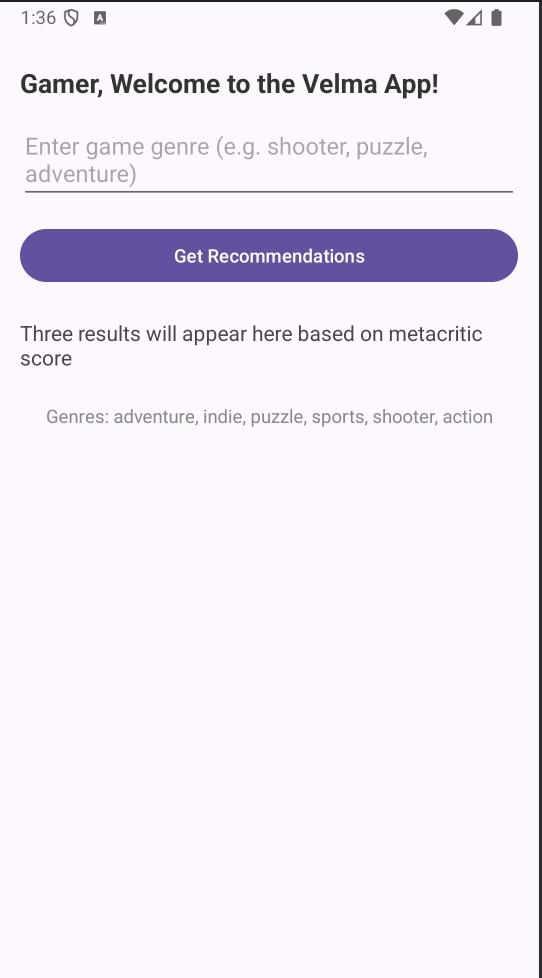
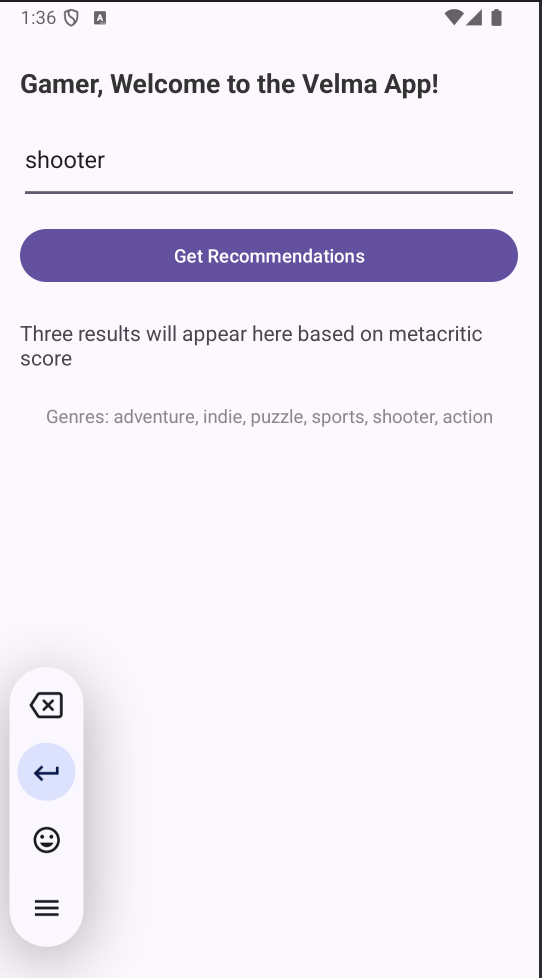
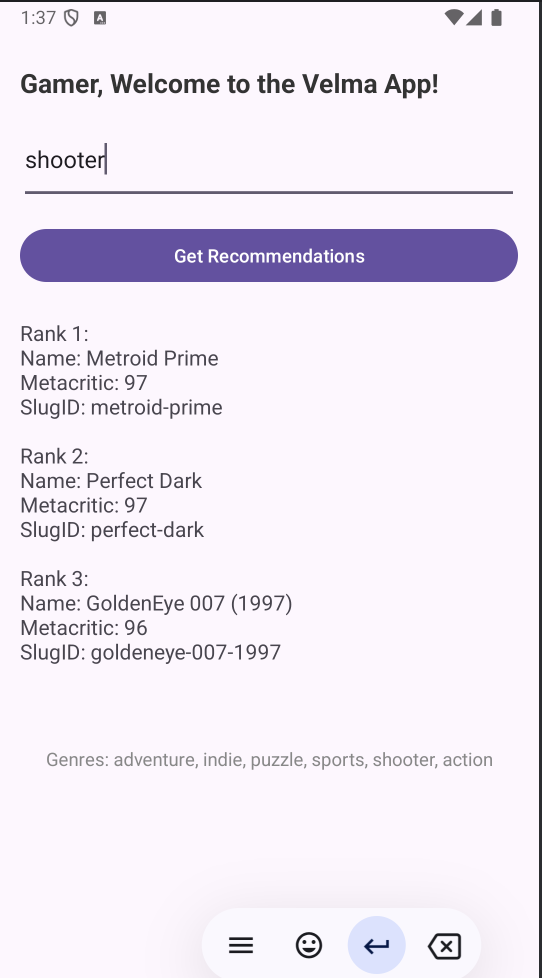
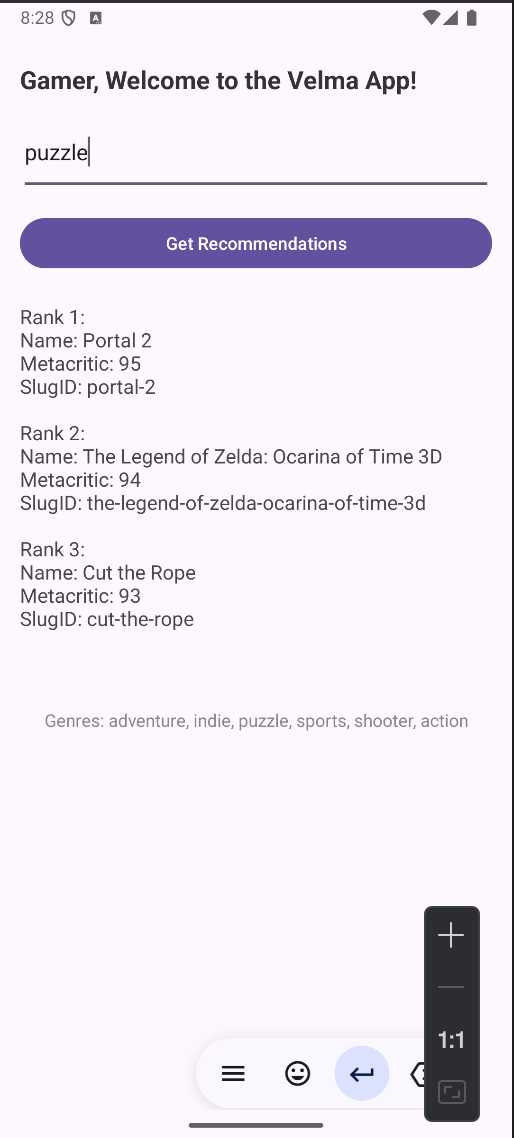
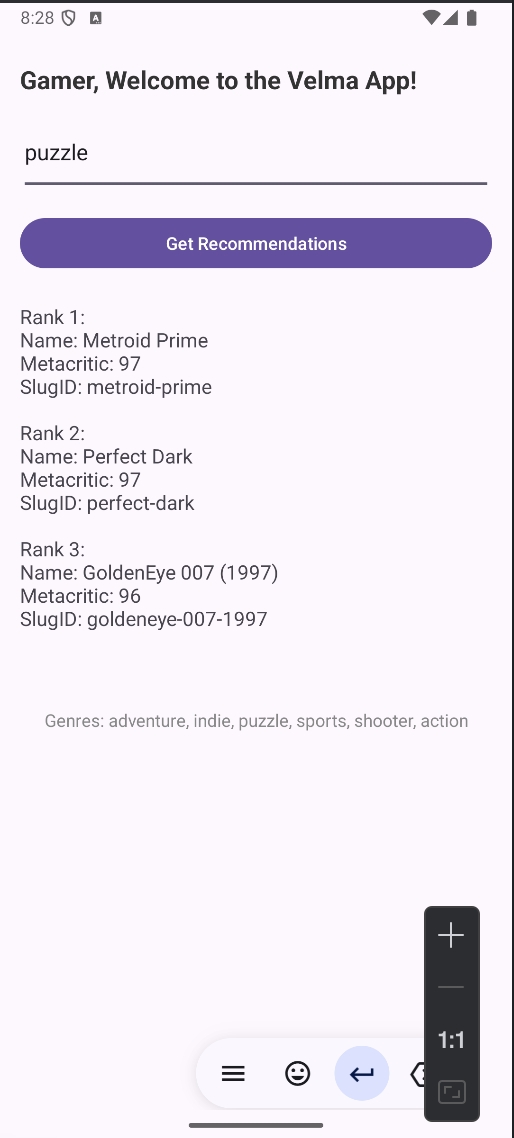
1. **Implement a native Android application**

