Project 4

Name: Aubrey Kuang

Andrew ID: yongbeik

My mobile app is a digital museum designed for Metropolitan Museum of Art enthusiasts. It allows users to search for artworks using the museum's API, and explore detailed information of their favorite pieces.

> API name: The Metropolitan Museum of Art Collection API
>
> API Documentation: https://metmuseum.github.io

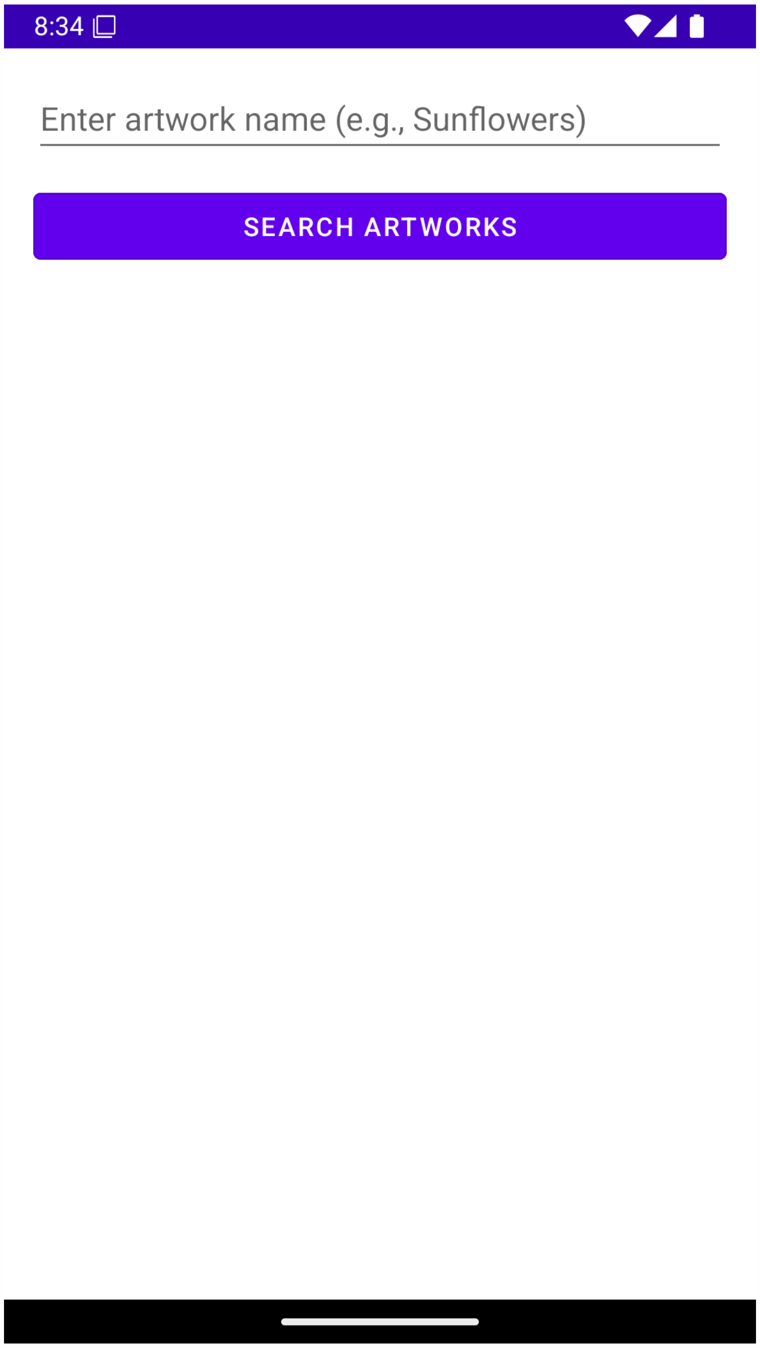
1. Implement a native Android application

The name of my native Android application project is **Project4APP**

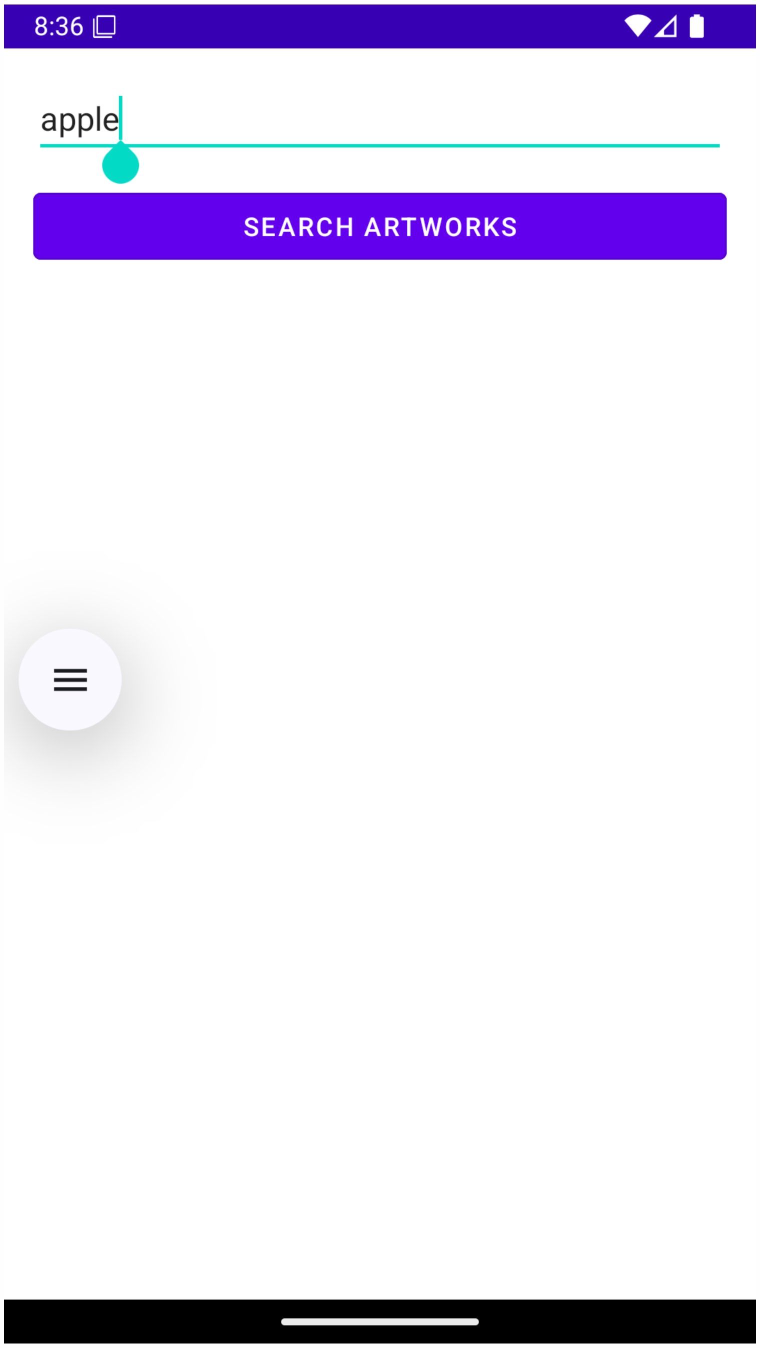
a. Has at least three different kinds of Views in Layout ✅

My application uses EditText, Button, ProgressBar, RecyclerView, TextView, and ImageView. See activity_-main.xml for details of how EditText, Button, ProgressBar, RecyclerView, and TextView are incorporated into the main screen layout. The ImageView is used in item_artwork.xml which defines the layout for individual artwork items displayed in the RecyclerView.

