**Project 4 Task 2 Writeup**

**Student Information**

* **Jung Ho Park, junghop**

**1. Native Android Application**

**Description**

The Book Search Android application allows users to search for books by entering keywords, titles, or author names. The app presents search results in a clean list format showing the book title, author, and publication year for each result.

**Implementation Details**

The app implements the required views:

1. **EditText** - Used for the search input field where users enter their search terms
2. **Button** - The search button that initiates the search operation
3. **ListView** - Displays the search results in a scrollable list
4. **TextView** - Shows status messages and result counts

The application makes HTTP GET requests to my web service deployed on GitHub Codespaces. These requests are handled by a background thread using Android's AsyncTask to ensure the UI remains responsive. The app parses the JSON response from the web service and displays formatted results to the user. Users can perform multiple searches without restarting the application.

**Screenshots**

[Note: Insert screenshots of your Android app here]

**2. Web Service Implementation**

**Description**

My web service acts as a middleware layer between the Android app and the Open Library API. It's implemented as a Java servlet that handles requests from the mobile app, retrieves data from the Open Library API, and returns a simplified JSON response.

**API Details**

* **Endpoint:** /api/books
* **Method:** GET
* **Parameters:**
  + q - The search query (required)
  + limit - Maximum number of results to return (optional, default: 5)

**Business Logic**

The web service:

1. **Validates** the incoming query parameters
2. **Makes a request** to the Open Library API with the search term
3. **Processes** the response to extract relevant book information
4. **Creates** a simplified JSON response with only the needed information
5. **Logs** details about the request and response to MongoDB
6. **Returns** the formatted response to the Android app

**Screenshots**

[Note: Insert screenshots of your web service API responses here]

**3. MongoDB Database Integration**

**Description**

The application uses MongoDB Atlas as a cloud database to store logs and analytics data. Each interaction with the web service is recorded in the database for analysis.

**Data Model**

Each log entry includes:

1. **Timestamp** - When the request was received
2. **Search query** - What the user searched for
3. **Client IP address** - Where the request came from
4. **User agent** - Information about the mobile device
5. **API request duration** - How long the 3rd party API call took
6. **Number of results** - How many matches were found
7. **Response size** - Size of the data returned to the mobile app

**Screenshots**

[Note: Insert screenshots of your MongoDB collections/data here]

**6. Dashboard Implementation**

**Description**

The analytics dashboard provides a web interface to monitor application usage. It's implemented as a JSP page that displays real-time analytics and log information.

**Analytics**

The dashboard shows three key analytics:

1. **Top search queries** - Shows the most popular search terms
2. **Average API response times by hour** - Displays how quickly the API responds
3. **User agent statistics** - Shows which devices are using the app most

**Log Display**

All logs are displayed in a formatted HTML table showing:

* **Request timestamp**
* **Search query**
* **Client IP address**
* **Number of results found**
* **API response time**

**Screenshots**

[Note: Insert screenshots of your dashboard here]

**7. GitHub Codespaces Deployment**

**Description**

The web service is deployed to GitHub Codespaces using Docker containerization, making it accessible from anywhere.

**Setup Steps**

1. Created a **Dockerfile** for running Tomcat
2. Set up **.devcontainer.json** with the necessary configuration
3. Packaged the web application as **ROOT.war**
4. Deployed to **GitHub Codespaces**
5. Set port 8080 to **public visibility** for external access

**Screenshots**

[Note: Insert screenshots of your Codespaces deployment here]

**Error Handling**

The application handles various error conditions:

**Invalid Mobile App Input**

* The Android app **validates** that the search field is not empty
* Users receive a **toast message** if they try to search with an empty query

**Invalid Server-Side Input**

* The web service **validates** all parameters
* Returns appropriate **HTTP status codes** and error messages

**Network Failure Handling**

* The Android app displays **clear error messages** if the server is unreachable
* The web service **gracefully handles** connection failures to the Open Library API

**Third-Party API Unavailability**

* If the Open Library API is down, the web service returns a **user-friendly error message**
* The error is **logged to MongoDB** for monitoring

**Invalid API Data**

* The code **handles missing fields** in the API response
* Provides **default values** when data is incomplete or malformed