**  **

The name of my android app is the Velma Recommender App. It is saved as project4Task2. A gamer is welcomed to the Velma app and prompted to enter a genre(e.g. shooter).  
After entry and clicking Get Recommendations, they get the top 3 games ranked by Metacritic score for that specific genre. They can also type in a different genre and get different Recommendations (e.g puzzle)

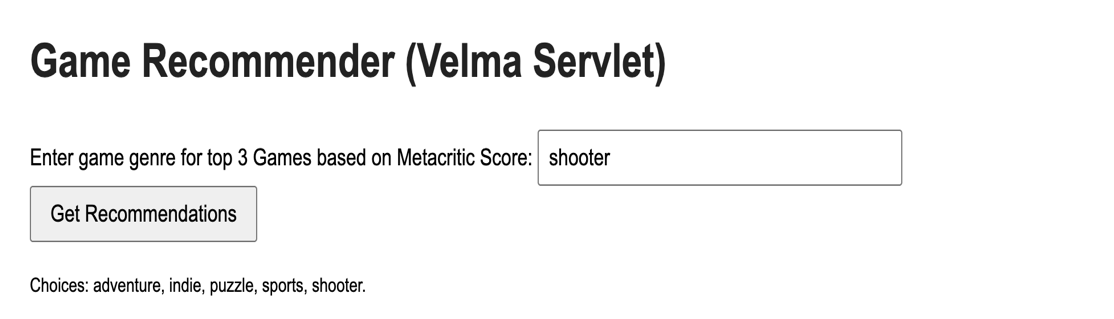
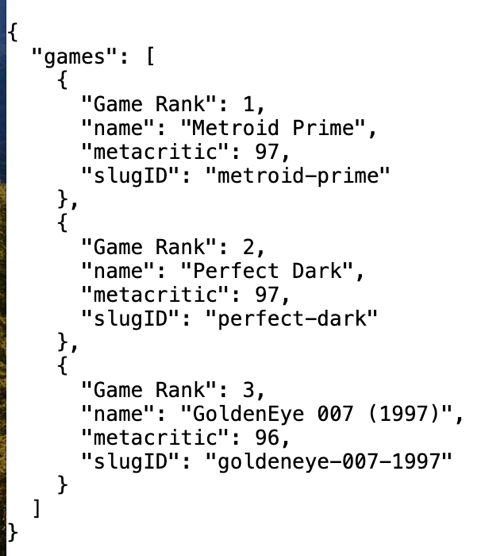


b. The app receives input from user, makes an HTTP request to my webservice via my codespaces URL [**https://supreme-rotary-phone-5gj6qqvjp79c9p6-8080.app.github.dev/getGames?genre=**](https://supreme-rotary-phone-5gj6qqvjp79c9p6-8080.app.github.dev/getGames?genre=)

The project directory name is GameRecommenderServlet

c. Makes an HTTP request (using an appropriate HTTP method) to your web service. Note that this request ***must*** be done using a background thread (see Lab 8's use of BackgroundTask). Do in background shown below:

/\*  
 \* Author: Vimbai Muyengwa  
 \* andrewID: vmuyengw  
 \* Performs the core functionality of the Velma app in a background thread.  
\*  
\* This method:  
\* - Builds a URL with the user's selected genre  
\* - (c) Sends an HTTP GET request to a web service running in GitHub Codespaces  
\* - Reads and collects the JSON response from the server  
\* - (d) Parses the JSON to extract details for the top 3 games (name, metacritic score, slug)  
\* - (e) Updates the UI to display new game recommendations to the user  
\* - (f) Supports repeatable use: the method is called every time the user enters a new genre  
\*  
\* This background execution ensures that the app remains responsive and complies  
\*  
\* References: Lab 8 Code: https://github.com/CMU-Heinz-95702/lab8-AndroidInterestingPicture?tab=readme-ov-file  
\* https://developer.android.com/reference/java/net/HttpURLConnection  
\* GenAI used for troubleshoot  
\*/  
private void doInBackground() {  
 try {  
 // Question 1, C: Makes an HTTP request (using an appropriate HTTP method) to your web service  
 String servletCodeSpaceUrl = "https://supreme-rotary-phone-5gj6qqvjp79c9p6-8080.app.github.dev/getGames?genre=" + genre;  
 Log.*d*("VELMA\_FETCH", "Retrieve from URL: " + servletCodeSpaceUrl); //inserted after trouble shoot  
  