EditText is used to enter search terms, Button triggers the search operation, ProgressBar shows the loading status, RecyclerView displays the search result list, and TextView is used to display the empty result status. In each item of the search result, ImageView is used to display the thumbnail of the artwork.



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 MainActivity.java using Volley library. The HTTP request is:

```
http://10.0.2.2:8000/api/search?q=<query>
```

where `<query>` is the user's search term.

The performSearch method constructs this request URL, sends it to my web service, and handles the JSON response, including artworkID, title, author name and pictureURL. Volley automatically handles the request on a background thread, ensuring network operations don't block the UI thread.

```java
// in MainActivity.java
private void performSearch() {
    // Check network connectivity first
    ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
    boolean isConnected = activeNetwork != null &&
activeNetwork.isConnectedOrConnecting();

    if (!isConnected) {
        Toast.makeText(this, "No network connection", Toast.LENGTH_LONG).show();
        return;
    }

    // Get and validate search query
    String query = etSearch.getText().toString().trim();
    if (query.isEmpty()) {
        Toast.makeText(this, "Please enter a search term", Toast.LENGTH_SHORT).show();
        return;
    }

    showLoading(true);

    // URL encode the query parameter
    try {
        query = java.net.URLEncoder.encode(query, "UTF-8");
    } catch (java.io.UnsupportedEncodingException e) {
        Log.e(TAG, "Error encoding URL", e);
    }

    // Construct the request URL
    String url = "http://10.0.2.2:8000/api/search?q=" + query;
    Log.d(TAG, "Search URL: " + url);

    // Create the HTTP GET request
    // This will run on a background thread automatically
    JsonArrayRequest request = new JsonArrayRequest(
            Request.Method.GET, url, null,
            new Response.Listener<JSONArray>() {
                @Override
                public void onResponse(JSONArray response) {
                    // Process the JSON array response on UI thread
                    showLoading(false);
                    parseArtworkResults(response);
                }
            },
            new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        // Handle errors
                        showLoading(false);
                        Toast.makeText(MainActivity.this, "Search failed: " +
error.getMessage(),
                                Toast.LENGTH_LONG).show();
                    }
                }
    );

    // Set timeout policy
    request.setRetryPolicy(new DefaultRetryPolicy(
            10000, // 10 seconds timeout
            DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
            DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
    ));

    // Add request to the queue - this executes the request on a background thread
    requestQueue.add(request);
}
```

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

My application receives and parses JSON responses from the web service. An example of the JSON reply is:

```
{
  "objectID": 551786,
  "title": "Book of the Dead of the Priest of Horus, Imhotep (Imuthes)",
  "culture": "",
  "imageUrl": "https://images.metmuseum.org/CRDImages/eg/web-large/2-35.9.20a-w_EGDP014589-4594.jpg",
  "artist": "",
  "date": "ca. 332-200 B.C."
}
```

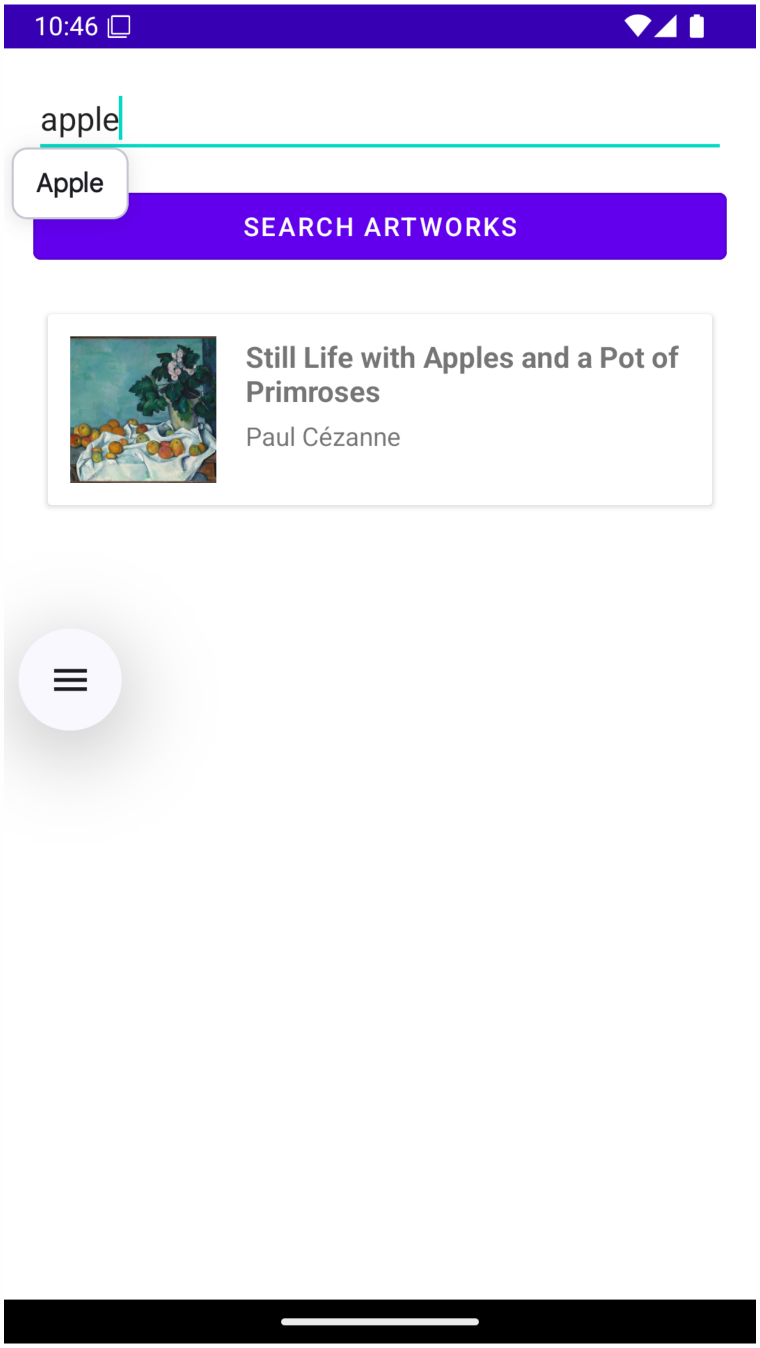
The handleSearchResponse method parses this JSON and creates Artwork objects from it:

```
// in MainActivity.java
private void handleSearchArrayResponse(JSONArray response) {
    // Clear any existing artwork data to prepare for new results
    artworks.clear();

    try {
        // Iterate through each JSONObject in the response array
        for (int i = 0; i < response.length(); i++) {
            JSONObject obj = response.getJSONObject(i);

            // Create a new Artwork object for each item in the array
            Artwork artwork = new Artwork();

            // Extract required fields from JSON and set them in the Artwork object
            artwork.setObjectID(obj.getInt("objectID"));  // Get artwork ID as integer
            artwork.setTitle(obj.getString("title"));     // Get artwork title
            artwork.setArtist(obj.getString("artist"));   // Get artist name
            artwork.setImageUrl(obj.getString("imageUrl")); // Get image URL

            // Handle optional fields - check if they exist before extracting
            if (obj.has("culture") && !obj.isNull("culture")) {
                artwork.setCulture(obj.getString("culture"));
            }
            if (obj.has("date") && !obj.isNull("date")) {
                artwork.setDate(obj.getString("date"));
            }

            // Add the populated Artwork object to the collection
            artworks.add(artwork);
        }
    } catch (JSONException e) {
        // Handle any JSON parsing errors
        Toast.makeText(this, "Error parsing array results: " + e.getMessage(),
Toast.LENGTH_SHORT).show();
    }

    // Update the UI with the new artwork data
    updateUI();

    // Hide the loading indicator
    showLoading(false);
}
```

