Name : KuanMing Hou

AndrewID : kuanminh

Name of API: US Bureau of Labor Statistics

https://www.bls.gov/developers/api_signature_v2.htmQ1.1 -

Name of application : Unemployment Rate Searcher

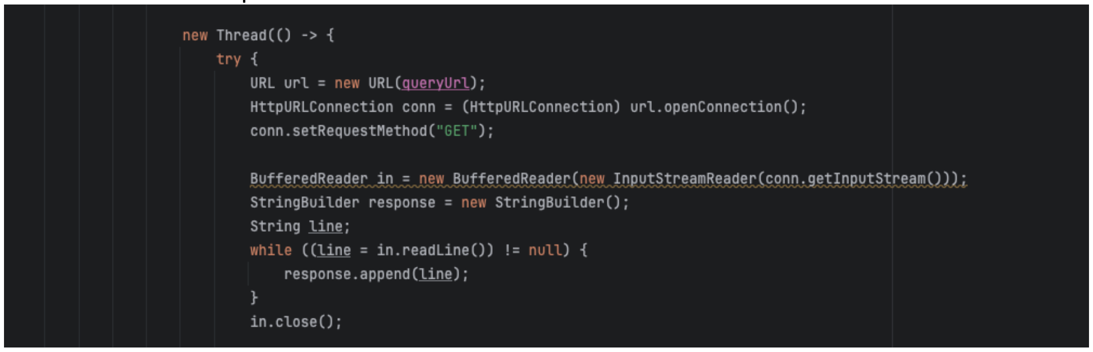
----------------------Requirements-----------------------

1. Implement a native Android application

a. Has at least three different kinds of Views in your Layout : I have EditText, Button, and TextView, see below screenshots.

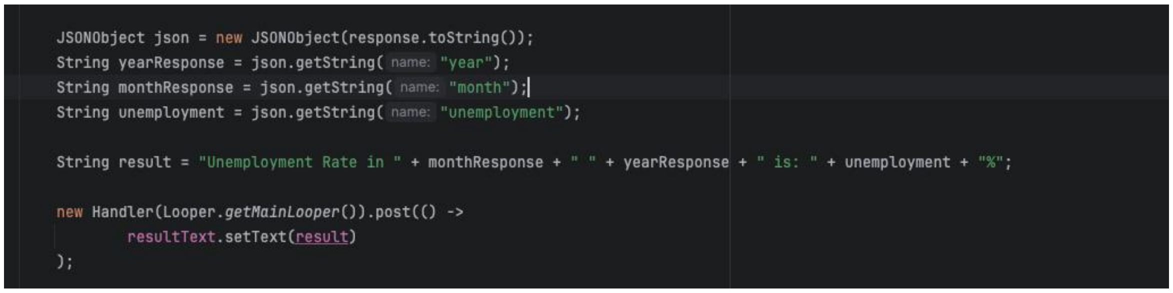
b. Requires input from the user : see below screenshots.

c. Makes an HTTP request



```
new Thread(() -> {
    try {
        URL url = new URL(queryUrl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        StringBuilder response = new StringBuilder();
        String line;
        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();
```

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



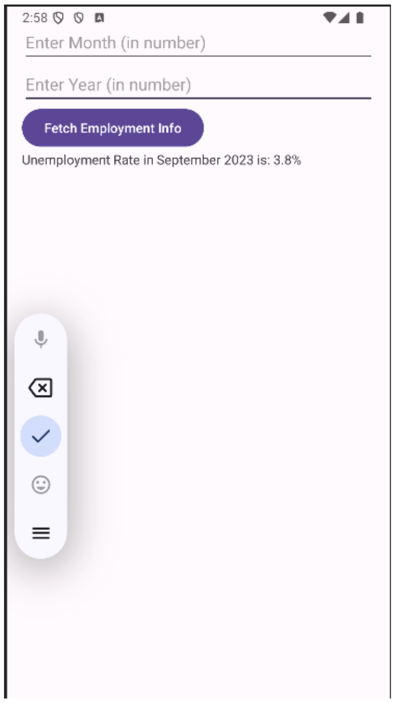
```
JSONObject json = new JSONObject(response.toString());
String yearResponse = json.getString( name: "year");
String monthResponse = json.getString( name: "month");
String unemployment = json.getString( name: "unemployment");

String result = "Unemployment Rate in " + monthResponse + " " + yearResponse + " is: " + unemployment + "%";

new Handler(Looper.getMainLooper()).post(() ->
        resultText.setText(result)
);
```

e. Displays new information to the user :see below screenshots.

f. Is repeatable : see below screenshots.