 URL url = new URL(servletCodeSpaceUrl); //Set up URL connection and define request method  
 HttpURLConnection conn = (HttpURLConnection) url.openConnection();  
 conn.setRequestMethod("GET"); //HTTP Get method  
  
 //troubleshoot used GENAI  
 conn.setRequestProperty("X-Device-Model", "Android " + android.os.Build.*MODEL*);  
  
 // Open a stream to read the incoming response line by line  
 BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));  
  
 // Use StringBuilder to accumulate the entire JSON response as one string  
 StringBuilder responseBuilder = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 responseBuilder.append(line);  
 }  
 reader.close(); // Close the stream after reading is complete  
  
 // Question 1 (d): Parse the JSON formatted reply from the web service  
 JSONObject jsonResponse = new JSONObject(responseBuilder.toString()); // Convert string to JSON object  
 JSONArray games = jsonResponse.getJSONArray("games"); // Extract the games array from the JSON  
  
 if (games.length() == 0) { // Handle case where no games were returned  
 topGamesText.append("Zero games in this genre.");  
 return;  
 }  
  
 //Limits to top 3 games  
 int count = Math.*min*(3, games.length());  
 for (int i = 0; i < count; i++) {  
 JSONObject game = games.getJSONObject(i); // Get each game object  
  
 // Safe extract game fields with fallback/default values  
 String name = game.optString("name", "Unknown");  
 String metacritic = game.has("metacritic") ? String.*valueOf*(game.getInt("metacritic")) : "N/A";  
 String slug = game.optString("slugID", "N/A");  
  
 // Question 1, e: Display each game’s rank and detail  
 // Display parsed game information to the user in the UI  
 topGamesText.append("Rank ").append(i + 1).append(":\n")  
 .append("Name: ").append(name).append("\n")  
 .append("Metacritic: ").append(metacritic).append("\n")  
 .append("SlugID: ").append(slug).append("\n\n");  
  
 }  
 //try, catch exception  
 } catch (Exception e) {  
 Log.*e*("VELMA\_FETCH", "Exception during fetch", e); //inserted after troubleshoot issues  
 topGamesText.append("Data fetch failure.");  
 }  
}

d. It takes and parses a JSON formatted reply from my web service seen. below: Web Service Page and response, and displays new information to the gamer. User can continue entering different genres for different results (Includes d-f)

1. Has at least three different kinds of Views in your Layout (TextView, EditText, ImageView, or anything that extends android.view.View). **In order to figure out if something is a View, find its API. If it extends android.view.View then it is a View.**  
   Featured in content\_main .xml. My app contains, TextView, EditText, and the button

<!--  
 Velma App Main Layout  
 This layout defines the UI for the main activity of the Velma game recommendation Android app.  
 stacks components for user interaction, results display, and metadata.  
   
 3 Different views textView, Button, and EditText  
  
 Reference Android Lab 8  
 https://github.com/CMU-Heinz-95702/lab8-AndroidInterestingPicture?tab=readme-ov-file  
  
 GenAI (ChatGPT) was used to assist with troubleshooting layout issues and code generation,.  
-->  
<LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 android:id="@+id/content\_main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="16dp">  
  
 <!-- Welcome Message -->  
 <TextView  
 android:layout\_marginTop="32dp"  
 android:id="@+id/welcomeText"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Gamer, Welcome to the Velma App!"  
 android:textSize="20sp"  
 android:textStyle="bold"  
 android:paddingBottom="12dp"  
 android:textColor="#333"/>  
  
 <!-- user to input a game genre -->  
 <EditText  
 android:id="@+id/genreInput"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Enter game genre (e.g. shooter, puzzle, adventure)"  
 android:minHeight="48dp" />  
  
 <!-- Button to trigger game recommendation based on user input -->  
 <Button  
 android:id="@+id/getGamesButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Get Recommendations"  
 android:minHeight="48dp"  
 android:layout\_marginTop="16dp" />  
  
 <!-- TextView shows top 3 recommended games will be shown -->  
 <TextView  
 android:id="@+id/resultsText"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Three results will appear here based on metacritic score"  
 android:paddingTop="24dp"  
 android:textSize="16sp" />  
   
  
 <!-- Footer to inform the user about valid genre options -->  
 <TextView  
 android:id="@+id/footerGenres"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Genres: adventure, indie, puzzle, sports, shooter, action"  
 android:textSize="14sp"  
 android:textColor="#888"  
 android:layout\_marginTop="24dp"  
 android:gravity="center\_horizontal" />  
  
</LinearLayout>

1. **Implement a web service**
2. Implement a simple (can be a single path) API. – see code below

* Model: The logic that interacts with the third-party API is built into the GameRecommendationServlet class. Receive android input, build RAWG games API URL, send HTTP Get request, read and parson Json, extract essential game data,replies with clean JSON
* View: Android app UI displays the results using VelmaMainActivity. Additionally, an analytics dashboard is served via JSP (e.g., dashboard.jsp) in theGameRecommenderServlet project
* Controller: The controller is implemented in GameRecommendationServlet.java, mapped to a single path: @WebServlet("/getGames")

1. Receives an HTTP request from the native Android application

GameRecommendationServlet.java receives a GET request from the Android app (RecommendationsGetter.java) containing a genre parameter from the user input. see doGet in code below

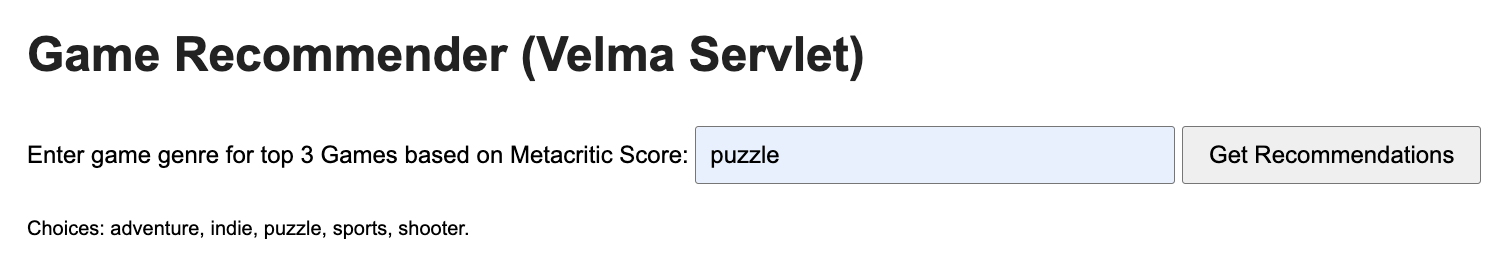
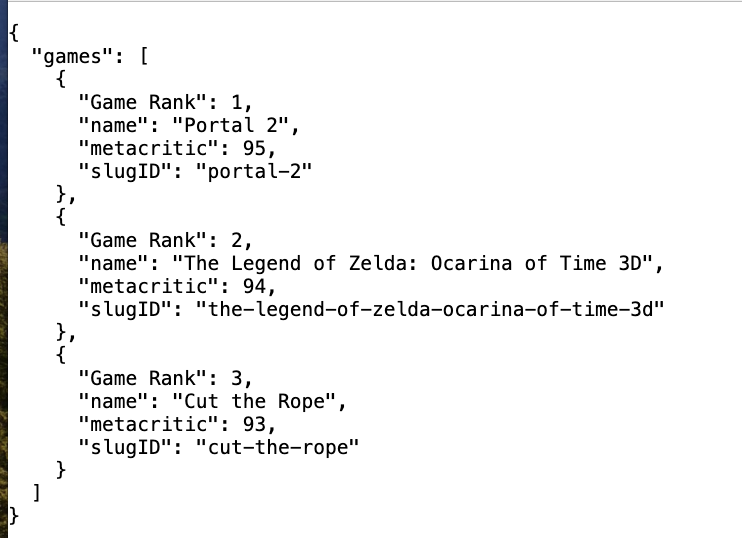
1. Executes business logic appropriate to your application. This includes fetching XML or JSON information from some 3rd party API and processing the response.