The app also handles JSON array responses for multiple artwork results using the handleSearchArrayResponse method, which iterates through each JSON object in the array and creates Artwork objects in a similar manner.

e. Displays new information to the user ✅

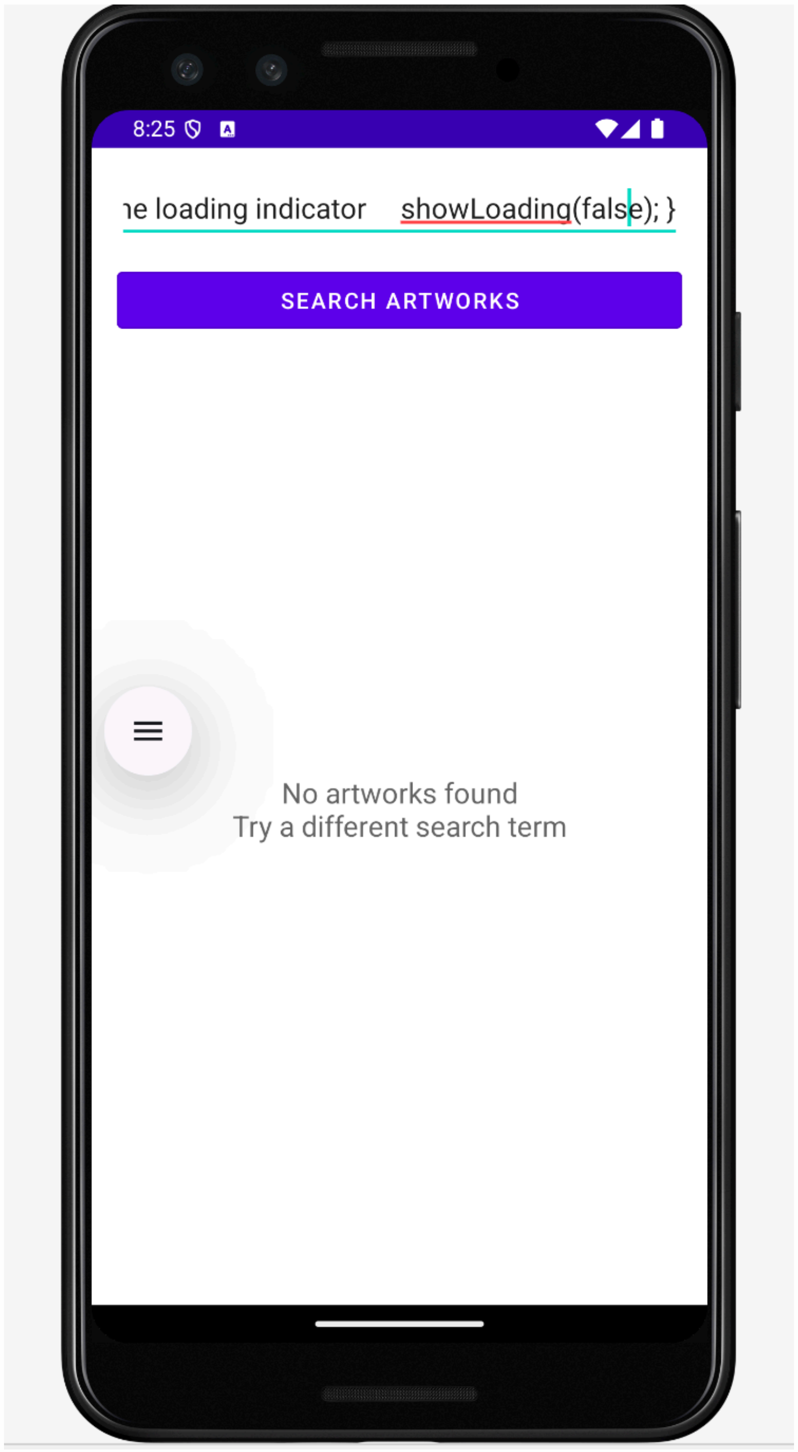
The screen shot shows results found. It displays the artwork's title, author and picture.



Still Life with Apples and a Pot of Primroses

Paul Cézanne

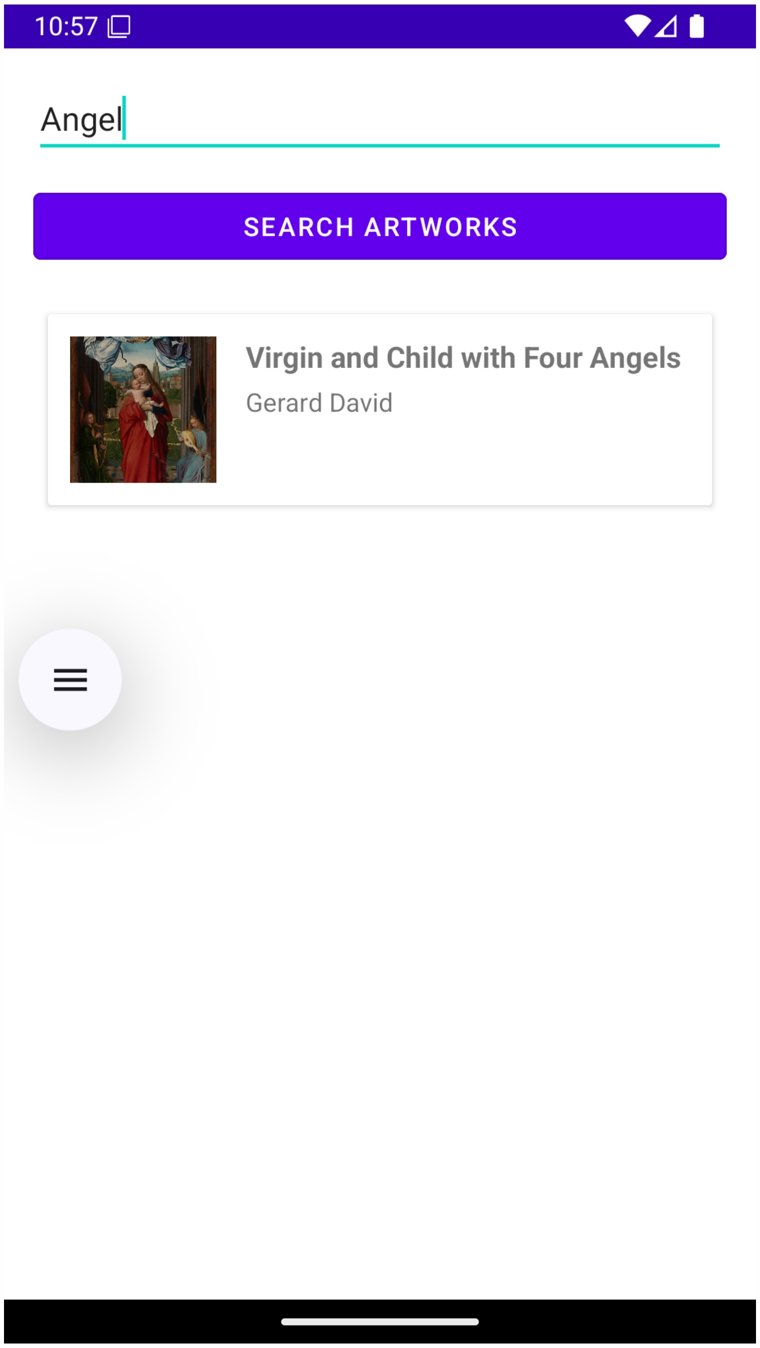
If cannot find artwork, it will shows 'No artwoks found'



No artworks found
Try a different search term

f. Is repeatable ✅

People can search another artwork and click search.