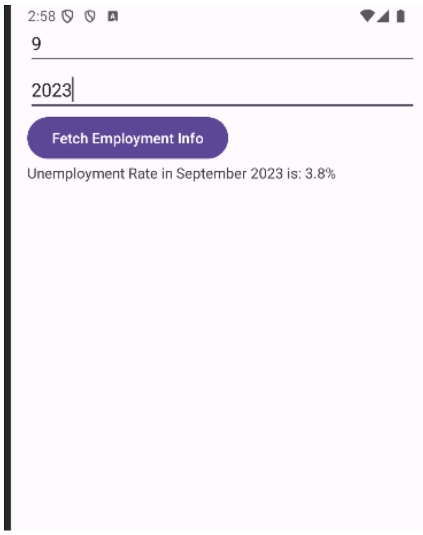
Demo Screenshots:

<Normal Page> (without input anything from user)

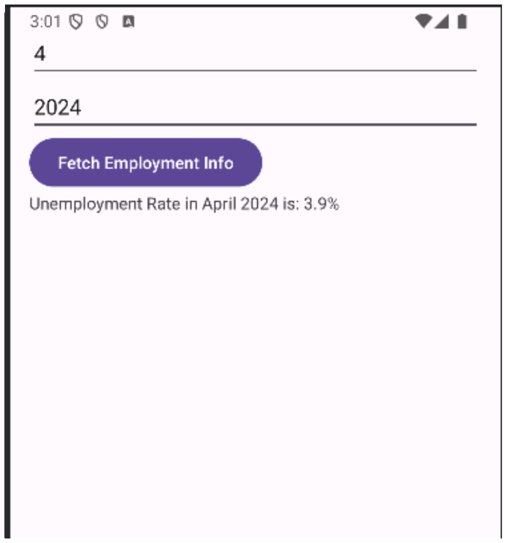


Unemployment Rate in September 2023 is: 3.8%

<Search Page> (User input year and month as below, and press the fetch button, no need to turn off application)



Unemployment Rate in September 2023 is: 3.8%

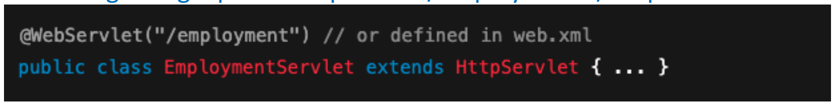


Unemployment Rate in April 2024 is: 3.9%

2. Implement a web service

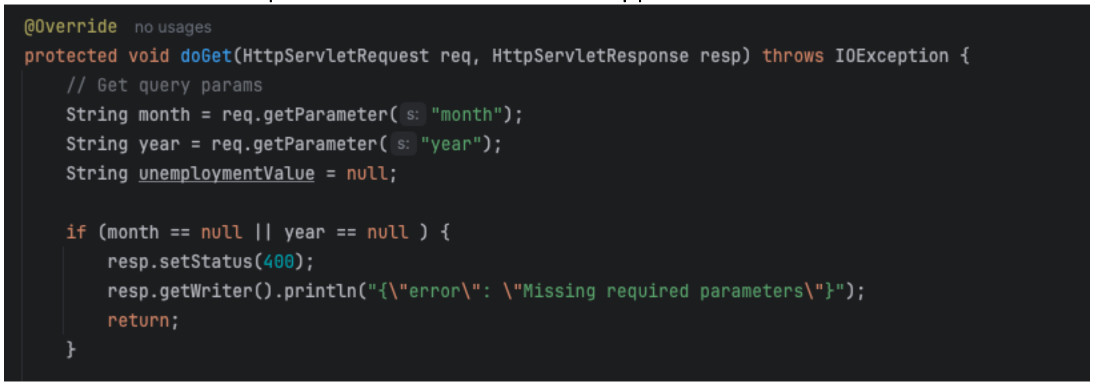
a. Implement a simple (can be a single path) API.

