1. Implement a native Android application
a. 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.
b. Requires input from the user
c. Makes an HTTP request (using an appropriate HTTP method) to your web service
d. Receives and parses an XML or JSON formatted reply from your web service
e. Displays new information to the user
f. Is repeatable (I.e. the user can repeatedly reuse the application without restarting it.)

a. contains three different kinds of views in my layout

```
<EditText
    android:id="@+id/tickerInput"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Stock Ticker"
    android:minHeight="48dp" />

<EditText
    android:id="@+id/dateInput"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Date (YYYY-MM-DD)"
    android:minHeight="48dp" />

<Button
    android:id="@+id/fetchButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Fetch Data" />

<TextView
    android:id="@+id/resultView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Results will appear here"
    android:paddingTop="16dp" />
```

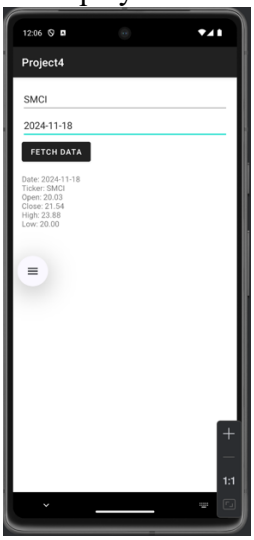
B.
Here is a screen shot for user to search the information of a ticker:

C.
The Android app sends a GET request to fetch stock data for a specific ticker and date.

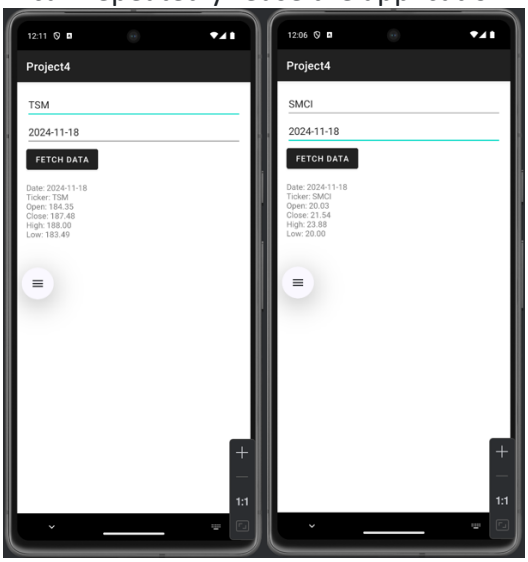
url:
```
http://100.110.161.104:8080/Project4Task2_war_exploded/stockdata?ticker=" +
ticker + "&date=" + date
```

D.
An example of the XML reply is:
```
<?xml version="1.0" encoding="UTF-8"?>
<StockData>
    <Ticker>DJT</Ticker>
    <Date>2024-11-11</Date>
    <Open>33.45</Open>
    <Close>33.41</Close>
    <High>34.4</High>
    <Low>31.62</Low>
</StockData>
```

e. Displays new information to the user



F. can repeatedly reuse the application without restarting it



2. Implement a web service
a. Implement a simple (can be a single path) API.
b. Receives an HTTP request from the native Android application
c. Executes business logic appropriate to your application. This includes fetching XML or JSON information from some 3rd party API and processing the response.
d. Replies to the Android application with an XML or JSON formatted response. The schema of the response can be of your own design.

a.
this part of the code implements a simple API:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    // Set response type
    response.setContentType("application/json");
    PrintWriter out = response.getWriter();
    ...
}
```

b.
this part of the code receives an HTTP request from the native Android application:

```
String ticker = request.getParameter("ticker");
String date = request.getParameter("date");
if (ticker == null || date == null || ticker.isEmpty() || date.isEmpty()) {
    response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
    out.write("{\"error\": \"Ticker and date parameters are required.\"}");
    return;
}
```

c.
this part of the code fetches XML or JSON information from some 3rd party API and processing the response.
fetching XML or JSON information from some 3rd party API and processing the response:

```
String polygonApiUrl = "https://api.polygon.io/v1/open-close/" + ticker + "/"
+ date + "?adjusted=true&apiKey=...";
URL url = new URL(polygonApiUrl);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
...
JSONObject polygonData = new JSONObject(jsonResponse.toString());

double open = polygonData.optDouble("open", -1);
double close = polygonData.optDouble("close", -1);
double high = polygonData.optDouble("high", -1);
double low = polygonData.optDouble("low", -1);
```

d.
this part of the code replies to the Android application with an XML or JSON formatted response:

```
JSONObject jsonResponseToApp = new JSONObject();
jsonResponseToApp.put("ticker", ticker);
jsonResponseToApp.put("date", date);
jsonResponseToApp.put("open", open);
jsonResponseToApp.put("close", close);
jsonResponseToApp.put("high", high);
jsonResponseToApp.put("low", low);

out.write(jsonResponseToApp.toString());
```

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.)

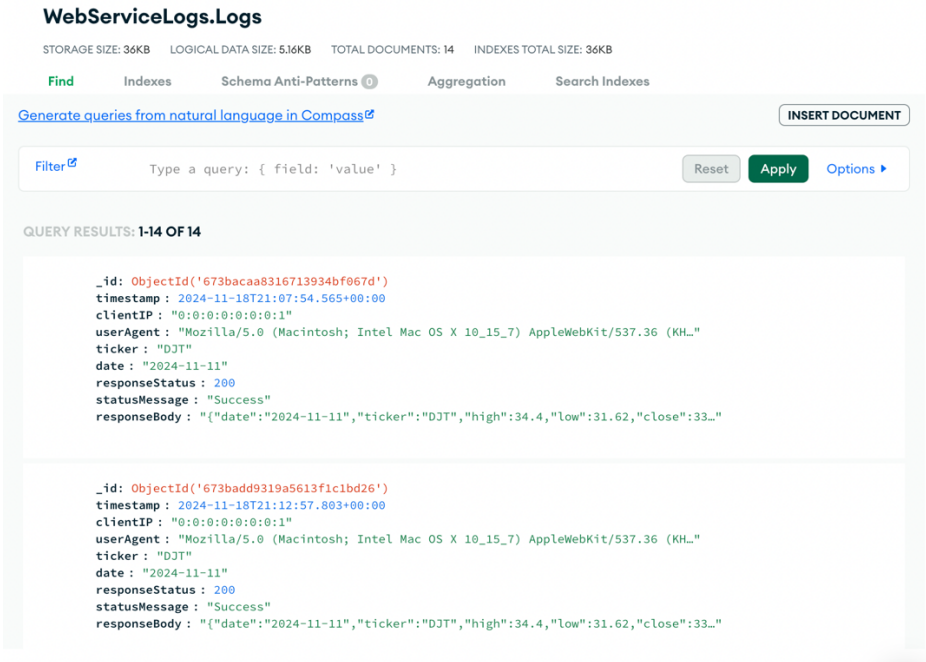
The following is an example of one log, it includes id, timestamp, clientIP, userAgent, ticker,date, responseStatus, statusMessage, responseBody:

{"_id":{"$oid":"673bacaa8316713934bf067d"},"timestamp":{"$date":{"$numberLong":"1731964074565"}},"clientIP":"0:0:0:0:0:0:0:1","userAgent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36","ticker":"DJT","date":"2024-11-11","responseStatus":{"$numberInt":"200"},"statusMessage":"Success","responseBody":"{\"date\":\"2024-11-11\",\"ticker\":\"DJT\",\"high\":34.4,\"low\":31.62,\"close\":33.41,\"open\":33.45}"}

Id: Unique identifier for the log entry in the MongoDB collection
timestamp: Records the exact time the log entry was created
clientIP: Logs the IP address of the client (mobile phone or other device) that sent the request.
userAgent: Logs details about the device or application making the request.
ticker: Represents the stock ticker symbol requested by the client.
date: The date for which the stock data is being requested.
responseStatus: Logs the HTTP status code sent back to the client.
statusMessage: A custom message that summarizes the status of the request processing.
responseBody: Logs the JSON data sent back to the client as the response.

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

The following screenshot from MongoDB shows that the web service can connect and store information from a MongoDB database in the cloud. And I am able to build a dashboard that contains all logs(see part 6), which means I can retrieve info from a MongoDB database in the cloud.



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.

a. http://localhost:8080/Project4Task2_war_exploded/dashboard
b. It displays 4 interesting operations analytics
c. The dashboard displays formatted full logs.

