Distributed Application Writeup

Introduction

This document outlines how each of the specified requirements (1-7) for the distributed application project has been met. The application comprises a native Android frontend and a Java-based backend utilizing Servlets and MongoDB for data management. Additionally, a web-based dashboard has been implemented to facilitate logging, analysis, and reporting.

Requirement 1: Android Application

1.a. Incorporate at Least Three Different Kinds of Views

The Android application layout includes a variety of views to enhance user interaction and functionality:

- **TextView:** Displays informational text to the user.

  ```
  <TextView
      android:id="@+id/resultText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Result will appear here"
      android:textSize="16sp"
      android:paddingTop="16dp" />
  ```

- **EditText:** Allows users to input data, specifically the number they wish to inquire about.

  ```
  <EditText
      android:id="@+id/numberInput"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Enter a number (optional)"
      android:inputType="number" />
  ```

- **ImageView:** Displays an image related to the number facts, enhancing the visual appeal of the application.

  ```
  <ImageView
  android:id="@+id/imageView"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:contentDescription="@string/image_desc"
  app:srcCompat="@drawable/numbers" />
  ```

1.b. Requires Input from the User

The application requires user input a number to fetch facts about. The user can choose the fact type from a dropdown menu and click the "Fetch Fact" button to retrieve the information. The user can also click the "Random" button to fetch fact about a random number.

1.c. Makes HTTP Requests

The application makes HTTP requests to the backend server to fetch number facts. The request is sent to `/numberfact` with the number and fact type as query parameters. The following code snippet demonstrates the HTTP request:

```
    // Perform the network request in a new thread
    new Thread(() -> {
        try {
            URL url = new URL(queryUrl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");

            // Get the response code
            int status = conn.getResponseCode();

            if (status == HttpURLConnection.HTTP_OK) {
                // Read the response
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()));
                String inputLine;
                StringBuilder content = new StringBuilder();

                while ((inputLine = in.readLine()) != null) {
                    content.append(inputLine);
                }
                in.close();

                // Update UI with the fetched data
                String jsonResponse = content.toString();
                runOnUiThread(() -> displayResult(jsonResponse));

            } else {
                runOnUiThread(() -> Toast.makeText(MainActivity.this,
                        "Error fetching data. Status code: " + status,
                        Toast.LENGTH_LONG).show());
            }

        } catch (IOException e) {
            e.printStackTrace();
            runOnUiThread(() -> Toast.makeText(MainActivity.this,
                    "Network error occurred.",
                    Toast.LENGTH_LONG).show());
        }
    }).start();
```