I am using a single path <url-pattern>/employment</url-pattern> that maps to



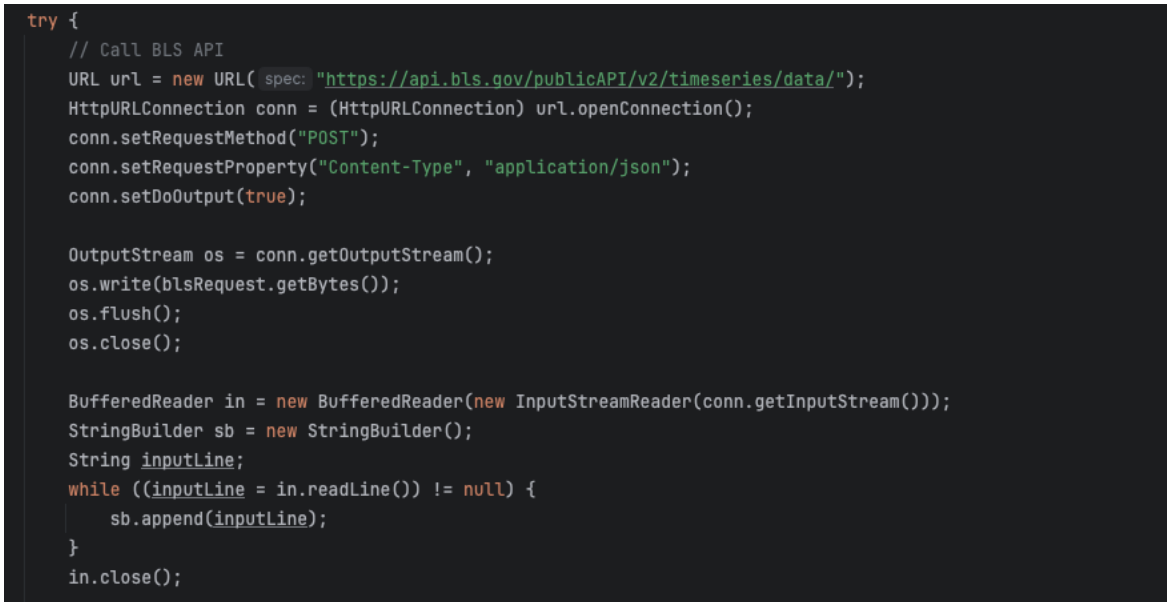
```
@WebServlet("/employment") // or defined in web.xml
public class EmploymentServlet extends HttpServlet { ... }
```

b. Receives an HTTP request from the native Android application



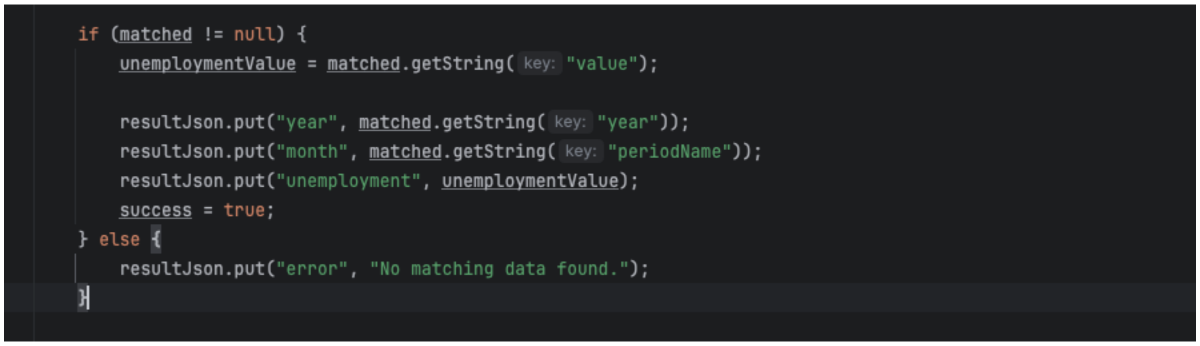
```
@Override  no usages
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    // Get query params
    String month = req.getParameter( s: "month");
    String year = req.getParameter( s: "year");
    String unemploymentValue = null;

    if (month == null || year == null ) {
        resp.setStatus(400);
        resp.getWriter().println("{\"error\": \"Missing required parameters\"}");
        return;
    }
```

