CharityScout : Charity Finder and Application

Name : Yukta Bhartia

Andrew ID : ybhartia

Github Repo: https://github.com/CMU-Heinz-95702/distributed-systems-project-04-yukta9-11

Project Overview

CharitySearch is a distributed application that allows users to search for charities using a mobile Android interface. The application communicates with a web service deployed in the cloud, which in turn interacts with the OrgHunter API to fetch charity data. The project also includes a web-based dashboard for displaying analytics and logs.

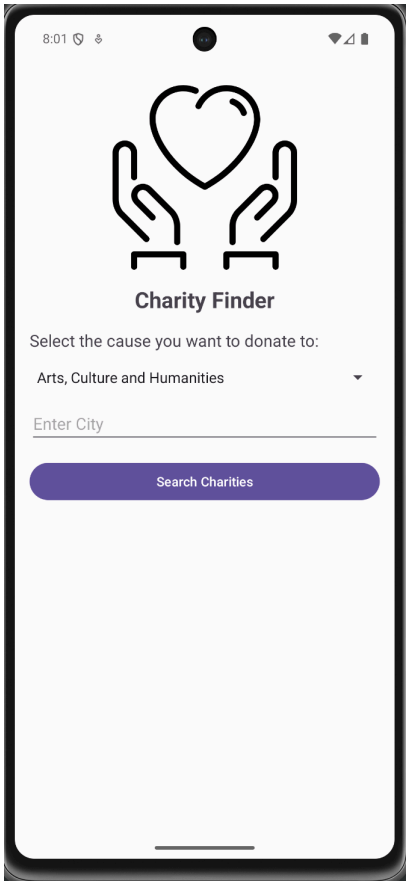
Distributed Application Requirements

1. Android Application Implementation

The Android application for CharitySearch has been implemented with the following features:

- **User Interface**: The app includes multiple View types:
  - EditText: Used for search input where users can enter category and city
  - ListView: Displays search results in a scrollable list
  - TextView: Shows charity details including name, location, and website
  - ImageView: Displays logo or related imagery
  - LinearLayout: Contains the charity information in the list view
- **User Input**: Users can enter category code and city for charity searches.

ANDROID APP SCREENSHOT : Here is a screenshot of the app before the user has entered any input.



- **HTTP Requests**: The app makes HTTP GET requests to the web service using background threads :

```
private void searchCharities(String category, String city) {
        // Execute search in background thread
        new AsyncTask<Void, Void, List<Charity>>() {
        @Override
        protected List<Charity> doInBackground(Void... voids) {
                try {
                String encodedCity = URLEncoder.encode(city,
StandardCharsets.UTF_8.toString());
                URL url = new URL(WEB_SERVICE_URL + "?category=" + category +
"&city=" + encodedCity);
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                 connection.setRequestMethod("GET");
```

```
                // Get response
                InputStream inputStream = connection.getInputStream();
                BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
                StringBuilder response = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                        response.append(line);
                }

               // Parse JSON response
                return parseCharitiesJson(response.toString());
                } catch (Exception e) {
                // Handle error
                return new ArrayList<>();
                }
        }

        @Override
        protected void onPostExecute(List<Charity> charities) {
                // Update UI with results
                if (charities.isEmpty()) {
                showMessage("No charities found. Try different parameters.");
                } else {
                updateCharitiesList(charities);
                }
        }
        }.execute();
}
```