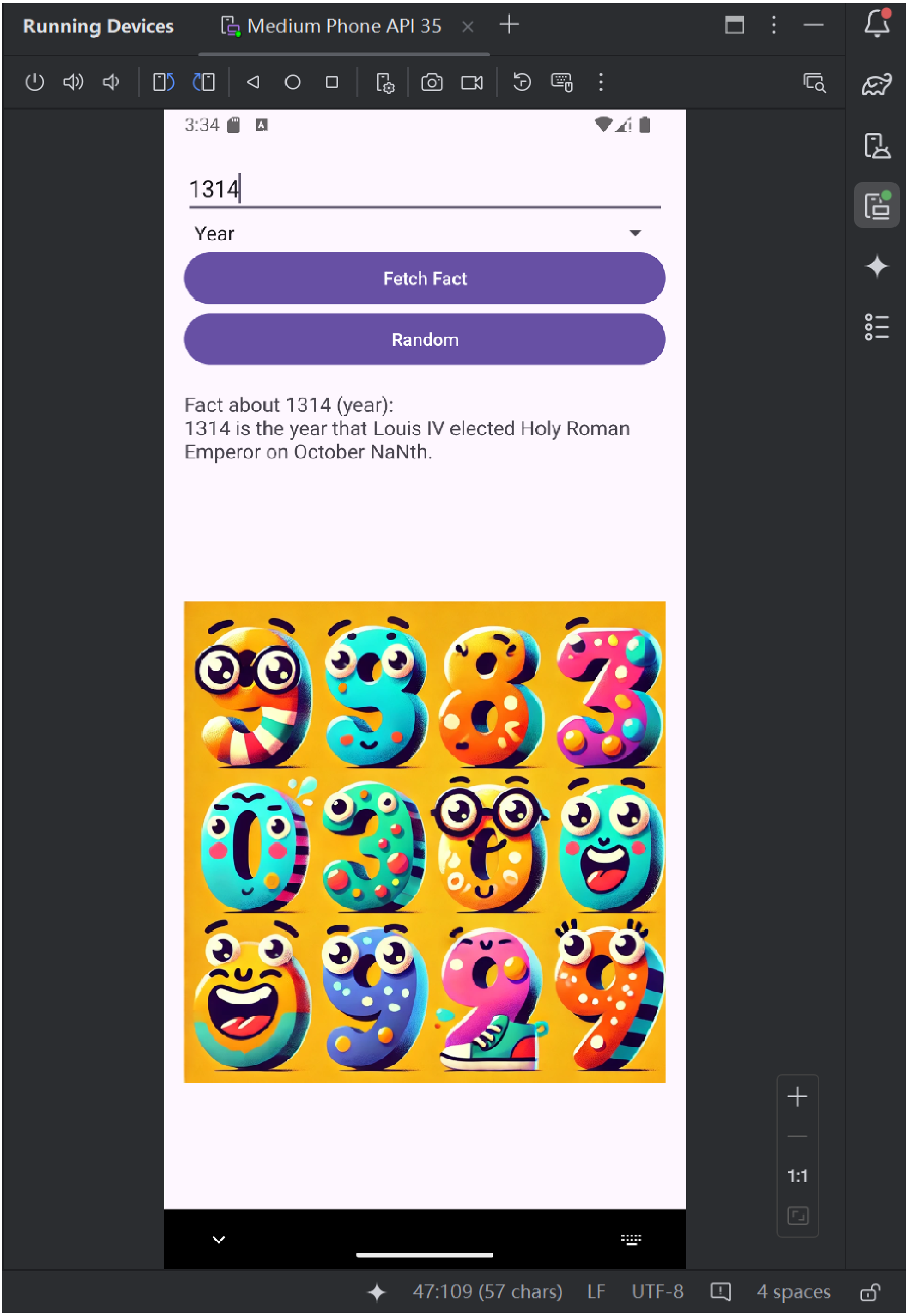
1.d. Receives JSON Formatted Reply from Web Service

The app receives json respones of the number fact from the backend server. An example of the json response is shown below:

```
{
    "date": "August 17",
    "text": "1984 is the year that Peru recognizes the Sahrawi Arab Democratic
Republic (SADR) on August 17th.",
    "number": 1984,
    "found": true,
    "type": "year"
}
```

1.e. Displays New Information to the User

After query the app load the facts and show the information to the user. Here is an example:



Fact about 1314 (year):
1314 is the year that Louis IV elected Holy Roman Emperor on October NaNth.

1.f. Repeatable

The user can repeatly enter numbers or just click the random button to get the facts about the numbers.

Requirement 2: Web Service

2.a. Implement a API

A Java Servlet (`NumberFactServlet`) has been developed to handle incoming HTTP GET requests from the Android application. The servlet processes the request and interacts with a third-party API to fetch number facts. The servlet listens on the `/numberfact` endpoint and accepts four query parameters: `number`, `type`, `model` and `android_version`.

2.b. Receives HTTP Requests

A HTTP Request with query parameters is sent from the Android application to the backend server. The servlet receives the request and processes the query parameters to fetch the number facts. Here is an example of the request:

```
https://localhost:8080/numberfact?
type=year&number=1984&model=Edge&android_version=10
```