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



```
try {
    // Call BLS API
    URL url = new URL( spec: "https://api.bls.gov/publicAPI/v2/timeseries/data/");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setDoOutput(true);

    OutputStream os = conn.getOutputStream();
    os.write(blsRequest.getBytes());
    os.flush();
    os.close();

    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    StringBuilder sb = new StringBuilder();
    String inputLine;
    while ((inputLine = in.readLine()) != null) {
        sb.append(inputLine);
    }
    in.close();
```

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



```
    if (matched != null) {
        unemploymentValue = matched.getString( key: "value");

        resultJson.put("year", matched.getString( key: "year"));
        resultJson.put("month", matched.getString( key: "periodName"));
        resultJson.put("unemployment", unemploymentValue);
        success = true;
    } else {
        resultJson.put("error", "No matching data found.");
    }
```

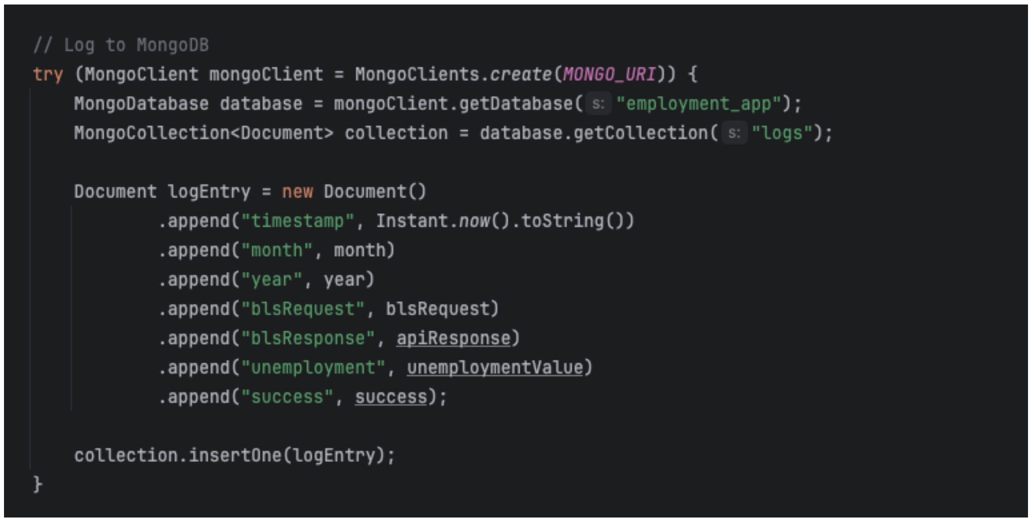
4. Log useful information

Below is the structure stored in MongoDB, there are 7 pieces of information

```
{
  "timestamp": "2025-04-09T03:51:32.893462097Z",
  "month": "12",
  "year": "2024",
  "blsRequest": "{...}",       // full request JSON to BLS API
  "blsResponse": "{...}",      // full raw BLS response JSON
  "unemployment": "4.1",       // unemployment value extracted
  "success": true              // whether parsing & matching succeeded
}
```

5. Store the log information in a database

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



```java
// Log to MongoDB
try (MongoClient mongoClient = MongoClients.create(MONGO_URI)) {
    MongoDatabase database = mongoClient.getDatabase( s: "employment_app");
    MongoCollection<Document> collection = database.getCollection( s: "logs");

    Document logEntry = new Document()
            .append("timestamp", Instant.now().toString())
            .append("month", month)
            .append("year", year)
            .append("blsRequest", blsRequest)
            .append("blsResponse", apiResponse)
            .append("unemployment", unemploymentValue)
            .append("success", success);

    collection.insertOne(logEntry);
}
```

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. see below screenshots.

b. The dashboard displays at least 3 interesting operations analytics. see below screenshots.