2. Implement a web service

The name of my web service file is **Project4-back**

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

In my web app project:

Model: ArtworkService.java manages the data processing logic including parsing artwork data from the Met Museum API and storing data in MongoDB. Additionally, **MongoDBClient.java** handles MongoDB database connections, provides database instance access methods, and implements the singleton pattern to ensure efficient database connectivity.

View: The application includes **index.jsp** for welcome page and **dashboard.jsp** for dashboard page.

Controller: ArtworkSearchServlet.java handles artwork search requests, interacts with the external Met Museum API, and transforms requests into appropriate model operations. **DashboardServlet.java** and **TestServlet.java** handles dashboard and test request separately.

b. Receives an HTTP request from the native Android application ✅

ArtworkSearchServlet processes HTTP GET request with the argument "q" containing the search term. The server handles this request in the "/api/search" endpoint handler where it parses the query parameter and prepares to process the search.

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    // Record the start time of the request
    long requestStartTime = System.currentTimeMillis();

    // Parse query parameter
    String searchQuery = request.getParameter("q");
    if (searchQuery == null || searchQuery.isEmpty()) {
        // Handle invalid input
        response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
        // Error handling code...
    }

    // Process valid request...
}
```

This servlet validates inputs, handles errors gracefully, and responds with appropriate HTTP status codes.

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

My web service implements a sophisticated search flow that communicates with the Metropolitan Museum of Art API to find relevant artworks. The `ArtworkSearchServlet` handles the core business logic:

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    // Validate input and prepare request
    String searchQuery = request.getParameter("q");

    // Execute two-step API interaction process
    Integer objectId = fetchFirstValidObjectId(searchQuery);  // First API call
    ArtworkService.ArtworkResponse artwork = fetchSingleArtwork(objectId);  // Second
API call

    // Format and return response to Android client
    String jsonResponse = GSON.toJson(artwork);
    response.getWriter().write(jsonResponse);

    // Log activity for analytics
    LoggingService.logSearchActivity(request, searchQuery, ...);
}
```

The service first fetches matching artwork IDs from the Met Museum API as JSON data, then retrieves detailed JSON information about specific artworks. Using the Google GSON library, the application parses these JSON responses, extracts only relevant fields (title, artist, image URL), and transforms them into a simplified JSON format optimized for mobile consumption. This JSON processing pipeline ensures efficient data transfer while maintaining all essential artwork information for a seamless user experience.

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

The server formats the response to the mobile application in a simple JSON structure containing only the necessary fields:

- objectID: The unique identifier for the artwork
- title: The title of the artwork
- culture: The cultural context of the artwork (if available)
- imageUrl: The URL to the artwork image
- artist: The name of the artist
- date: The date of the artwork

ArtworkService.parseArtwork ensures only the needed fields are extracted from the Met Museum API response:

```
public static ArtworkResponse parseArtwork(String json) {
        try {
            JsonObject obj = JsonParser.parseString(json).getAsJsonObject();
            ArtworkResponse response = new ArtworkResponse();

            // Ensure necessary fields exist
            if (!obj.has("objectID")) {
                System.err.println("API response missing objectID field");
                return null;
            }

            response.objectID = getIntValue(obj, "objectID");
            response.title = getStringValue(obj, "title");

            // Validate if image URL is valid
            String imageUrl = getStringValue(obj, "primaryImageSmall");
            if (imageUrl == null || imageUrl.isEmpty()) {
                // Try alternative image
                imageUrl = getStringValue(obj, "primaryImage");
                if (imageUrl == null || imageUrl.isEmpty()) {
                    // Use placeholder image
                    imageUrl = "https://placehold.co/300x200?text=Sample&font=roboto";
                }
            }
            response.imageUrl = imageUrl;

            response.culture = getStringValue(obj, "culture");
            response.artist = getStringValue(obj, "artistDisplayName");
            response.date = getStringValue(obj, "objectDate");

            return response;
        } catch (Exception e) {
            System.err.println("Error parsing Met Museum API response: " +
e.getMessage());
            return null;
        }
    }
```

This approach ensures the Android application receives only the data it needs to display, without having to perform additional processing, thus optimizing both bandwidth usage and client-side performance.

I use Servlets, not JAX-RS, for my web services.

4. Log useful information ✅

6 pieces of information is logged for each request/reply with the mobile phone, without logging data from interactions from the operations dashboard.

1. **Request IP Address** : I log the client's IP address to understand where, and from what kind of devices requests are made. This data is essential for diagnosing client-side issues, detecting potential abuse, and analyzing device-specific usage patterns.
2. **Search keywords entered by users**: I store the exact search terms that users input to analyze popular searches, identify trending topics, and improve search functionality based on common user interests.
3. **Artwork title of search results**: By logging the titles of artworks returned in search results, I can track which pieces are frequently appearing in searches, helping to understand what content is most relevant to users.
4. **Artist information from search results**: Artist names are logged to track which artists are most frequently searched for or appear in results, providing insights into user preferences and potentially informing future content curation.
5. **Timestamp of Each Search**: Every request is timestamped to enable chronological analysis of user activity. This helps uncover peak search hours, daily usage patterns, and seasonal trends in art exploration.

6. **System Response Time**: I record the server's response time for each request to monitor system performance. This information aids in identifying latency bottlenecks and optimizing backend processes to ensure a fast and smooth user experience.