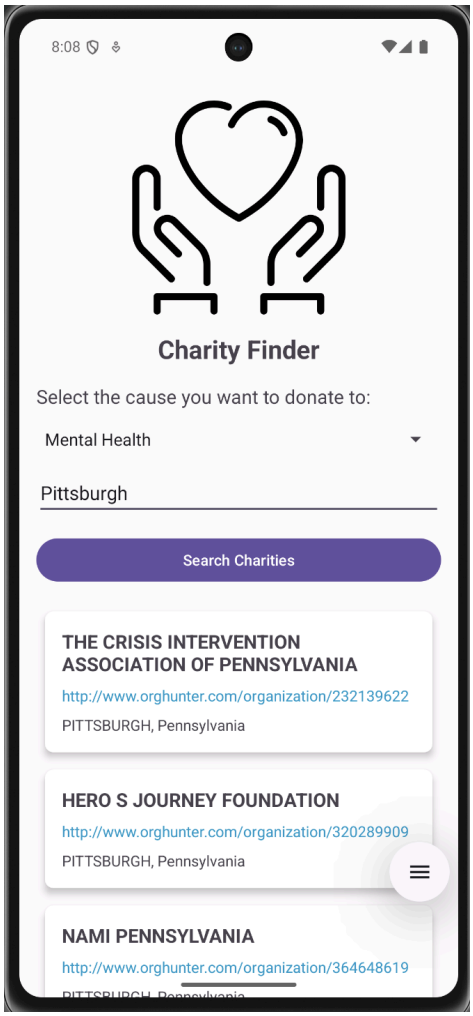
- **JSON Parsing**: Responses from the web service are parsed from JSON format:

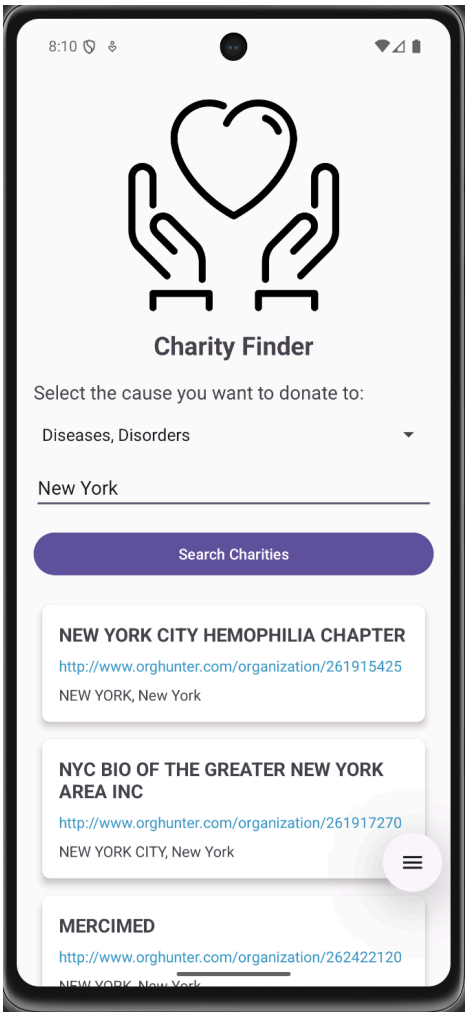
```
private List<Charity> parseCharitiesJson(String jsonResponse) throws
JSONException {
    List<Charity> charities = new ArrayList<>();
    JSONObject jsonObject = new JSONObject(jsonResponse);

    if (jsonObject.getBoolean("success")) {
        JSONArray charitiesArray = jsonObject.getJSONArray("charities");
        for (int i = 0; i < charitiesArray.length(); i++) {
            JSONObject charityObject = charitiesArray.getJSONObject(i);

            String name = charityObject.getString("name");
            String website = charityObject.getString("website");
            String location = charityObject.getString("location");

            Charity charity = new Charity(name, website, location);
            charities.add(charity);
        }
```

```
    }

    return charities;
}
```

- **Display of Results**: Search results are displayed in a ListView, and charity details are shown when a specific charity is selected.
- **Repeatable Usage**: Users can perform multiple searches without restarting the app.

RESULT SCREENSHOT : Here are two screenshots of two different search results being performed based on two different user inputs. The operations are repeatable.