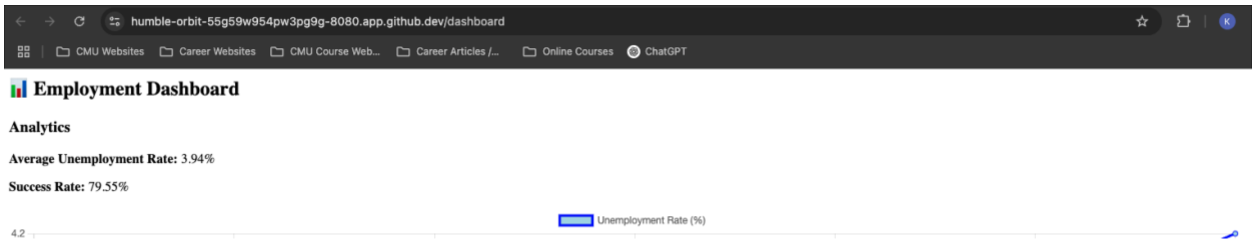
1. Display Average Unemployment Rate

2. Display success Rate for the response

3. Display the chart of the requested month and year of unemployment rates

c. The dashboard displays formatted full logs. see below screenshots.

<a. URL and analytics 1 & 2>



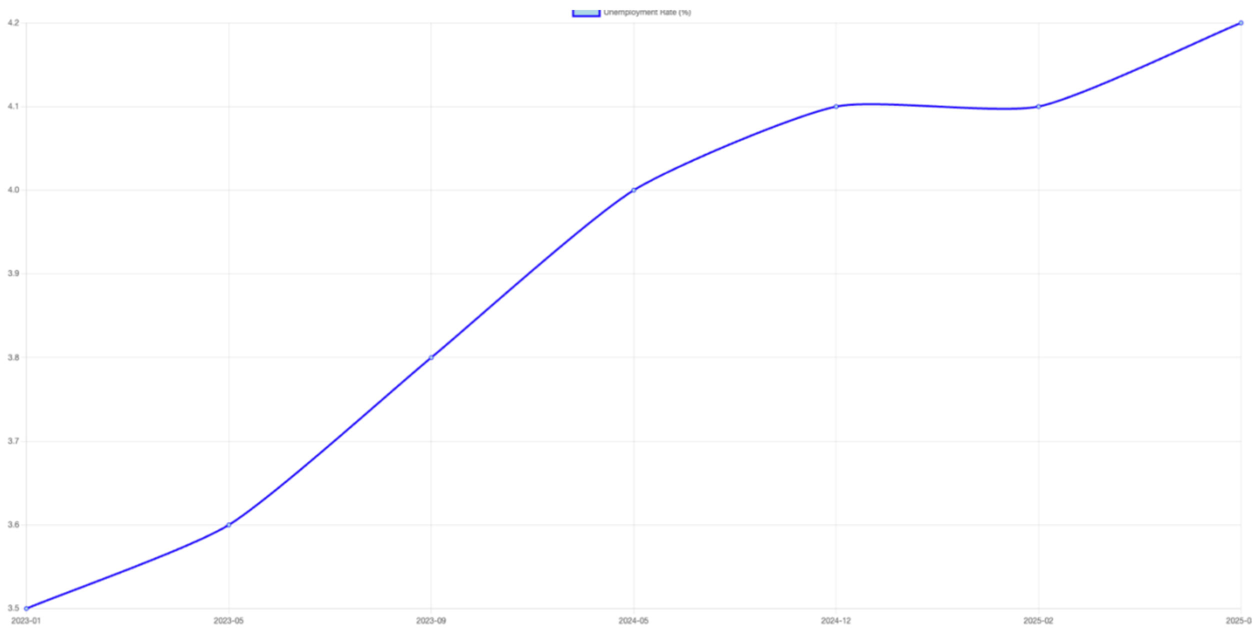
📊 Employment Dashboard

Analytics

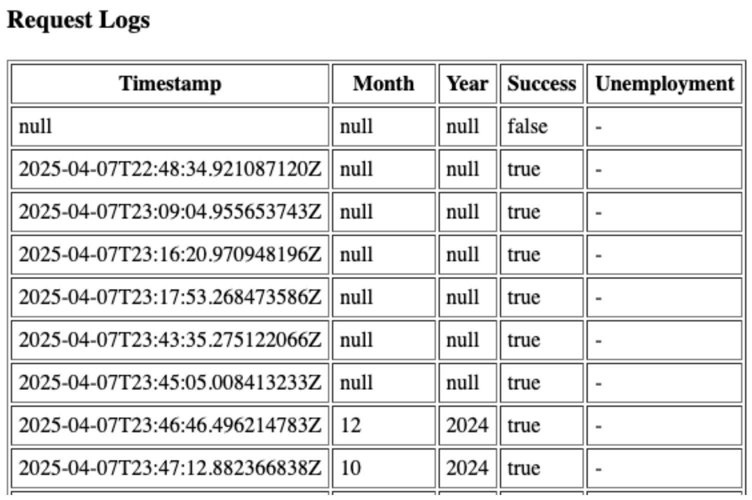
Average Unemployment Rate: 3.94%

Success Rate: 79.55%

<b. analytics 3 – the chart>

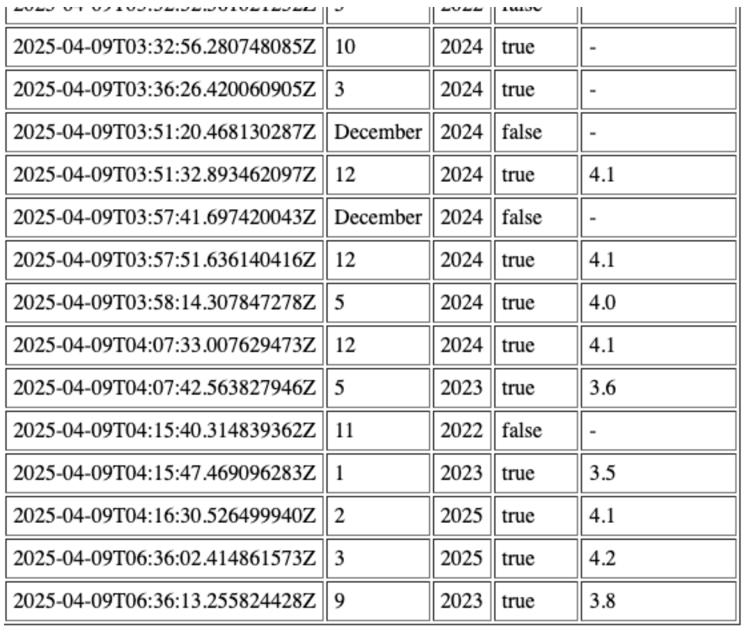


<c. – full logs>



Request Logs

Timestamp	Month	Year	Success	Unemployment
null	null	null	false	-
2025-04-07T22:48:34.921087120Z	null	null	true	-
2025-04-07T23:09:04.955653743Z	null	null	true	-
2025-04-07T23:16:20.970948196Z	null	null	true	-
2025-04-07T23:17:53.268473586Z	null	null	true	-
2025-04-07T23:43:35.275122066Z	null	null	true	-
2025-04-07T23:45:05.008413233Z	null	null	true	-
2025-04-07T23:46:46.496214783Z	12	2024	true	-
2025-04-07T23:47:12.882366838Z	10	2024	true	-



[illegible]	[illegible]	2022	false	
2025-04-09T03:32:56.280748085Z	10	2024	true	-
2025-04-09T03:36:26.420060905Z	3	2024	true	-
2025-04-09T03:51:20.468130287Z	December	2024	false	-
2025-04-09T03:51:32.893462097Z	12	2024	true	4.1
2025-04-09T03:57:41.697420043Z	December	2024	false	-
2025-04-09T03:57:51.636140416Z	12	2024	true	4.1
2025-04-09T03:58:14.307847278Z	5	2024	true	4.0
2025-04-09T04:07:33.007629473Z	12	2024	true	4.1
2025-04-09T04:07:42.563827946Z	5	2023	true	3.6
2025-04-09T04:15:40.314839362Z	11	2022	false	-
2025-04-09T04:15:47.469096283Z	1	2023	true	3.5
2025-04-09T04:16:30.526499940Z	2	2025	true	4.1
2025-04-09T06:36:02.414861573Z	3	2025	true	4.2
2025-04-09T06:36:13.255824428Z	9	2023	true	3.8