GameRecommendationServlet sends an HTTP GET request to the **RAWG Video Game Database API**: String rawGUrl = "https://api.rawg.io/api/games?key=" + API\_KEY + "&genres=" + genre + "&page\_size=3&ordering=-metacritic";

It then parses the JSON response using Gson, Extracts the top 3 games by metacritic score, spits out key game data (name, metacritic, slugID) into JSON response

package project4tasks.ds.gamerecommenderservlet;  
  
  
import java.io.\*;  
import java.net.\*;  
import java.time.Instant;  
  
import com.google.gson.JsonArray;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParser;  
import com.mongodb.client.MongoClient;  
import com.mongodb.client.MongoClients;  
import com.mongodb.client.MongoCollection;  
import com.mongodb.client.MongoDatabase;  
import jakarta.servlet.annotation.WebServlet;  
import jakarta.servlet.http.HttpServlet;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
  
import java.time.ZoneId;  
import java.time.format.DateTimeFormatter;  
  
/\*  
Student: Vimbai Muyengwa  
andrewID: vmuyengw  
  
 \* GameRecommendationServlet  
 \*  
 \* This servlet implements a simple web service that receives HTTP GET requests from a native Android application,  
 \* fetches top 3 game recommendations from the RAWG third-party API (based on a provided genre), processes the JSON  
 \* response to extract only the essential fields (name, metacritic score, slug), and returns a trimmed-down JSON  
 \* response to the client. It also logs detailed request/response information to MongoDB for dashboard analytics.  
 \*  
 \* This servlet satisfies the following:  
 \* a. Implements a single-path API: /getGames  
 \* b. Receives requests from a mobile app (Android)  
 \* c. Executes business logic by calling a 3rd-party API (RAWG) and processes JSON  
 \* d. Replies with a clean JSON response (with only needed data)  
 \* e. Logs 6+ analytics fields to MongoDB for dashboard display  
  
 References:  
 RAWG. (n.d.). RAWG API documentation. https://rawg.io/apidocs  
 https://github.com/CMU-Heinz-95702/Project4/blob/master/README.md  
 Mongo Bison information et al.  
  
 Previous lab2 for servlet functionality Review   
 GenAI used for troubleshoot and code generation  
\*/  
  
  
// Implement a simple can be a single path API  
@WebServlet("/getGames")  
public class GameRecommendationServlet extends HttpServlet {  
  
 // API key for authenticating requests to the RAWG API  
 private final String API\_KEY = "19456881592b452f998b624c2fbd70ca";  
  
  
 @Override // Handles HTTP GET requests sent to /getGames  
 protected void doGet(HttpServletRequest servletRequest, HttpServletResponse servletResponse) throws IOException {  
  
 servletResponse.setContentType("application/json"); //Set the response content type to application/json  
  
 // Get and read genre from request parameter  
 // This is input from the user (Android app)  
 String genre = servletRequest.getParameter("genre");  
  
  
 //Used GenAI to for troubleshoot help with date format  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("MMM dd, yyyy HH:mm:ss")  
 .withZone(ZoneId.*of*("America/New\_York"));  
  
 //track time  
 String requestReceivedTimestamp = formatter.format(Instant.*now*());  
  
 long startTime = System.*currentTimeMillis*();  
  
 //// Handle missing or invalid input from the client  
 //Invalid mobile app input  
 if (genre == null || genre.trim().isEmpty()) {  
 servletResponse.setStatus(HttpServletResponse.*SC\_BAD\_REQUEST*);  
 servletResponse.getWriter().write("{\"error\": \"Genre missing\"}");  
 return;  
 }  
  
 try { //try-catch block if 3rd party API is unavailable  
 /\* Fetch top games from RAWG API: Executes business logic appropriate to my application  
 // Fetching JSON information from a 3rd party API  
 Retrive top 3 games  
 \*/  
 String rawGUrl = "https://api.rawg.io/api/games?key=" + API\_KEY + "&genres=" + genre + "&page\_size=3&ordering=-metacritic";  
  
 String thirdPartyApiRequestTimestamp = formatter.format(Instant.*now*()); //// Timestamp for when 3rd-party API request is made  
  
 URL url = new URL(rawGUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
  
 // Read the response from the RAWG API into a StringBuilder  
 BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder responseBuilder = new StringBuilder();  
 String line;  
  
 while ((line = reader.readLine()) != null) {  
 responseBuilder.append(line);  
 }  
 reader.close();  
  
 //troubleshoot code  
 // System.out.println("RAWG RESPONSE: " + responseBuilder.toString());  
  
 // Parse the JSON response into a JsonObject  
 JsonObject jsonResponse = JsonParser.*parseString*(responseBuilder.toString()).getAsJsonObject();  
 JsonArray gameResults = jsonResponse.getAsJsonArray("results"); // // Extract the array of game results  
  
 // Handle case where RAWG API returns no results (invalid or rare genre)  
 if (gameResults == null || gameResults.size() == 0) {  
 servletResponse.setStatus(HttpServletResponse.*SC\_NOT\_FOUND*);  
 servletResponse.getWriter().write("{\"error\": \"No games found for the genre '" + genre + "'. Try a different genre.\"}");  
 return;  
 }  
  
 // Construct a simplified JSON response. Replies to the Android application with a JSON formatted response  
 JsonArray essentialGameData = new JsonArray();  
  
