Project 4 Task 2
Sijia Ma (sijiam@andrew.cmu.edu)

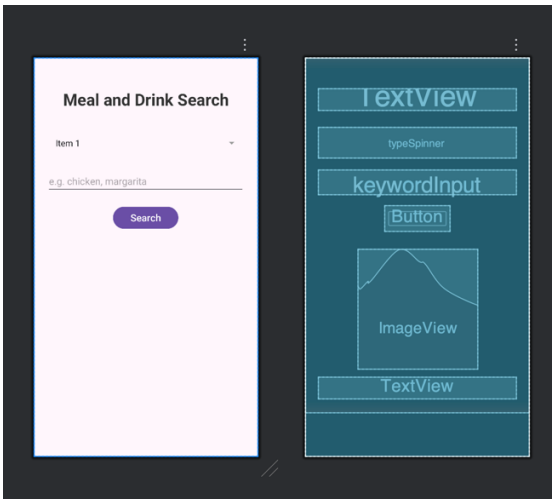
Description:

My application is a 'Meal and drink search'. It prompts users for the keyword of the meal or drink they want (users can choose either meal or drinks), and returns the name, picture, category, and area of the food.

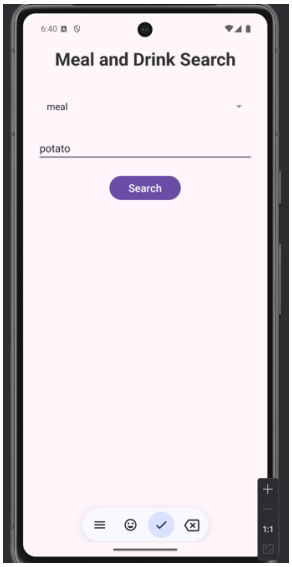
Here is how my application meets the task requirements:

1. Implement a native Android application

The name of my native Android application project in Android Studio is: MealSearchAndroid
a. Has at least three different kinds of Views in your Layout: My application uses TextView, EditText, ImageView, TypeSpinner, Button. And the image shows how they are incorporated into the LinearLayout



b. Requires input from the user: Here is a screenshot of the user searching for a meal has potato



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

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

```
private static final String BASE_URL = "https://probable-trout-
q5wp4jwj9j6h94q6-8080.app.github.dev/api/search";
……
// Build URL
String requestUrl = BASE_URL + "?type=" + encodedType + "&keyword=" +
encodedKeyword;
```

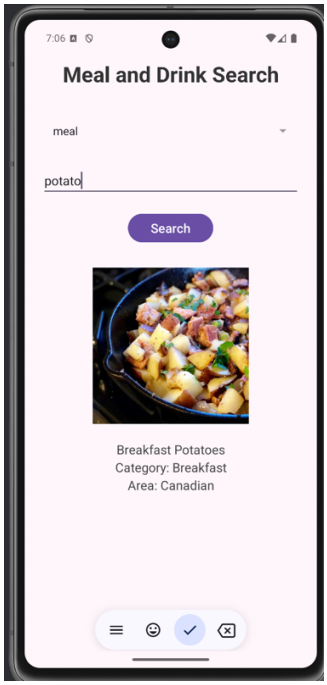
The search method makes this request of my web application, parses the returned JSON to find the URL of meal / drinks.