2. Web Service Implementation

The CharitySearchServlet class implements a RESTful web service with the following characteristics:

- **API Implementation**: A simple RESTful API that accepts parameters for category and city to search for relevant charities.
- **Business Logic**: The service processes requests, interacts with the OrgHunter API, and formats responses for the mobile app. It also handles error cases and logs interaction data.
- **Third-party API Integration**: The service fetches data from the OrgHunter API, processing JSON responses:

- **Response Formatting**: Replies to the Android app are formatted in JSON with simplified charity information:


```
{
  "success": true,
  "charities": [
    {
      "name": "Food Bank of Central New York",
      "website": "https://www.foodbankcny.org",
      "location": "Syracuse, NY"
    },
    {
      "name": "American Red Cross Central New York Chapter",
      "website":
"https://www.redcross.org/local/new-york/central-new-york.html",
      "location": "Syracuse, NY"
    }
  ]
}
```

3. Error Handling

The application implements error handling for various `scenarios`:

- Invalid user inputs are validated on both the mobile app and server-side.
- Empty responses are detected and conveyed to the user.
- Third-party API unavailability is handled gracefully.
- Invalid data from the third-party API is managed to prevent app crashes.

```
private void sendErrorResponse(HttpServletResponse response, String message)
throws IOException {
    response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
    JSONObject errorResponse = new JSONObject();
    errorResponse.put("success", false);
    errorResponse.put("message", message);

    try (PrintWriter out = response.getWriter()) {
        out.print(errorResponse.toString());
    }
}
```

Web Service Logging and Analysis Dashboard

4. Logging Implementation

The web service logs the following information for each request:

- Timestamp of the request
- Device model information
- Request parameters (category and city)
- API URL accessed
- Response time for API call
- Total response time
- This data is stored in a MongoDB Atlas database for analysis.

5. Database Integration

The web service successfully connects to, stores, and retrieves information from a MongoDB Atlas database using the official MongoDB Java Driver. The database is configured with three shards for improved performance and scalability:

MongoDB Connection String:
mongodb://ybhartia:distributedsystems@cluster0-shard-00-00.4if0i.mongodb.net:27017,cluster0-shard-00-01.4if0i.mongodb.net:27017,cluster0-shard-00-02.4if0i.mongodb.net:27017/?replicaSet=atlas-c1i6x1-shard-0&ssl=true&authSource=admin&retryWrites=true&w=majority&appName=Cluster0

The MongoDB Atlas cluster is configured with three shards to distribute the data evenly and handle high volumes of log entries efficiently. Each shard contains a portion of the log data, which enables parallel processing of queries and improves the overall performance of the analytics dashboard.