 // Loop through each game in the results (top 3)  
 for (int i = 0; i < gameResults.size(); i++) {  
 JsonObject gameDetails = gameResults.get(i).getAsJsonObject();  
 JsonObject essentialData = new JsonObject();  
  
 // Add cleaned-up game data to the new JSON object  
 //https://api.rawg.io/docs/#operation/games\_suggested\_read  
 //pulls essential data for Game  
 essentialData.addProperty("Game Rank", i + 1); // 1 based ranking  
 essentialData.addProperty("name", gameDetails.get("name").getAsString());  
  
 // Check if metacritic rating exists before adding  
 // checks if Third-Party API Returns Invalid Data  
 if (gameDetails.has("metacritic") && !gameDetails.get("metacritic").isJsonNull()) {  
 essentialData.addProperty("metacritic", gameDetails.get("metacritic").getAsDouble());  
 } else {  
 essentialData.addProperty("metacritic", -1); // default if unavailable  
 }  
 essentialData.addProperty("slugID", gameDetails.get("slug").getAsString()); //An ID or a slug identifying this Game per API rawg.io/docs  
 essentialGameData.add(essentialData);  
 }  
  
 // Wrap the array in a parent JSON object  
 JsonObject finalResponse = new JsonObject(); // Create JSON object to return to Android client  
 finalResponse.add("games", essentialGameData);  
  
 // Send the successful 200 OK response to the client with the JSON data  
 servletResponse.setStatus(HttpServletResponse.*SC\_OK*);  
 servletResponse.getWriter().write(finalResponse.toString());  
  
 /\*  
 \* Dashboard Logging  
 \*The purpose of logging data to the database is to be able to create an operations dashboard for your web service.  
 \* This dashboard should be web page interface for use from a desktop or laptop browser (not a mobile device).  
 \* The dashboard should display two types of data: 3.1. Operations analytics – display at least 3 interesting operations analytics from your web service.  
 \* You should choose analytics that are relevant to your specific web service. Examples for InterestingPicture might be top 10 picture search terms, average Flickr search latency, or the top 5 Android phone models making requests.  
 \* 3.2. Logs – display the data logs being stored for each mobile phone user interaction with your web service  
 \*  
 \*/  
 //Logging to MongoDB  
 long endTime = System.*currentTimeMillis*();  
 long latency = endTime - startTime;  
  
 // // Get metadata from request (device model)  
 String phoneModel = servletRequest.getHeader("Device Model");  
 if (phoneModel == null || phoneModel.trim().isEmpty()) { // error response if phone model is null  
 phoneModel = "Unknown Device";  
 }  
 String appParams = "genre=" + genre;  
  
 //construct log fields  
 StringBuilder replyData = new StringBuilder(); //stringbuilder for replyData  
 for (int i = 0; i < essentialGameData.size(); i++) {  
 JsonObject game = essentialGameData.get(i).getAsJsonObject();  
 replyData.append(game.get("name").getAsString());  
 if (i < essentialGameData.size() - 1) replyData.append(", ");  
 }  
  
 String replyInfo = "Status: 200 OK | Games Returned: " + essentialGameData.size();  
  
 // Build LoggingData object for Mongo DB. Persists log to mongo db  
 // track overall performance with latency measures for the requests for specific genres  
 LoggingData log = new LoggingData(  
 phoneModel,  
 appParams, // the appParams are the genres  
 requestReceivedTimestamp,  
 thirdPartyApiRequestTimestamp,  
 replyData.toString(),  
 replyInfo,  
 latency  
 );  
  
 // Insert into MongoDB using login credentials  
 MongoClient mongoClient = MongoClients.*create*("mongodb+srv://vmuyengw:2Dzidzayi$$@cluster0.jppey.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0");  
 MongoDatabase database = mongoClient.getDatabase("velmaGameApp"); // Access the specific MongoDB database named 'velmaGameApp'  
 MongoCollection<org.bson.Document> logCollection = database.getCollection("logs"); //// Access the collection named 'logs' within the 'velmaGameApp' database  
  
 logCollection.insertOne(log.toDocument()); // Insert the current request's log data as a new document into the collection  
  
 mongoClient.close();  
  
 } catch (Exception e) {  
 // Handle 3rd-party API unavailable or invalid data  
 servletResponse.setStatus(HttpServletResponse.*SC\_INTERNAL\_SERVER\_ERROR*);  
 servletResponse.getWriter().write("{\"error\": \"Something went awry. Try again later.\"}");  
 e.printStackTrace();  
 }  
 }  
}

d. Replies to the Android application with an XML or JSON formatted response. The schema of the response can be of your own design.

After running web servlet, the below UI pops up. User enters genre and it generates the top 3 ranked games based on the Metacritic score in JSON format 

**Handle error conditions**

Your application should test for and handle gracefully: Please see excerpts included in code below

* Invalid mobile app input

// Handle missing or invalid input from the client  
//Invalid mobile app input  
if (genre == null || genre.trim().isEmpty()) {  
 servletResponse.setStatus(HttpServletResponse.*SC\_BAD\_REQUEST*);  
 servletResponse.getWriter().write("{\"error\": \"Genre missing\"}");  
 return;  
}

* Invalid server-side input (regardless of mobile app input validation)

// Handle case where RAWG API returns no results (invalid or rare genre)  
if (gameResults == null || gameResults.size() == 0) {  
 servletResponse.setStatus(HttpServletResponse.*SC\_NOT\_FOUND*);  
 servletResponse.getWriter().write("{\"error\": \"No games found for the genre '" + genre + "'. Try a different genre.\"}");  
 return;  
}

* Third-party API unavailable

catch (Exception e) {  
 // Handle 3rd-party API unavailable or invalid data  
 servletResponse.setStatus(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR);  
 servletResponse.getWriter().write("{\"error\": \"Something went awry. Try again later.\"}");  
 e.printStackTrace();  
 }

* Third-party API invalid data

// checks if Third-Party API Returns Invalid Data  
 if (gameDetails.has("metacritic") && !gameDetails.get("metacritic").isJsonNull()) {  
 essentialData.addProperty("metacritic", gameDetails.get("metacritic").getAsDouble());  
 } else {  
 essentialData.addProperty("metacritic", -1);

1. **Dashboard**

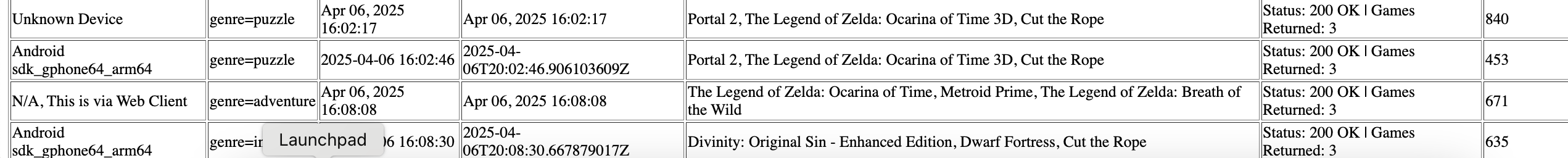
My implementation logs all key request-response data from the Android app (and optionally the web browser) to MongoDB. These logs are displayed in a structured JSP-based web interface accessible from a desktop browser. Each log shows 6 pieces of Info. I used Use <20 lines of JSP, don’t display logs as JSON/XML, and also utilized tables