Web Service Logging and Analytics Dashboard

Analytics

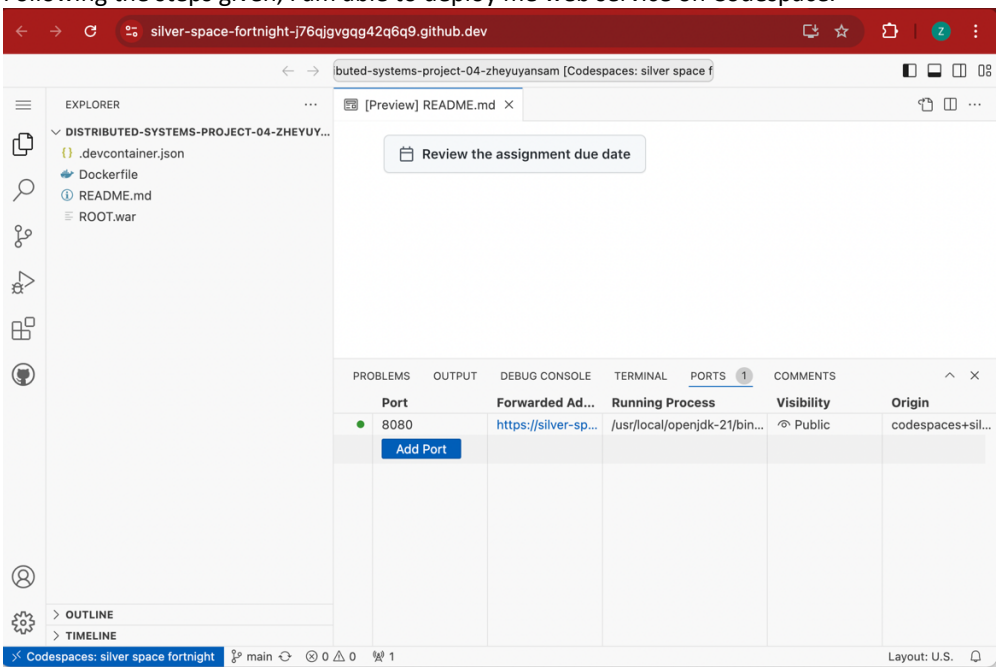
- Total Requests: 14
- Unique Tickers Queried: 7
- Unique Dates Queried: 4
- Unique IPs: 2

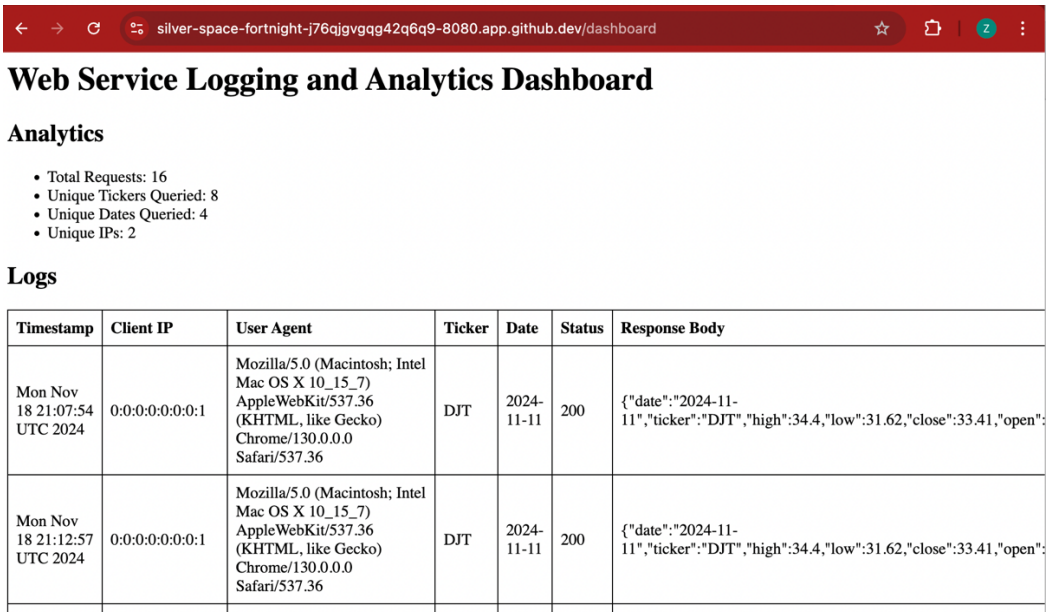
Logs

Timestamp	Client IP	User Agent	Ticker	Date	Status	Response Body
Mon Nov 18 16:07:54 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	DJT	2024-11-11	200	{"date":"2024-11-11","ticker":"DJT","high":34.4,"low":31.62,"close":33.41,"open":33.45}
Mon Nov 18 16:12:57 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	DJT	2024-11-11	200	{"date":"2024-11-11","ticker":"DJT","high":34.4,"low":31.62,"close":33.41,"open":33.45}
Mon Nov 18 16:13:03 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	DJT	2024-11-13	200	{"date":"2024-11-13","ticker":"DJT","high":31.22,"low":28.8,"close":28.93,"open":30.965}
Mon Nov 18 16:18:34 EST 2024	100.110.161.104	Dalvik/2.1.0 (Linux; U; Android 14; sdk_gphone64_arm64 Build/UE1A.230829.036.A4)	TSLA	2024-11-15	200	{"date":"2024-11-15","ticker":"TSLA","high":324.6799,"low":309.22,"close":320.72,"open":310.57}
Mon Nov 18 16:40:25 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	DJT	2024-11-11	200	{"date":"2024-11-11","ticker":"DJT","high":34.4,"low":31.62,"close":33.41,"open":33.45}
Mon Nov 18 16:40:36 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	tsla	2024-11-11	404	null
Mon Nov 18 16:40:42 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	TSLA	2024-11-11	200	{"date":"2024-11-11","ticker":"TSLA","high":358.64,"low":336,"close":350,"open":346.3}
Mon Nov 18 16:40:52 EST 2024	0:0:0:0:0:0:0:1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36	MSFT	2024-11-11	200	{"date":"2024-11-11","ticker":"MSFT","high":424.81,"low":416,"close":418.01,"open":422.515}
Mon Nov 18		Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)		2024-		{"date":"2024-11-

7. Deploy the web service to GitHub Codespaces

Following the steps given, I am able to deploy me web service on Codespace.





Then I changed the url on my Andriod Mobile to:

```
String urlString = "https://silver-space-fortnight-j76qjgvgqg42q6q9-8080.app.github.dev/stockdata?ticker=" + ticker + "&date=" + date;
```

And the mobile app is able to run