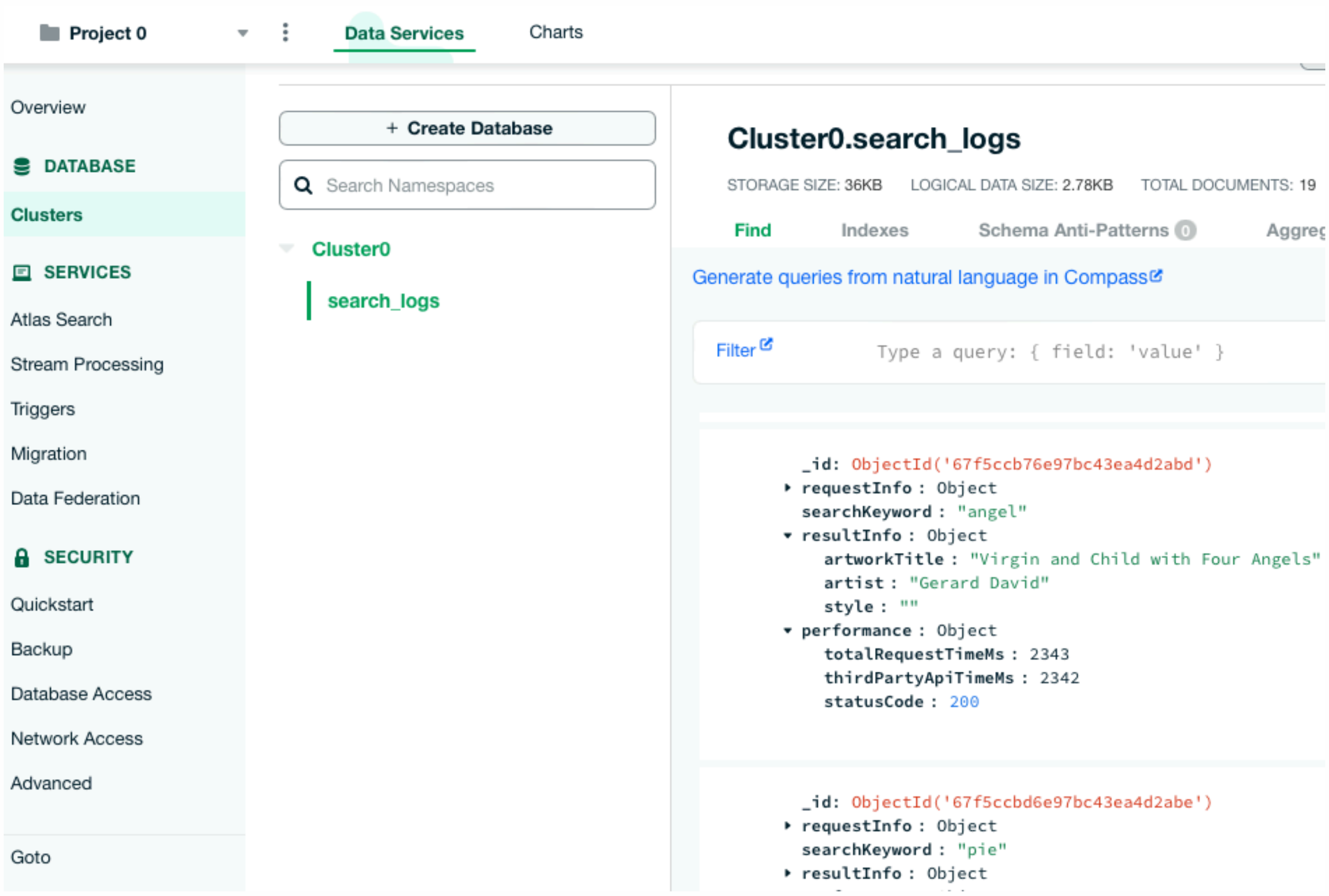
```java
public class LoggingService {

    /**
     * Log details about the search request, Met Museum API interaction, and response
     */
    public static void logSearchActivity(HttpServletRequest request, String searchQuery,
                                         long requestStartTime, long thirdPartyApiStartTime,
                                         long thirdPartyApiEndTime, ArtworkService.ArtworkResponse artwork,
                                         int statusCode) {

        // Create a separate thread for logging to prevent blocking the API
        Thread loggingThread = new Thread(() -> {
            try {
                // Get MongoDB collection
                MongoCollection<Document> logsCollection = MongoDBClient.getCollection("search_logs");

                // Calculate timing information
                long totalRequestTime = System.currentTimeMillis() - requestStartTime;
                long thirdPartyApiTime = thirdPartyApiEndTime - thirdPartyApiStartTime;

                // 1. Request metadata from the mobile phone
                Document requestInfo = new Document()
                        .append("timestamp", new Date(requestStartTime))
                        .append("clientIp", request.getRemoteAddr())
                        .append("userAgent", request.getHeader("User-Agent"))
                        .append("httpMethod", request.getMethod());

                // 2. User search keyword
                // 6. Search timestamp (already included in requestInfo)

                // 3-4-5. Result information (artwork title, artist, style)
                Document resultInfo = new Document();
                if (artwork != null) {
                    resultInfo.append("artworkTitle", artwork.title)  // 3. Artwork title
                            .append("artist", artwork.artist)        // 4. Artist information
                            .append("style", artwork.culture);       // 5. Artwork style
                } else {
                    resultInfo.append("artworkTitle", "No results found")
                            .append("artist", "N/A")
                            .append("style", "N/A");
                }

                // Performance metrics
                Document performance = new Document()
                        .append("totalRequestTimeMs", totalRequestTime)
                        .append("thirdPartyApiTimeMs", thirdPartyApiTime)
                        .append("statusCode", statusCode);

                // Create complete log document with all 6 required data points
                Document logEntry = new Document()
                        .append("requestInfo", requestInfo)               // Contains #1 and #6
                        .append("searchKeyword", searchQuery)             // #2
                        .append("resultInfo", resultInfo)                 // Contains #3, #4, and #5
                        .append("performance", performance);

                // Try to insert with unacknowledged write concern for better performance
                logsCollection.withWriteConcern(WriteConcern.UNACKNOWLEDGED).insertOne(logEntry);

                System.out.println("Search log recorded: " + searchQuery);
            } catch (Exception e) {
                System.err.println("Failed to log search: " + e.getMessage());
                e.printStackTrace();
                // Don't let this affect the main application
            }
        });

        // Start logging in background without waiting for it to complete
        loggingThread.setDaemon(true); // Don't let logging threads prevent app shutdown
        loggingThread.start();
    }
}
```

5. Store the log information in a database ✅

The web service can connect, store, and retrieve information from a MongoDB database in the cloud. Using **LoggingService.java** and **MongoDBClient.java**

My Atlas connection string with the three shards

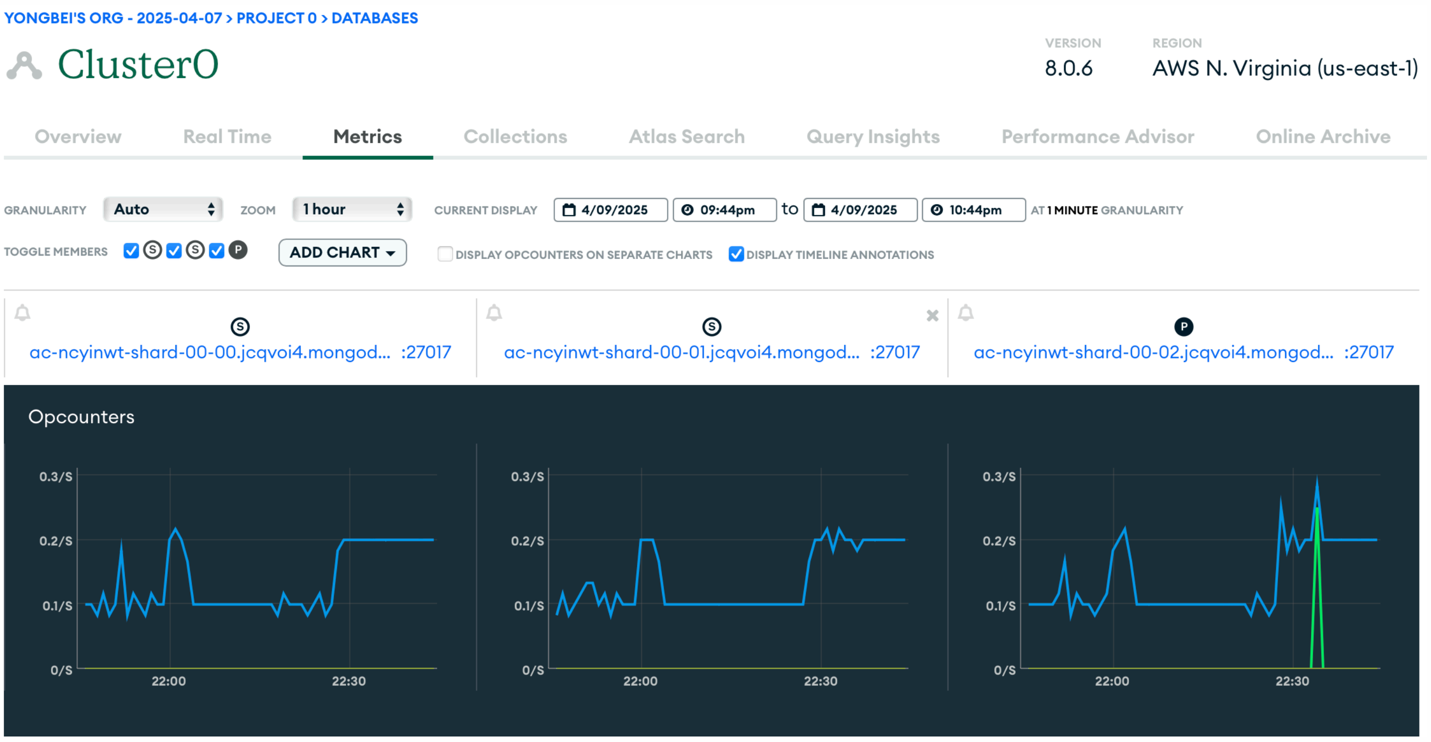
```
"mongodb://Aubrey:Aubrey66@ac-ncyinwt-shard-00-00.jcqvoi4.mongodb.net:27017,ac-ncyinwt-
shard-00-01.jcqvoi4.mongodb.net:27017,ac-ncyinwt-shard-00-
02.jcqvoi4.mongodb.net:27017/Cluster0?
w=majority&retryWrites=true&tls=true&authMechanism=SCRAM-SHA-1&authSource=admin";
```