Three or more interesting Operations analytics shown

1. Total Requests – count of requests
2. Average Latency – if I were designing this app to go live, this would be top of mind for my users/gamers. No one wants to wait to play
3. The Top 2 searched game genres – this give me insights on what we should urge our partners to create and or advise to users based on user interest/behavior
4. Success/Failure shown in reply info



…



##### 4. Log useful information: At least 6 pieces of information is logged for each request/reply with the mobile phone. It should include information about the request from the mobile phone, information about the request and reply to the 3rd party API, and information about the reply to the mobile phone. (You should NOT log data from interactions from the operations dashboard.)

My app contains 6 pieces of information phoneModel, appRequestParameters (gener), requestReceivedTimestamp, thirdPartyApiRequestTimestamp, and thirdPartyAPIReplyData, replytoMobileInfo about status plus the numbers of Games Returned, and finally the latency

**phoneModel** – Logs device model using the X-Device-Model header from the Android app "Android sdk\_gphone64\_arm64"

**appRequestParameters** – Logs the genre parameter sent by the Android app ("genre=shooter") for user interest tracking

**requestReceivedTimestamp** – Records when server first receives request from the app. Tracks total handling time to identify peak usage times

**thirdPartyApiRequestTimestamp** – Timestamp of when the RAWG API call is made.  
Part of latency analysis and API delays

**thirdPartyApiReplyData** – Logs a summary of the game names returned from the RAWG API displayed to user

**replyToMobilePhoneInfo** – A message showing response status "Status: 200 OK | Games Returned: 3". Confirms success/failure and count of games

**latency** – Calculates time from request to API response for performance monitoring.

**Logging**

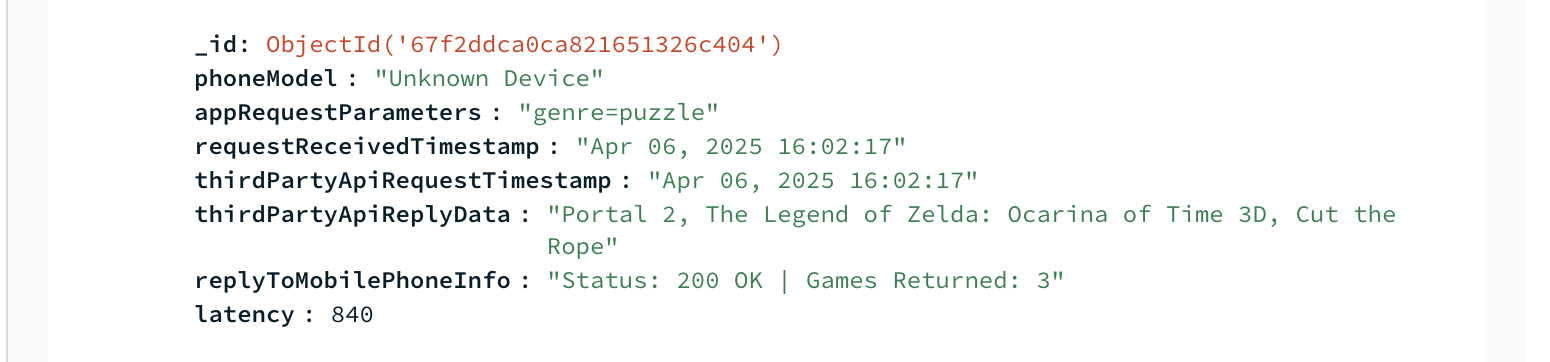
1. Store the log information in a database

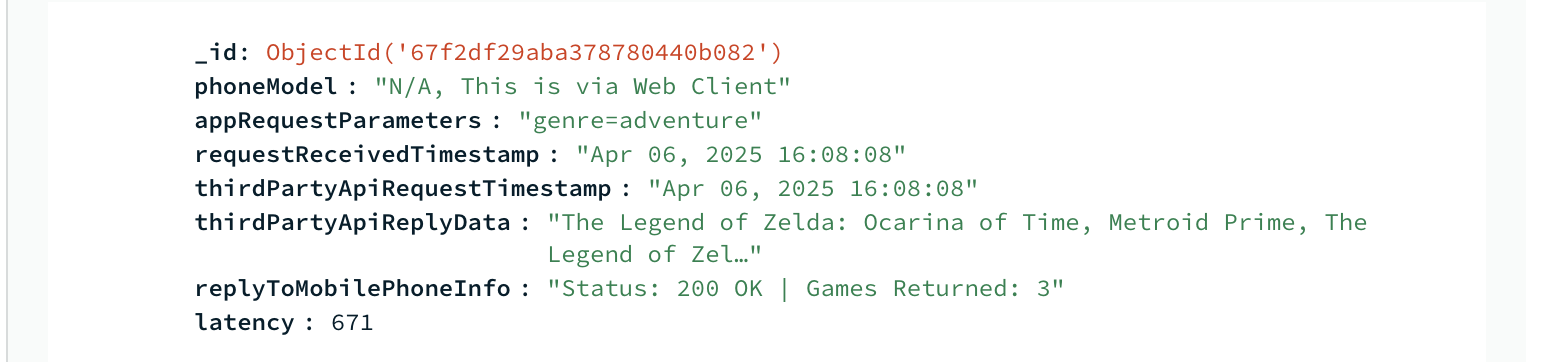
I store all log data in **MongoDB Atlas**, using the following connection:

mongodb+srv://vmuyengw:2Dzidzayi$$@cluster0.jppey.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

