**Project 4 Task 1 – Spotify Artist Search App**

By Amos Xiao (AndrewID: dinghuax)

**Description:**

My application allows users to search for artists on Spotify, displaying their details and metrics. The application consists of a mobile interface, a web service that interacts with Spotify's API, and a dashboard showing usage analytics.

**1. Native Android Application Implementation**

Project name in Android Studio: SpotifySearchApp

**a. Views in Layout**

A black cell phone with a white screen

Description automatically generated

This layout includes six different types of Views (This satisfies the requirement of having at least three different kinds of Views in mine Layout.)

1. TextView (titleText and resultsTextView)
2. EditText (searchEditText)
3. Button (searchButton)
4. ProgressBar (loadingSpinner)
5. ImageView (artistImageView)
6. LinearLayout (the root view)

**b. User Input**

**A screenshot of a cell phone

Description automatically generated**

The application accepts artist names as search input from users.

**c. HTTP Request to Web Service**

My application does an HTTP GET request in SpotifySearchAsyncTask.java. The HTTP request is: "<https://turbo-halibut-q77v9v54775vf9xpp-8080.app.github.dev/SpotifySearchServlet?term=>" + searchTerm where term is the user's input search string for the artist name.

This URL is constructed from:

* Base URL: <https://turbo-halibut-q77v9v54775vf9xpp-8080.app.github.dev>
* Servlet path: /SpotifySearchServlet
* Query parameter: ?term= followed by the user's search input

**d. JSON Response Parsing**

An example of the JSON reply from SpotifySearchServlet for searching "Eagles" is:

{

"artists": {

"items": [

{

"name": "Eagles",

"popularity": 77,

"followers": {

"total": 11149592

}

},

{

"name": "Eagles Of Death Metal",

"popularity": 54,

"followers": {

"total": 552049

}

}

// ... more artists

],

"total": 10,

"limit": 10,

"offset": 0

}

}

**e. Information Display**

**A screenshot of a cell phone

Description automatically generated**

The app displays artist information including:

* Artist name
* Popularity score
* Follower count

**f. Repeatability**

Users can perform multiple searches without restarting the app, with each search updating the display with new artist information.

A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**

**2. Web Service Implementation**

The URL of my web service deployed to GitHub Codespaces is: <https://turbo-halibut-q77v9v54775vf9xpp-8080.app.github.dev>

The project repository is: <https://github.com/CMU-Heinz-95702/distributed-systems-project-04-amshaww23>

The project contains:

* Web Service: SpotifySearchServlet.java
* MongoDB integration for logging
* Integration with Spotify API
* Dashboard functionality for viewing logs and analytics

**a. API Implementation**

* Model: SpotifyAPIClient.java
* Controller: SpotifySearchServlet.java
* View: Dashboard interface via dashboard.jsp

**b. HTTP Request Handling**

SpotifySearchServlet.java processes GET requests with the "term" parameter containing the search query.

**c. Business Logic**

SpotifySearchServlet.java makes an HTTP request to:

* First authenticates with Spotify at: <https://accounts.spotify.com/api/token>
* Then searches artists using: <https://api.spotify.com/v1/search>

The business logic includes:

1. Getting Spotify access token using client credentials (CLIENT\_ID and CLIENT\_SECRET)
2. Making authenticated search request to Spotify API with the user's search term
3. Processing the JSON response which contains artist information
4. Logging request details and responses to MongoDB including:
   * Search term
   * Timestamp
   * User agent
   * Client IP
   * Device type
   * Response time
   * API status code
   * Result count
5. Formatting and returning the artist data to the Android application

**d. Response Format**

Replies to the Android application with a JSON formatted response

In SpotifySearchServlet.java, the response is formatted and sent as JSON:

java

response.setContentType("application/json");

response.getWriter().write(responseBody);

An example of the JSON response for a search of "Eagles" is:

json

Copy

{

"artists": {

"items": [

{

"name": "Eagles",

"popularity": 77,

"followers": {

"total": 11149592

}

},

{

"name": "Eagles Of Death Metal",

"popularity": 54,

"followers": {

"total": 552049

}

}

],

"total": 10,

"limit": 10,

"offset": 0

}

}

**4. Logging Information**

The following information is logged for each request:

1. Search term used
2. Timestamp of request
3. Device type/user agent
4. API response time
5. Success/failure status
6. Result count returned

**5. Database Storage**

MongoDB Atlas connection string:

Copy

mongodb+srv://[username]:[password]@cluster0.qy0vr.mongodb.net/project4

**6. Dashboard ImplementationA screenshot of a data analysis dashboard

Description automatically generated**

(the success rate is 60% and not looking good because I been debugging the code first but now it fixed and will success every time and I been too lazy to run It like 100 time to make it 95%+)

The dashboard is accessible at: https://turbo-halibut-q77v9v54775vf9xpp-8080.app.github.dev/

1. The dashboard displays key analytics:
   1. Overall Statistics:
   2. Total Searches: 26
   3. Average Response Time: 177.31 ms
   4. Success Rate: 65.4%
   5. Unique Users: 1
   6. Recent Searches Log Table showing:
   7. Search Term (e.g., "Eagles", "Justin Biber", "daniel caesar")
   8. Timestamp (Mon Nov 18 00:58:04 UTC 2024)
   9. Device Type (Other)
   10. API Status (Success/Failed)
   11. Results Found (e.g., 900, 812, 612)
   12. Response Time (e.g., 344 ms, 172 ms)
   13. The logs are formatted in a clear tabular format with:
   14. Column headers for each data point
   15. Alternating row colors for readability
   16. Chronological ordering (most recent first)
   17. Success/Failure status clearly indicated
   18. Response times in milliseconds
2. Log useful information I log the following information for each request to my web service:

* searchTerm: Track what artists users are searching for
* timestamp: Monitor when searches occur for usage patterns
* userAgent: Identify what devices/browsers are being used
* clientIP: Track unique users and geographic patterns
* deviceType: Understand mobile vs other usage
* requestHeaders: Debug any client-side issues
* responseTimeMs: Monitor API performance
* spotifyApiStatusCode: Track API reliability
* spotifyApiResponse: Store complete response for analysis
* successful: Monitor error rates
* resultCount: Track how many results each search returns
* apiLatency: Measure Spotify API response times
* responseToMobile: Store what was sent to mobile app
* mobileResponseStatus: Track mobile response success rates

1. Stored the log information in a database MongoDB Atlas connection string:

mongodb+srv://amshaww:Xdh20011123@cluster0.qy0vr.mongodb.net/project4?retryWrites=true&w=majority&tls=true&tlsAllowInvalidHostnames=true&tlsAllowInvalidCertificates=true

1. Display operations analytics and full logs on a web-based dashboard [You would insert a screenshot of your dashboard here showing:]

* Top searched artists
* Average API response times
* Success/error rates
* Recent search logs with timestamps
* Device type distribution
* Total number of searches