Cluster0.search_logs

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 2.78KB TOTAL DOCUMENTS: 19

```
_id: ObjectId('67f5ccb76e97bc43ea4d2abd')
▸ requestInfo : Object
  searchKeyword : "angel"
▾ resultInfo : Object
    artworkTitle : "Virgin and Child with Four Angels"
    artist : "Gerard David"
    style : ""
▾ performance : Object
    totalRequestTimeMs : 2343
    thirdPartyApiTimeMs : 2342
    statusCode : 200

_id: ObjectId('67f5ccbd6e97bc43ea4d2abe')
▸ requestInfo : Object
  searchKeyword : "pie"
▸ resultInfo : Object
```


YONGBEI'S ORG - 2025-04-07 > PROJECT 0 > DATABASES

Cluster0

VERSION 8.0.6 REGION AWS N. Virginia (us-east-1)

GRANULARITY Auto ZOOM 1 hour CURRENT DISPLAY 4/09/2025 09:44pm to 4/09/2025 10:44pm AT 1 MINUTE GRANULARITY

TOGGLE MEMBERS ☑ S ☑ S ☑ P

☐ DISPLAY OPCOUNTERS ON SEPARATE CHARTS ☑ DISPLAY TIMELINE ANNOTATIONS

ac-ncyinwt-shard-00-00.jcqvoi4.mongod... :27017

ac-ncyinwt-shard-00-01.jcqvoi4.mongod... :27017

ac-ncyinwt-shard-00-02.jcqvoi4.mongod... :27017

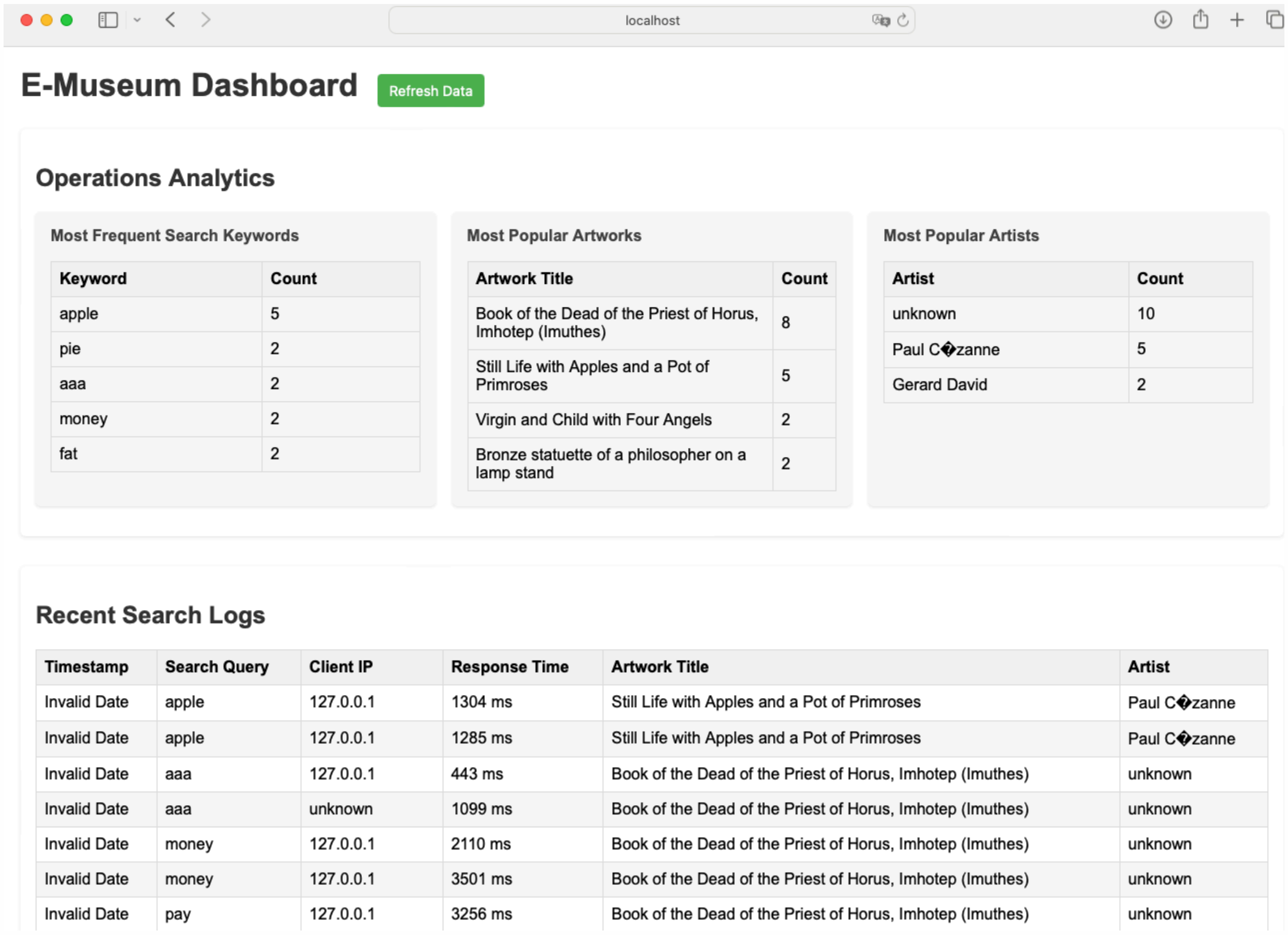
6. Display operations analytics and full logs on a web-based dashboard ✅

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

b. The dashboard displays at least 3 interesting operations analytics.

c. The dashboard displays formatted full logs.

> http://localhost:8080/dashboard



E-Museum Dashboard

Operations Analytics

Most Frequent Search Keywords

Keyword	Count
apple	5
pie	2
aaa	2
money	2
fat	2

Most Popular Artworks

Artwork Title	Count
Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	8
Still Life with Apples and a Pot of Primroses	5
Virgin and Child with Four Angels	2
Bronze statuette of a philosopher on a lamp stand	2