2.c. Execute Bussiness Logic

The servlet executes the business logic to fetch the number facts from the third-party API. The servlet then constructs a JSON response and sends it back to the Android application. The following code snippet demonstrates the business logic:

```
        try {
            // Use the model to get the number fact
            String jsonResponse = model.getNumberFact(type, number);

            // Record reply time
            long replyTime = System.currentTimeMillis();

            // Log data
            model.logData(type, number, jsonResponse, modelParam, androidVersion,
userIP, requestTime, replyTime);

            // Write the response back to the client
            out.print(jsonResponse);
            out.flush();
        } catch (IOException e) {
            e.printStackTrace(); // Log the exception
            // Handle error when fetching data from Numbers API
            response.setStatus(HttpServletResponse.SC_BAD_GATEWAY);
            out.print("{\"error\":\"Unable to fetch data from Numbers API.\"}");
            out.flush();
        } catch (Exception e) {
```

```
        e.printStackTrace(); // Log the exception
        // Handle other exceptions
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        out.print("{\"error\":\"An unexpected error occurred: " + e.getMessage() +
"\"}");
        out.flush();
    }
```

2.d. Replies to the Android Application

After the business logic is executed, the servlet constructs a JSON response and sends it back to the Android application. The response is generate in the following lines in the previous provided code snippet:

```
    // Write the response back to the client
    out.print(jsonResponse);
    out.flush();
```

Requirement 4: Log Useful Information

The backend server logs useful information to a MongoDB database to facilitate analysis and reporting. The following information is logged for each request/response:

- Mobile Model
- Android Version
- User IP Address
- Request Time
- Reply Time
- Request Number: The number for which the fact is requested, if the user press the random button then the value is "Random".
- Response Number: The number for which the fact is returned (for random request we also log the number returned).
- Fact Type: The type of fact requested (trivia, year or math).

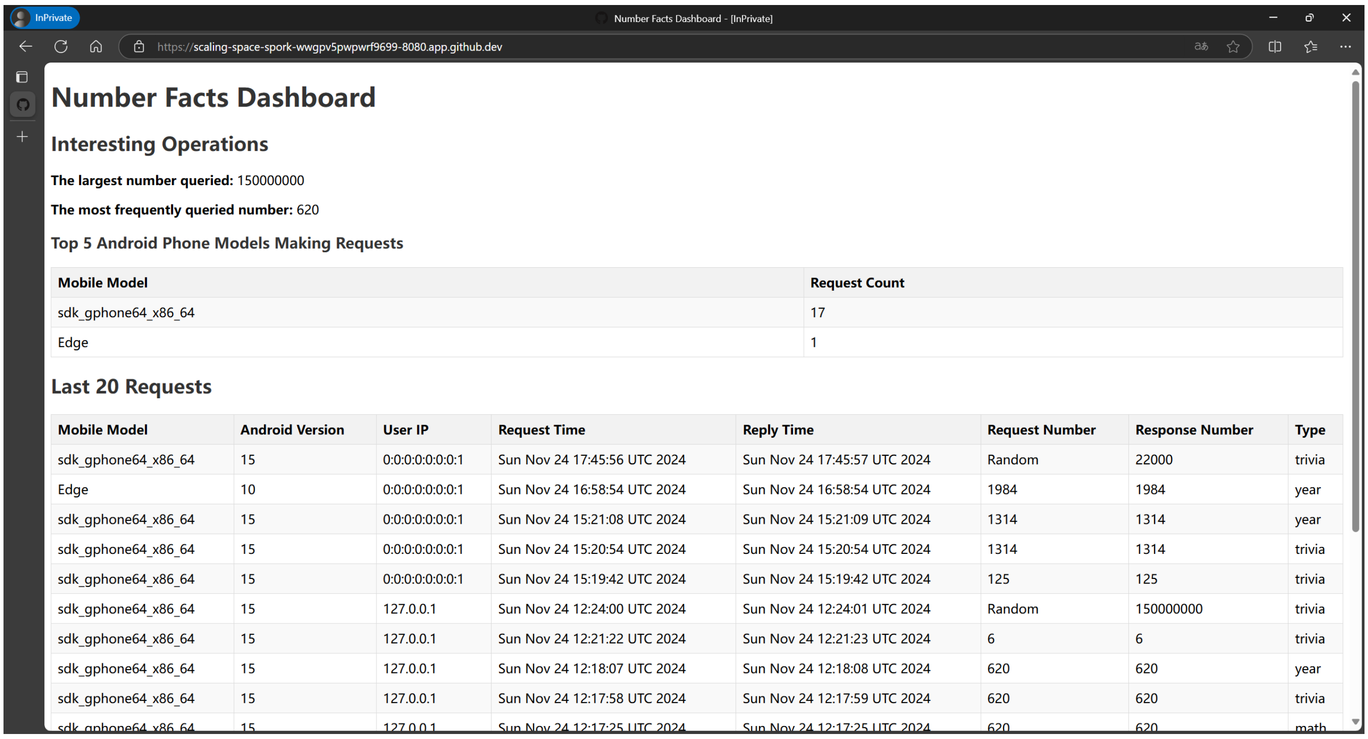
Requirement 5: Store the Log Information in a Database

