	11.0 ms	139.0 ms	Failed	127.0.0.1

The dashboard is accessible through the web application and automatically refreshes to show the latest data.