Most Popular Artists

Artist	Count
unknown	10
Paul C�zanne	5
Gerard David	2

Recent Search Logs

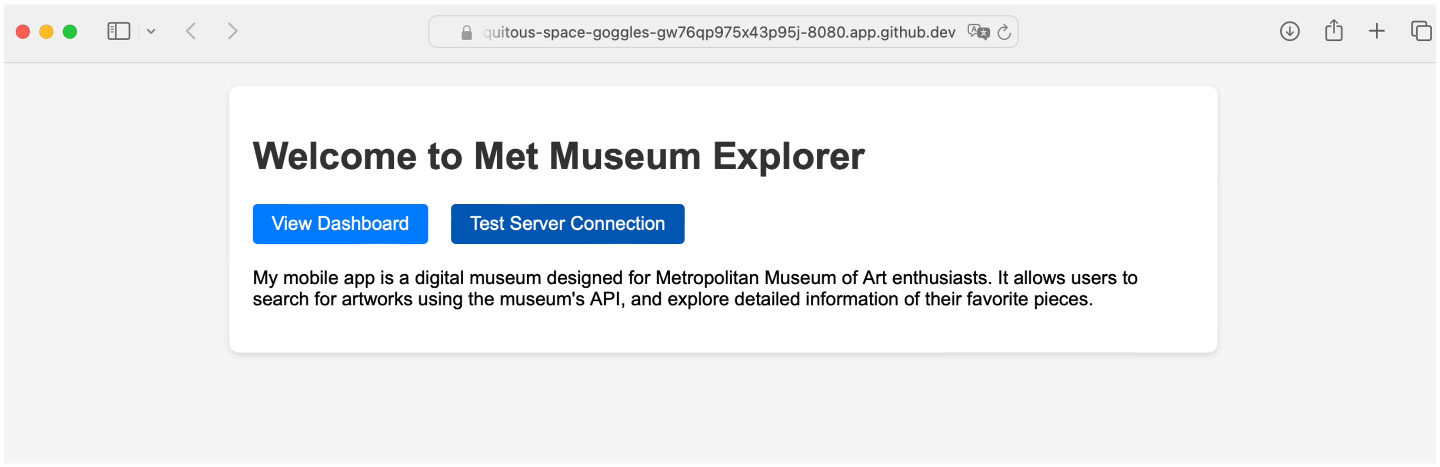
Timestamp	Search Query	Client IP	Response Time	Artwork Title	Artist
Invalid Date	apple	127.0.0.1	1304 ms	Still Life with Apples and a Pot of Primroses	Paul C�zanne
Invalid Date	apple	127.0.0.1	1285 ms	Still Life with Apples and a Pot of Primroses	Paul C�zanne
Invalid Date	aaa	127.0.0.1	443 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
Invalid Date	aaa	unknown	1099 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
Invalid Date	money	127.0.0.1	2110 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
Invalid Date	money	127.0.0.1	3501 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
Invalid Date	pay	127.0.0.1	3256 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown

I've set up a welcome page (as shown in the next picture), where clicking "View Dashboard" navigates to `/dashboard`. The dashboard features three operations analytics cards, which respectively display the most frequently used search keywords, the most popular artworks, and the most popular artists. Below that, a table provides detailed, formatted full logs.

7 Deploy to Github ✅

1 codespace page ✅

Taking an example of codespace *ubiquitous space goggles*.

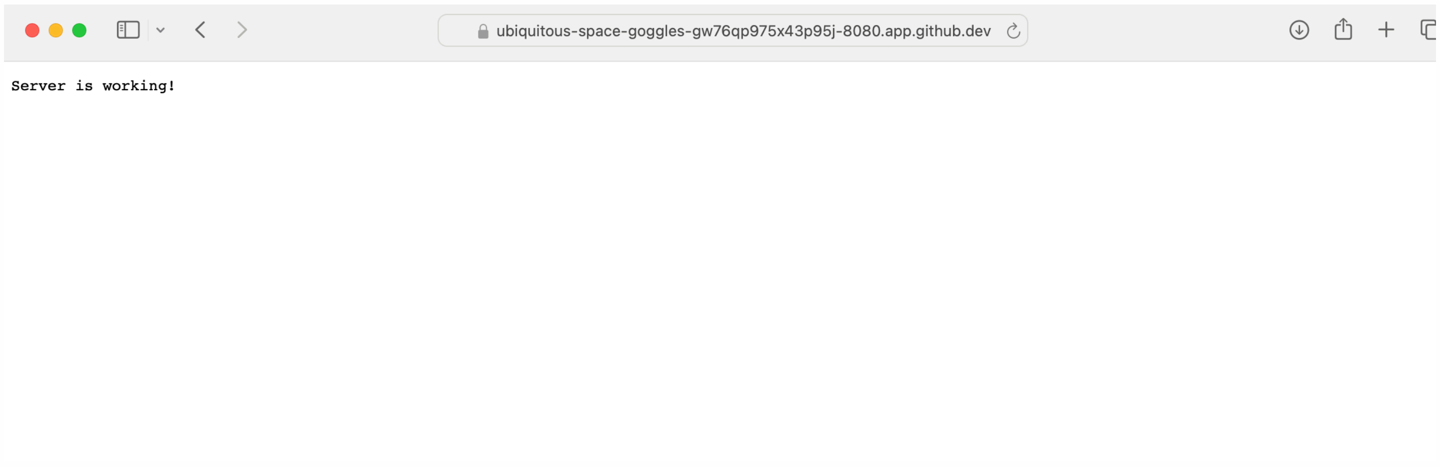


Welcome to Met Museum Explorer

My mobile app is a digital museum designed for Metropolitan Museum of Art enthusiasts. It allows users to search for artworks using the museum's API, and explore detailed information of their favorite pieces.

Click test server connection button:

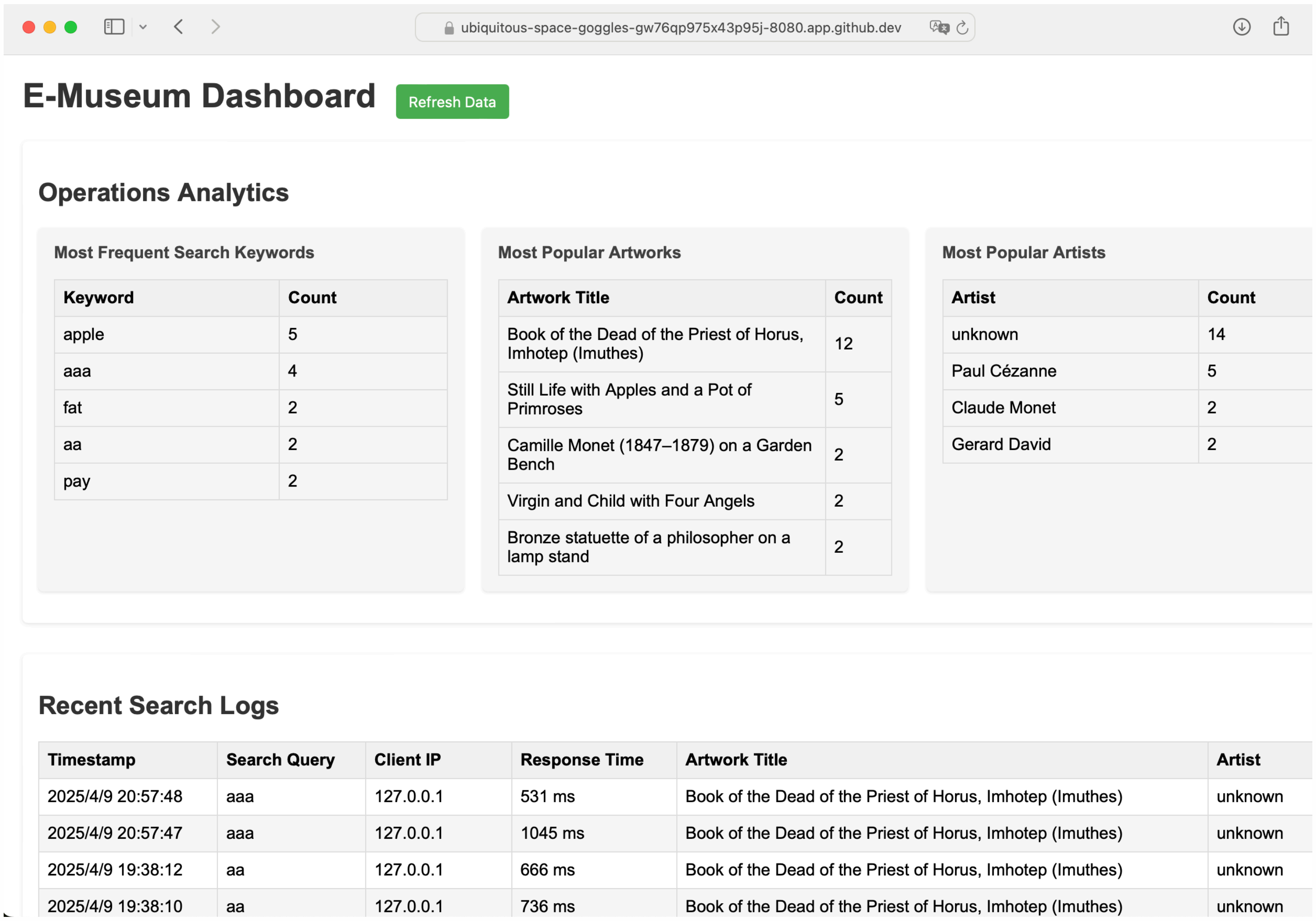
It shows the server is working.