All log information are stored in a MongoDB database. The database is hosted on a remote server and the backend server connects to it to store the log information. The following code snippet demonstrates the connection to the MongoDB database:

```
    // Initialize MongoDB client
    String uri =
"mongodb+srv://Cluster36659:vvcjES1WmL05IXif@cluster0.7iipo.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0";
    mongoClient = MongoClients.create(uri);
    database = mongoClient.getDatabase("numberfactsdb");
    collection = database.getCollection("logs");
```

Requirement 6: Display Dashboard

By this url you can access the dashboard: (https://scaling-space-spork-wwgpv5pwpwrf9699-8080.app.github.dev/). The dashboard shows the largest number queried, the most frequently queried number and the 5 top Android phone models makeing requests. You can see the dashboard as shown below:


InPrivate
Number Facts Dashboard - [InPrivate]
https://scaling-space-spork-wwgpv5pwpwrf9699-8080.app.github.dev

Number Facts Dashboard

Interesting Operations

The largest number queried: 150000000

The most frequently queried number: 620

Top 5 Android Phone Models Making Requests

Mobile Model	Request Count
sdk_gphone64_x86_64	17
Edge	1

Last 20 Requests

Mobile Model	Android Version	User IP	Request Time	Reply Time	Request Number	Response Number	Type
sdk_gphone64_x86_64	15	0:0:0:0:0:0:0:1	Sun Nov 24 17:45:56 UTC 2024	Sun Nov 24 17:45:57 UTC 2024	Random	22000	trivia
Edge	10	0:0:0:0:0:0:0:1	Sun Nov 24 16:58:54 UTC 2024	Sun Nov 24 16:58:54 UTC 2024	1984	1984	year
sdk_gphone64_x86_64	15	0:0:0:0:0:0:0:1	Sun Nov 24 15:21:08 UTC 2024	Sun Nov 24 15:21:09 UTC 2024	1314	1314	year
sdk_gphone64_x86_64	15	0:0:0:0:0:0:0:1	Sun Nov 24 15:20:54 UTC 2024	Sun Nov 24 15:20:54 UTC 2024	1314	1314	trivia
sdk_gphone64_x86_64	15	0:0:0:0:0:0:0:1	Sun Nov 24 15:19:42 UTC 2024	Sun Nov 24 15:19:42 UTC 2024	125	125	trivia
sdk_gphone64_x86_64	15	127.0.0.1	Sun Nov 24 12:24:00 UTC 2024	Sun Nov 24 12:24:01 UTC 2024	Random	150000000	trivia
sdk_gphone64_x86_64	15	127.0.0.1	Sun Nov 24 12:21:22 UTC 2024	Sun Nov 24 12:21:23 UTC 2024	6	6	trivia
sdk_gphone64_x86_64	15	127.0.0.1	Sun Nov 24 12:18:07 UTC 2024	Sun Nov 24 12:18:08 UTC 2024	620	620	year
sdk_gphone64_x86_64	15	127.0.0.1	Sun Nov 24 12:17:58 UTC 2024	Sun Nov 24 12:17:59 UTC 2024	620	620	trivia
sdk_gphone64_x86_64	15	127.0.0.1	Sun Nov 24 12:17:25 UTC 2024	Sun Nov 24 12:17:25 UTC 2024	620	620	math

Requirement 7: GitHub Codespaces

Just by following the instructions the codespaces was created and the project was run successfully.