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

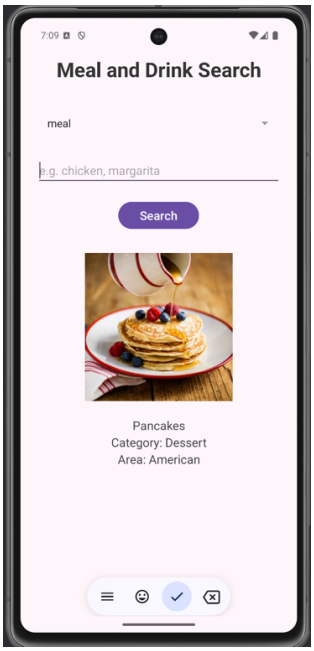
```
    private void parseJSONResponse(String type, String json) {
        Handler handler = new Handler(Looper.getMainLooper());

        handler.post(() -> {
            try {
                JSONObject jsonObject = new JSONObject(json);

                if (type.equals("meal") && jsonObject.has("meals")
&& !jsonObject.isNull("meals")) {
                    JSONArray meals = jsonObject.getJSONArray("meals");

                    if (meals.length() > 0) {
                        JSONObject meal = meals.getJSONObject(0);

                        String name = meal.getString("strMeal");
                        String imageUrl = meal.getString("strMealThumb");
                        String category = meal.optString("strCategory",
"N/A");
                        String area = meal.optString("strArea", "N/A");

                        String display = name + "\nCategory: " + category +
"\nArea: " + area;
                        resultText.setText(display);
                        Picasso.get().load(imageUrl).into(resultImage);
                    } else {
                        resultText.setText("No meals found for: " +
keywordInput.getText().toString());
                        resultImage.setImageDrawable(null);
                    }
                } else if (type.equals("drink") && jsonObject.has("drinks")
&& !jsonObject.isNull("drinks")) {
                    JSONArray drinks = jsonObject.getJSONArray("drinks");

                    if (drinks.length() > 0) {
                        JSONObject drink = drinks.getJSONObject(0);

                        String name = drink.getString("strDrink");
                        String imageUrl = drink.getString("strDrinkThumb");
                        String category = drink.optString("strCategory",
"N/A");
```

```
                        String alcoholic = drink.optString("strAlcoholic",
"N/A");

                        String display = name + "\nCategory: " + category +
"\nAlcoholic: " + alcoholic;
                        resultText.setText(display);
                        Picasso.get().load(imageUrl).into(resultImage);
                    } else {
                        resultText.setText("No drinks found for: " +
keywordInput.getText().toString());
                        resultImage.setImageDrawable(null);
                    }
                } else {
                    resultText.setText("No results found.");
                    resultImage.setImageDrawable(null);
                }
            } catch (Exception e) {
                resultText.setText("Error parsing response: " +
e.getMessage());
                resultImage.setImageDrawable(null);
                Log.e(TAG, "Parse error: " + e.getMessage());
                e.printStackTrace();
            }
        });
    }
}
```

e. Displays new information to the user: Here is the screen shot after the meal has been returned.



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



2. Implement a web service

The URL of my web service: https://probable-trout-q5wp4jwj9j6h94q6.github.dev

a. Implement a simple (can be a single path) API

In my web app project:
Model: MenuApiClient.java, MongoLogger.java
View: index.jsp(Welcome page), test.jsp(Test the servlet/client), dashboard.jsp(Log)
Controller: MealDrinkServlet.java

b. Receives an HTTP request from the native Android application

My Android client sends an HTTP GET request to the endpoint /api/search, which is handled by the MealDrinkServlet on the server side. The servlet extracts two parameters from the request: the search type (either "meal" or "drink") and the keyword. This logic is implemented in the doGet():

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    // Set response content type
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
```

```
    // Get request parameters
    String type = request.getParameter("type"); // "meal" or "drink"
    String keyword = request.getParameter("keyword");
    String userAgent = request.getHeader("User-Agent");

    // Validate input
    if (type == null || (!type.equals("meal") && !type.equals("drink"))) {
        sendErrorResponse(response, "Invalid type parameter. Use 'meal' or
'drink'");
        return;
    }

    if (keyword == null || keyword.trim().isEmpty()) {
        sendErrorResponse(response, "Missing or empty keyword parameter");
        return;
    }

    // Prepare log data (with at least 6 pieces of information)
    JsonObject logData = new JsonObject();
    logData.addProperty("timestamp", new Date().toString());
    logData.addProperty("type", type);
    logData.addProperty("keyword", keyword);
    logData.addProperty("userAgent", userAgent);
    logData.addProperty("ipAddress", request.getRemoteAddr());
    logData.addProperty("requestTime", System.currentTimeMillis());
    logData.addProperty("queryString", request.getQueryString());

    try {
        long startTime = System.currentTimeMillis();
        String resultJson;

        // Call the appropriate API based on type
        if (type.equals("meal")) {
            resultJson = apiClient.searchMeal(keyword);
        } else {
            resultJson = apiClient.searchDrink(keyword);
        }

        long endTime = System.currentTimeMillis();
        long responseTime = endTime - startTime;

        // Add response info to log
        JsonObject parsedResult = gson.fromJson(resultJson,
JsonObject.class);
        String resultsArrayKey = type.equals("meal") ? "meals" : "drinks";
        int resultCount = 0;

        if (parsedResult.has(resultsArrayKey)
&& !parsedResult.get(resultsArrayKey).isJsonNull()) {
            resultCount =
parsedResult.getAsJsonArray(resultsArrayKey).size();
        }

        logData.addProperty("responseTime", responseTime);
        logData.addProperty("resultCount", resultCount);
        logData.addProperty("apiResponseTime", responseTime + " ms");
```

```
        logData.addProperty("status", "success");

        // Log to MongoDB
        mongoLogger.logSearch(logData.toString());
        System.out.println("Logged search: " + type + " - " + keyword);

        // Send the API response directly to client
        PrintWriter out = response.getWriter();
        out.print(resultJson);
        out.flush();