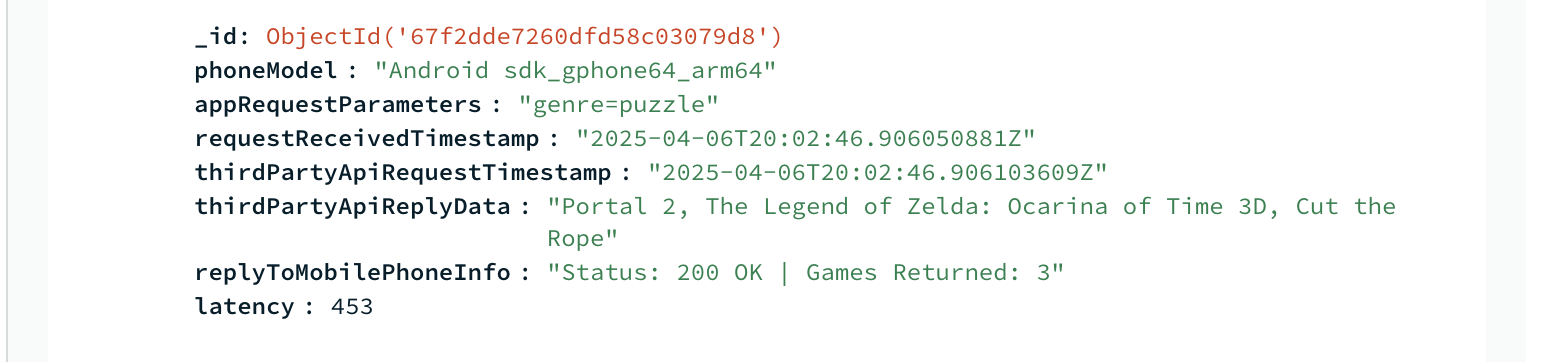
* **Cluster Name:** Cluster0
* **Shards:** This Atlas cluster is a free-tier cluster
* **Database Name:** velmaGameApp
* **Collection:** logs

The web service can connect, store, and retrieve information from a MongoDB database in the cloud. Data is stored persistently.

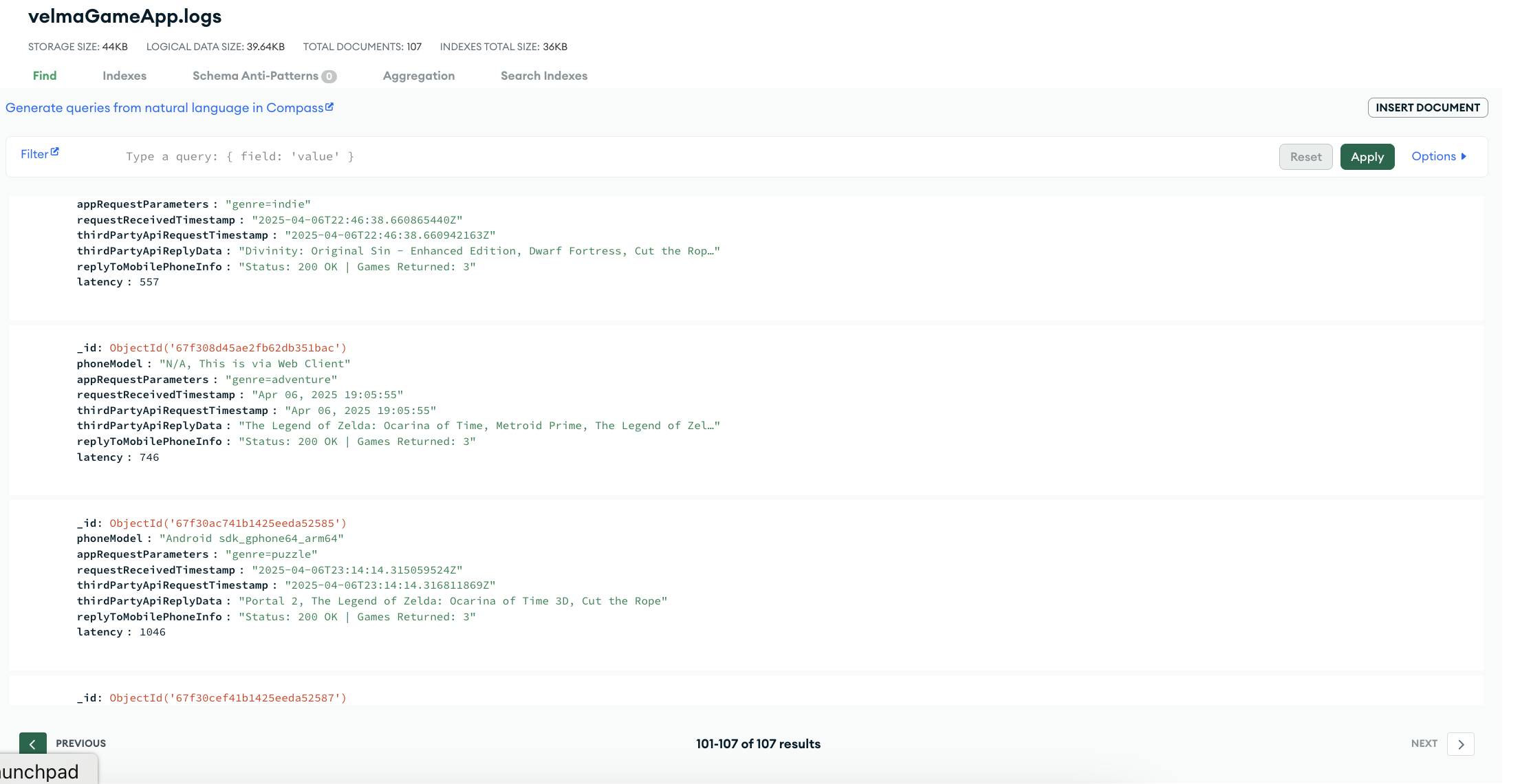




Above is the Mongo Entry from running Web Service locally. Below shows android data



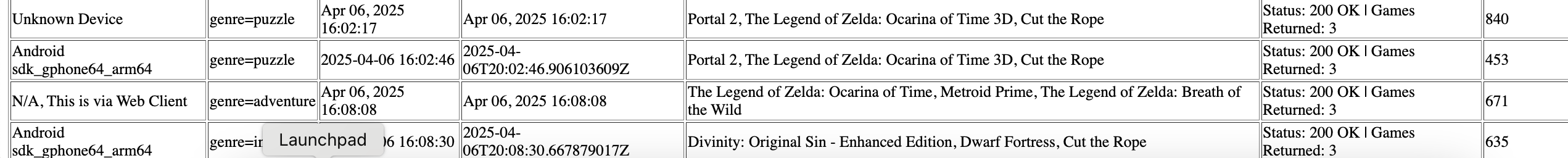
Data persistency shown below: 107 results from multiple troubleshoot sessions until finalization