7. Deploy the web service to GitHub Codespaces

In order to deploy your web application to the cloud using Github Cloudspaces, first make sure you have the basics working.

a. Accept the Github Classroom Assignment that you have been given the URL for : done

b. Click the green <> Code dropdown button, select the Codespaces tab, then click on "Create codespace on master". done

c. Once the Codespace is running, the Terminal tab will show that Catalina (the Servlet container) is running. You should also see a "1" next to the Ports tab. Click on the Ports tab and you should see that port 8080 has been made available. done

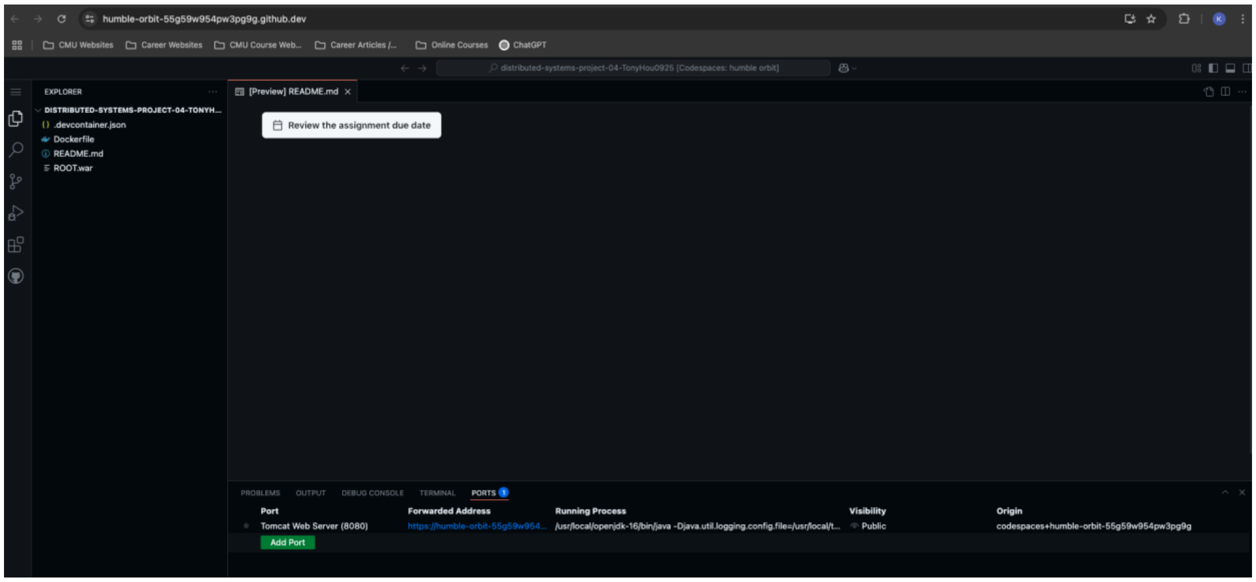
d. Mouse over the Local address item of the port 8080 line and you will find three icons. The leftmost is to copy the URL of your deployed application, the middle one (a globe) is to launch that URL in a browser. Clicking on the globe is a quick way to test your web service in a browswer. The copy is useful to use the URL in your Android App. done

e. Click on the globe to confirm that the Hello World servlet is working. done

f. By default, the URL in (d) requires you to be authenticated with Github. To test in a browser, that is fine, but when accessing your web service from your Android app, the Android app will not be authenticated. Therefore you must make the port visibility "Public". To do this, right or control click on the word "Private" in the Visibility column, and change Port Visibility to "Public". You will now be able to access the web service from your Android App or from an unauthenticated browser. done

g. Copy the URL and paste into an Incognito Chrome window to confirm that the Hello World web app can be reached without authentication. checked

h. To deploy your own web service, create a ROOT.war like you did in Lab 3, upload or push the ROOT.war to your repository, and create a Codespace as has just been described. done



Port	Forwarded Address	Running Process	Visibility	Origin
Tomcat Web Server (8080)	https://humble-orbit-55g59w954...	/usr/local/openjdk-16/bin/java -Djava.util.logging.config.file=/usr/local/t...	Public	codespaces+humble-orbit-55g59w954pw3pg9g