ubiquitous-space-goggles-gw76qp975x43p95j-8080.app.github.dev

```
Server is working!
```

Click view dashboard button:

I allowed any IP Address to visit my Database in Atlas setting. So we can see dashboard as follows:



ubiquitous-space-goggles-gw76qp975x43p95j-8080.app.github.dev

E-Museum Dashboard

Operations Analytics

Most Frequent Search Keywords

Keyword	Count
apple	5
aaa	4
fat	2
aa	2
pay	2

Most Popular Artworks

Artwork Title	Count
Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	12
Still Life with Apples and a Pot of Primroses	5
Camille Monet (1847–1879) on a Garden Bench	2
Virgin and Child with Four Angels	2
Bronze statuette of a philosopher on a lamp stand	2

Most Popular Artists

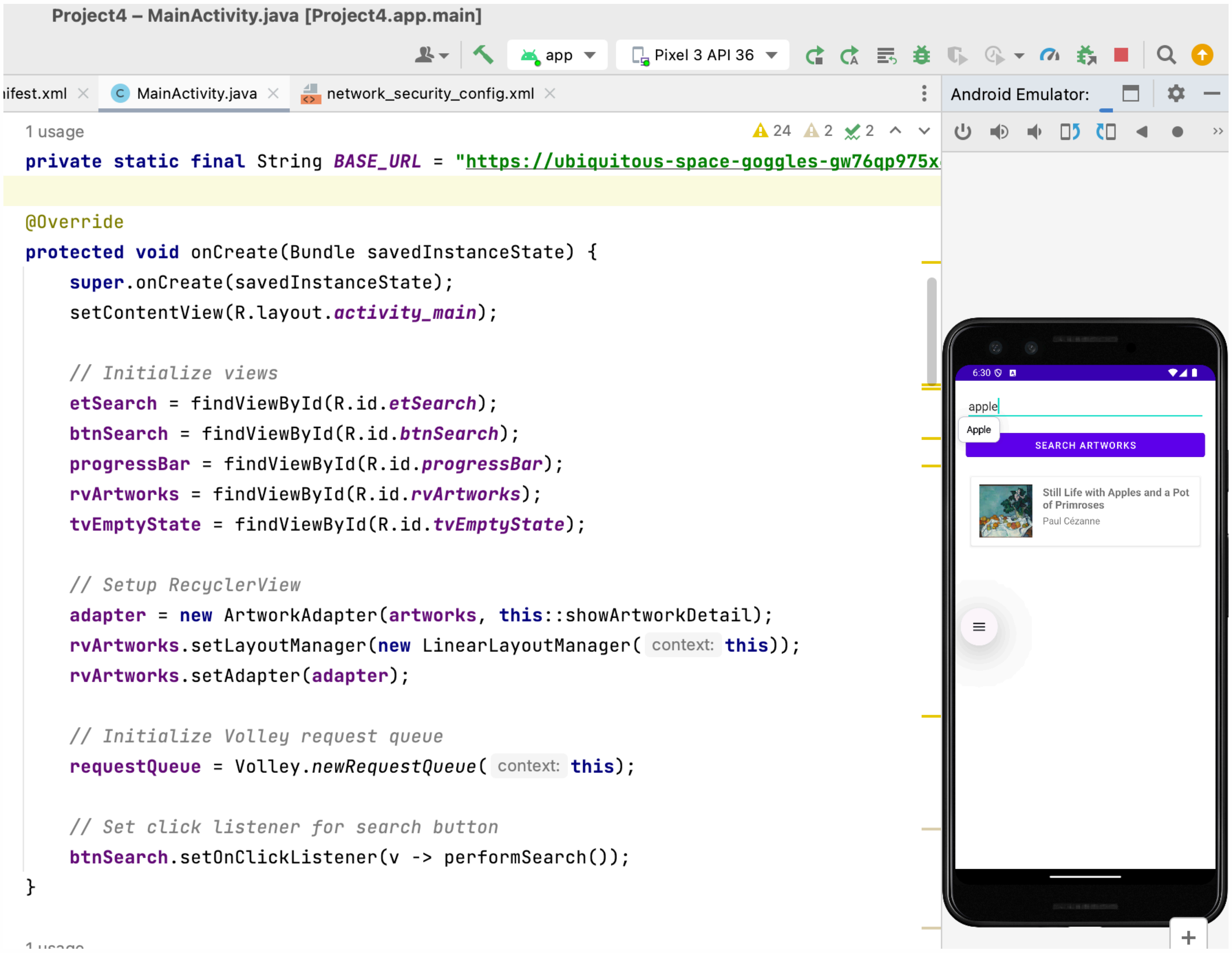
Artist	Count
unknown	14
Paul Cézanne	5
Claude Monet	2
Gerard David	2

Recent Search Logs

Timestamp	Search Query	Client IP	Response Time	Artwork Title	Artist
2025/4/9 20:57:48	aaa	127.0.0.1	531 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
2025/4/9 20:57:47	aaa	127.0.0.1	1045 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
2025/4/9 19:38:12	aa	127.0.0.1	666 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown
2025/4/9 19:38:10	aa	127.0.0.1	736 ms	Book of the Dead of the Priest of Horus, Imhotep (Imuthes)	unknown

2 Android app built on codespace ✅

The app runs smoothly when change BASE_URL to the codespaceurl


Project4 – MainActivity.java [Project4.app.main]
app
Pixel 3 API 36
ifest.xml
MainActivity.java
network_security_config.xml
Android Emulator:
24 2 2
6:30
apple
Apple
SEARCH ARTWORKS
Still Life with Apples and a Pot of Primroses
Paul Cézanne

```java
1 usage
private static final String BASE_URL = "https://ubiquitous-space-goggles-gw76qp975x

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Initialize views
    etSearch = findViewById(R.id.etSearch);
    btnSearch = findViewById(R.id.btnSearch);
    progressBar = findViewById(R.id.progressBar);
    rvArtworks = findViewById(R.id.rvArtworks);
    tvEmptyState = findViewById(R.id.tvEmptyState);

    // Setup RecyclerView
    adapter = new ArtworkAdapter(artworks, this::showArtworkDetail);
    rvArtworks.setLayoutManager(new LinearLayoutManager( context: this));
    rvArtworks.setAdapter(adapter);

    // Initialize Volley request queue
    requestQueue = Volley.newRequestQueue( context: this);

    // Set click listener for search button
    btnSearch.setOnClickListener(v -> performSearch());
}

1 usage
```