    } catch (Exception e) {
        // Log error
        logData.addProperty("status", "error");
        logData.addProperty("errorMessage", e.getMessage());
        mongoLogger.logSearch(logData.toString());

        // Send error response
        sendErrorResponse(response, "Error processing request: " +
e.getMessage());
    }
}
```

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

The servlet delegates the API call to the MenuApiClient, which sends a request to either TheMealDB or TheCocktailDB, depending on the search type. The client retrieves a JSON response and returns it as a string. The servlet then parses part of this response using Gson in order to log additional metadata (e.g., result count and response time) to MongoDB:

```
// Call the appropriate API based on type
if (type.equals("meal")) {
    resultJson = apiClient.searchMeal(keyword);
} else {
    resultJson = apiClient.searchDrink(keyword);
}

long endTime = System.currentTimeMillis();
long responseTime = endTime - startTime;

// Add response info to log
JsonObject parsedResult = gson.fromJson(resultJson, JsonObject.class);
String resultsArrayKey = type.equals("meal") ? "meals" : "drinks";
int resultCount = 0;

if (parsedResult.has(resultsArrayKey)
&& !parsedResult.get(resultsArrayKey).isJsonNull()) {
    resultCount = parsedResult.getAsJsonArray(resultsArrayKey).size();
}
```

The actual API communication is performed in MenuApiClient:

```
public String searchMeal(String keyword) throws Exception {
    String encodedKeyword = URLEncoder.encode(keyword,
```

```
StandardCharsets.UTF_8.toString());
    String urlString = MEAL_API_BASE_URL + "/search.php?s=" + encodedKeyword;
    return fetchRawJsonFromUrl(urlString);
}
```

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 logging the data, the servlet sends the full JSON response back to the Android client. The servlet sets the response content type to "application/json" and writes the string directly to the output stream:

```
// Set response content type
response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
……
// Send the API response directly to client
PrintWriter out = response.getWriter();
out.print(resultJson);
out.flush();
```

On the Android side, the client parses this JSON using the JSONObject and JSONArray classes to extract specific fields like the name, category, and image URL:

```
JSONObject jsonObject = new JSONObject(json);
……
String name = meal.getString("strMeal");
String imageUrl = meal.getString("strMealThumb");
String category = meal.optString("strCategory", "N/A");
String area = meal.optString("strArea", "N/A");
```

3. Handle error conditions

Invalid mobile app input：Returns “Please enter a keyword”:

```
// Validate user input
if (keyword.isEmpty()) {
    Toast.makeText(this, "Please enter a keyword",
Toast.LENGTH_SHORT).show();
    return;
}
```

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

```
// Validate input
if (type == null || (!type.equals("meal") && !type.equals("drink"))) {
    sendErrorResponse(response, "Invalid type parameter. Use 'meal' or
'drink'");
    return;
}

if (keyword == null || keyword.trim().isEmpty()) {
    sendErrorResponse(response, "Missing or empty keyword parameter");
    return;
}
```

Mobile app network failure, unable to reach server

```
    } catch (Exception e) {
        e.printStackTrace();
        final String errorMessage = e.getMessage();
        runOnUiThread(() -> {
            resultText.setText("Error: " + errorMessage);
            Toast.makeText(MainActivity.this, "Network Error",
Toast.LENGTH_LONG).show();
        });
    }
}).start();
```

Third-party API unavailable: If the third-party API returns an error status code (e.g., 500, 404), an exception is thrown, which is then caught and handled in the servlet or app.

```
// Check if the request was successful
int responseCode = conn.getResponseCode();
if (responseCode != 200) {
    throw new Exception("HTTP error code: " + responseCode);
}
```

Third-party API invalid data

```
    } else {
        resultText.setText("No drinks found for: " +
keywordInput.getText().toString());
        resultImage.setImageDrawable(null);
    }
} else {
    resultText.setText("No results found.");
    resultImage.setImageDrawable(null);
}
```

4. Log useful information