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

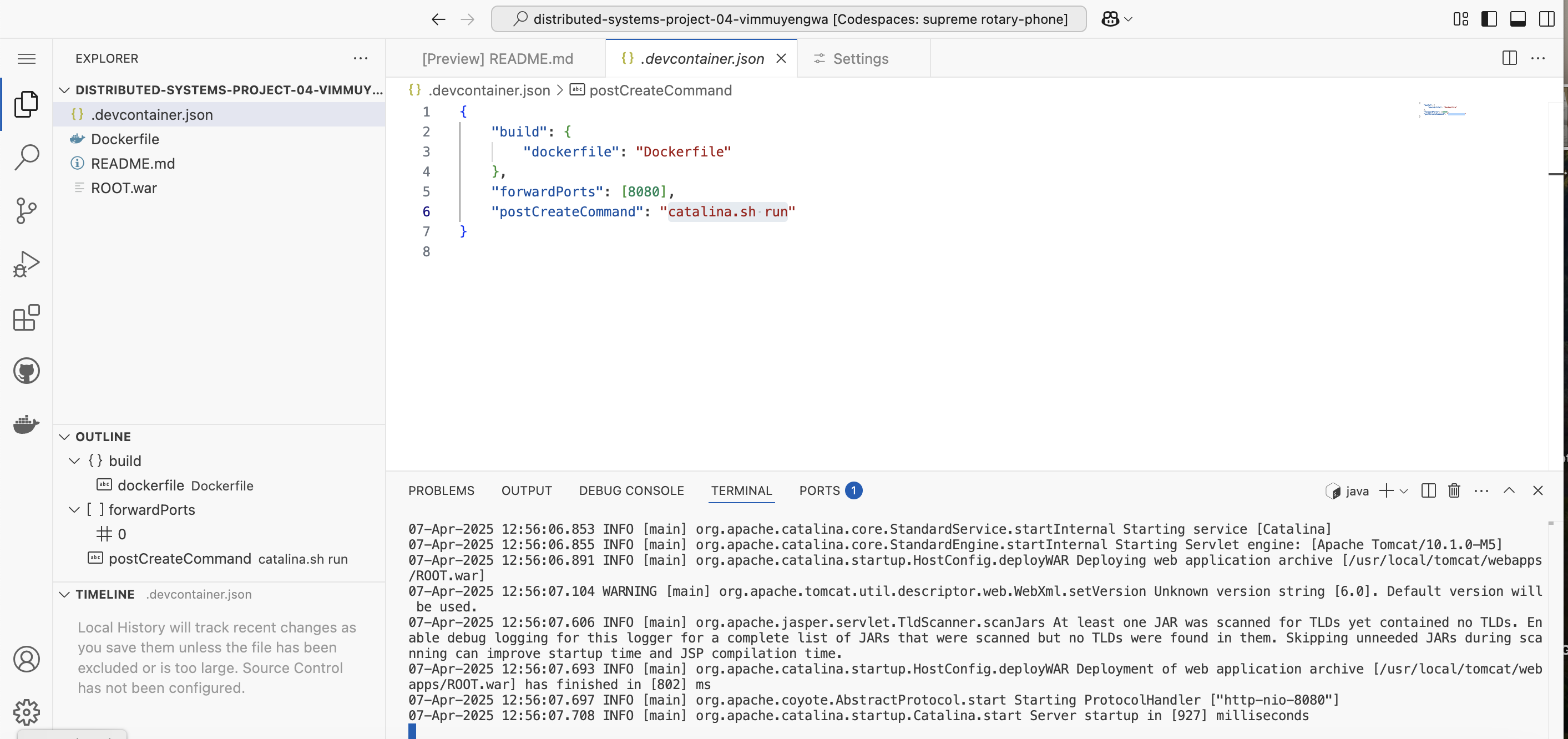


The above was before I corrected Phone model response. See below for phone model and better timestamp legibility.



1. A unique URL addresses a web interface dashboard for the web service.

<http://localhost:8080/GameRecommenderServlet/dashboard> (run the GameServletRecommender first)

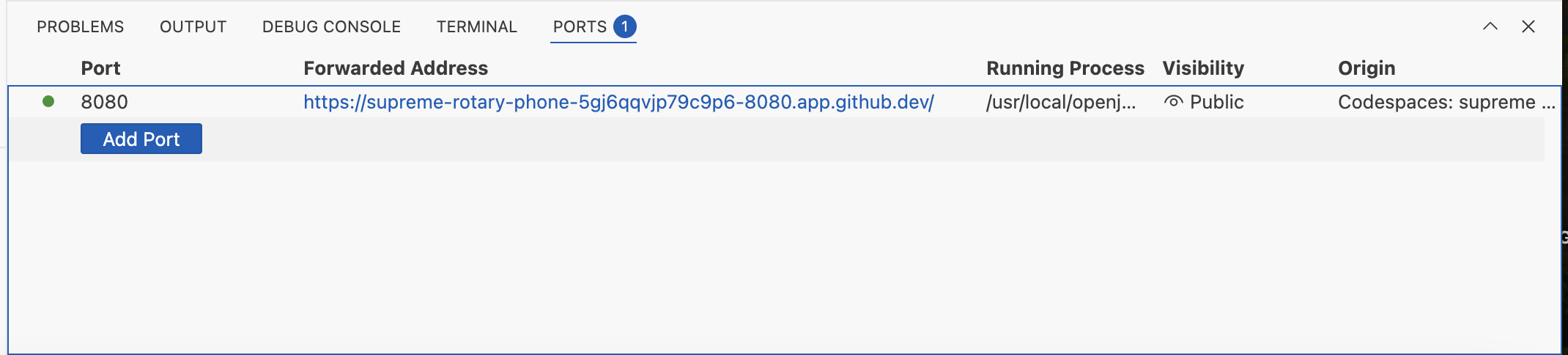
7. **Codespaces**

ports

my web service

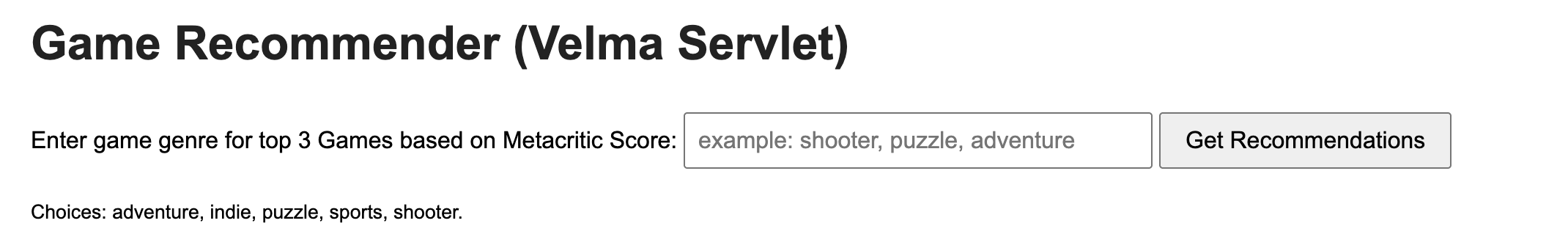
v

v



change from private to public

visit forwarded address with global icon





Results

**Works Cited:**

Public APIs. (n.d.). Art & design. GitHub. <https://github.com/public-apis/public-apis?tab=readme-ov-file#art--design>

RAWG. (n.d.). RAWG API documentation. <https://rawg.io/apidocs>