```
// Prepare log data (with at least 6 pieces of information)
JsonObject logData = new JsonObject();
logData.addProperty("timestamp", new Date().toString());
logData.addProperty("type", type);
logData.addProperty("keyword", keyword);
logData.addProperty("userAgent", userAgent);
logData.addProperty("ipAddress", request.getRemoteAddr());
logData.addProperty("requestTime", System.currentTimeMillis());
logData.addProperty("queryString", request.getQueryString());
```

I log 10 fields per request to support monitoring, analytics, and debugging:

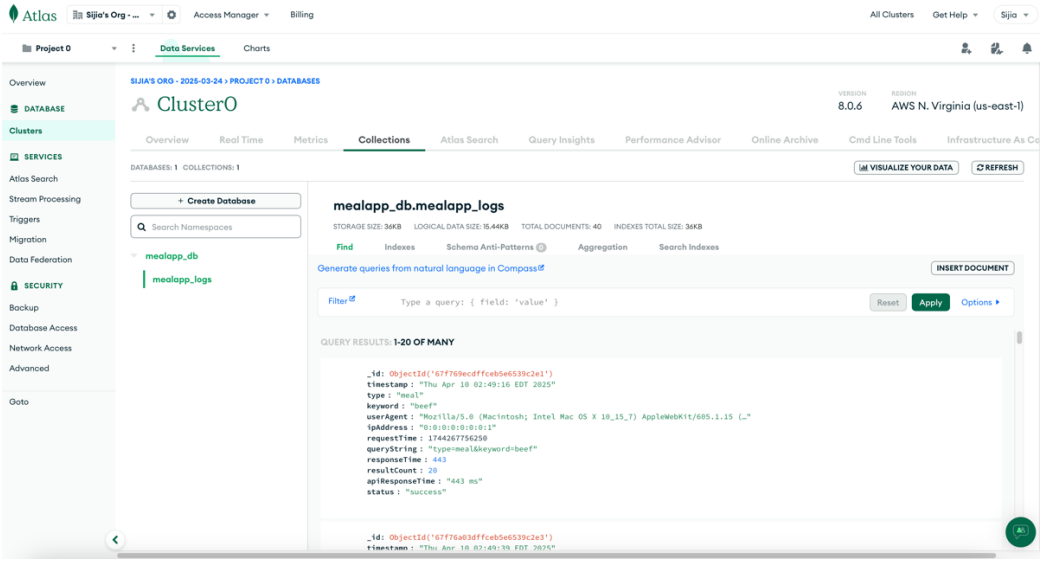
- **timestamp** – to track when requests happen and analyze peak usage times
- **type** & **keyword** – to understand what users are searching for and generate top keyword stats
- **userAgent** & **ipAddress** – to identify client devices and trace potential issues
- **requestTime** & **responseTime** – to measure backend performance and detect slowdowns
- **queryString** – to help reproduce and debug specific requests
- **resultCount** & **status** – to assess API success and search quality

These fields were chosen to balance system visibility, user behavior insight, and operational reliability.

5. Store the log information in a database

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

```
public class MongoLogger {
    // MongoDB connection details
    private static final String CONNECTION_STRING =
"mongodb+srv://sijiam:gaosuwo-
90@cluster0.0hh3o.mongodb.net/?appName=Cluster0";
    private static final String DATABASE_NAME = "mealapp_db";
    private static final String LOGS_COLLECTION = "mealapp_logs";
```



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

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

https://probable-trout-q5wp4jwj9j6h94q6-8080.app.github.dev/dashboard.jsp
b. The dashboard displays at least 3 interesting operations analytics.
c. The dashboard displays formatted full logs.

Meal & Drink Search Dashboard

Refresh Data

Analytics Overview

Total Searches
40

Average Response Time
268.33 ms

Search Type Distribution
23 Meals / 17 Drinks

Top Search Keywords

Keyword	Search Count
chicken	6
beef	5
gin	4
potato	3
tonic	3
coke	2
cake	2
fish	2
glass	1
glue	1

Recent Search Logs

Time	Type	Keyword	Results	Response Time	Status
2025-04-10 11:09:24	meal	cake	19	251 ms	success
2025-04-10 11:09:08	meal	sesame	0	196 ms	success
2025-04-10 11:06:45	meal	potato	10	334 ms	success
2025-04-10 11:02:24	drink	coke	4	190 ms	success
2025-04-10 11:01:37	drink	lemon	11	331 ms	success
2025-04-10 10:40:21	meal	potato	10	293 ms	success
2025-04-10 10:26:28	drink	cake	0	104 ms	success
2025-04-10 10:26:14	drink	glue	1	105 ms	success
2025-04-10 10:25:41	drink	gin	20	361 ms	success
2025-04-10 10:25:12	meal	glass	0	196 ms	success
2025-04-10 10:24:18	meal	salmon	5	202 ms	success
2025-04-10 10:19:03	drink	margarita	6	209 ms	success
2025-04-10 10:11:56	drink	coke	4	204 ms	success
2025-04-10 10:11:47	drink	sprite	0	126 ms	success
2025-04-10 10:09:38	meal	chicken	25	366 ms	success
2025-04-10 10:05:35	meal	chicken	25	122 ms	success
2025-04-10 10:04:24	meal	chicken	25	391 ms	success
2025-04-10 08:59:38	meal	chicken	25	342 ms	success
2025-04-10 08:47:38	drink	gin	20	354 ms	success
2025-04-10 08:44:16	meal	chicken	25	374 ms	success