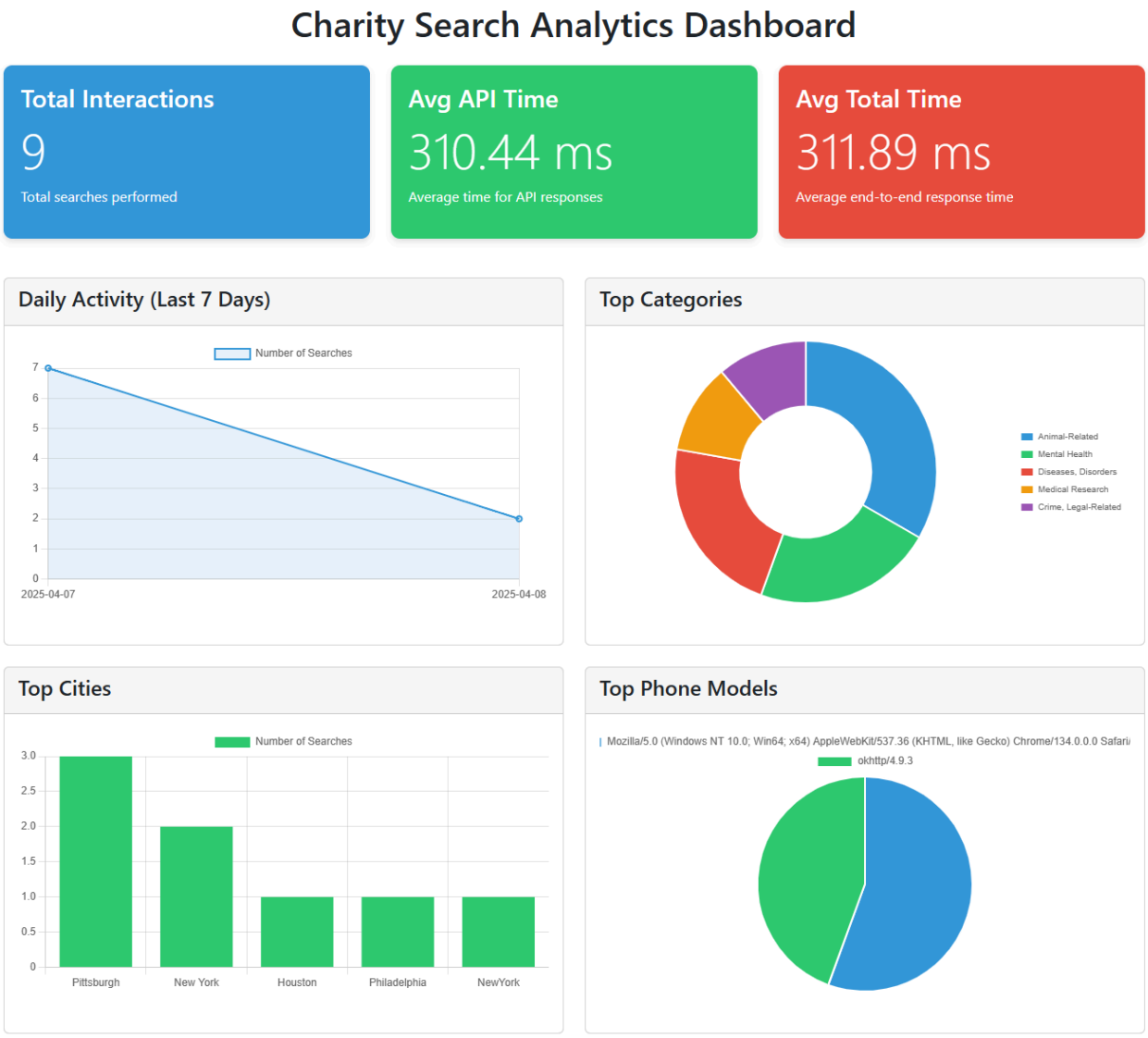
```
private static final String CONNECTION_STRING =
        "mongodb+srv://ybhartia:distributedsystems@cluster0.4if0i.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0";
private static final String DB_NAME = "CharitySearch";
private static final String COLLECTION_NAME = "Logs";

// MongoDB client configuration and connection setup
ServerApi serverApi = ServerApi.builder()
        .version(ServerApiVersion.V1)
```

```
        .build();

MongoClientSettings settings = MongoClientSettings.builder()
        .applyConnectionString(new ConnectionString(CONNECTION_STRING))
        .serverApi(serverApi)
        .build();

mongoClient = MongoClients.create(settings);
database = mongoClient.getDatabase(DB_NAME);
collection = database.getCollection(COLLECTION_NAME);
```

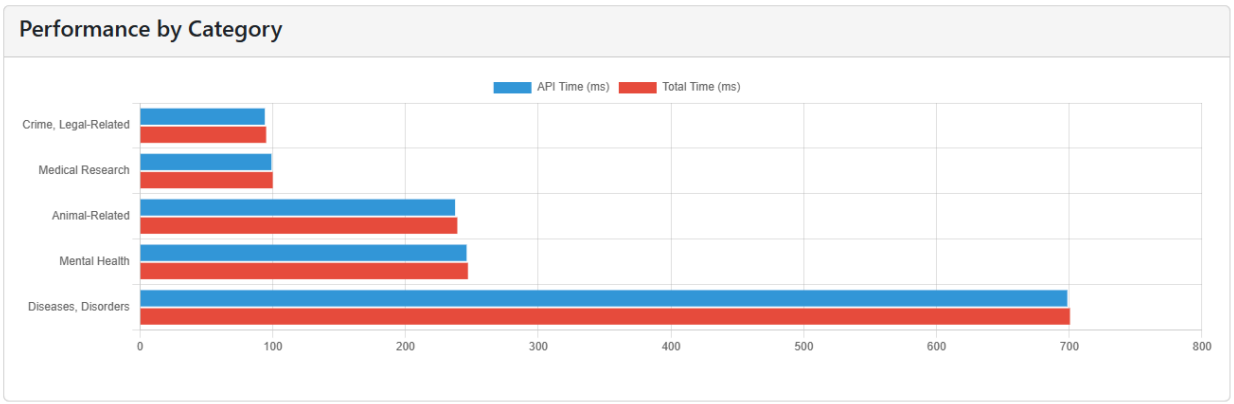
6. Dashboard Implementation

A web-based dashboard has been created with the following features:

- **Unique URL**: The dashboard is accessible via a specific URL.
- **Analytics Display**: The dashboard shows multiple interesting analytics:
    - Total interactions count
    - Average API response time
    - Average total response time
    - Top search categories
    - Top cities searched
    - Top phone models used for searching
    - Daily activity trends
    - Performance by category



- **Full Logs Display**: Logs are displayed in a formatted table showing phone model, category, city, API time, total time, and timestamp.



Recent Searches

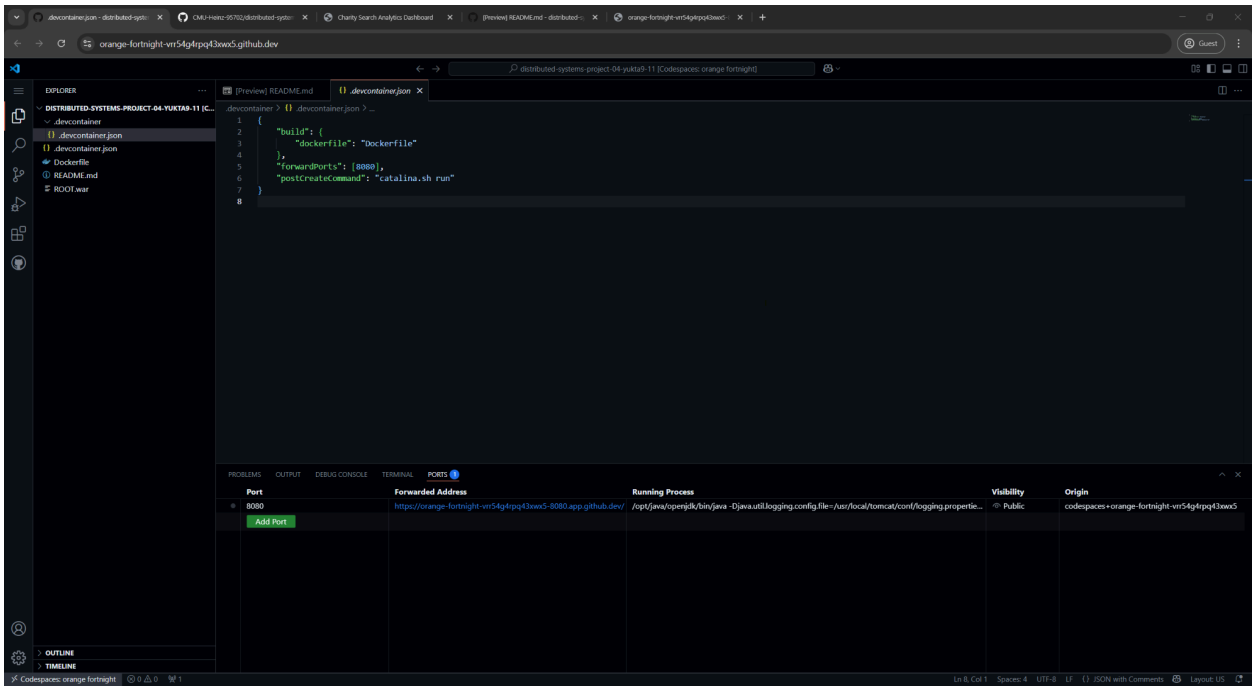
Phone Model	Category	City	API Time (ms)	Total Time (ms)	Timestamp
okhttp/4.9.3	Diseases, Disorders	New York	98	100	2025-04-08 00:09:56
okhttp/4.9.3	Mental Health	Pittsburgh	87	89	2025-04-08 00:08:32
okhttp/4.9.3	Crime, Legal-Related	Pittsburgh	94	95	2025-04-07 23:39:18
okhttp/4.9.3	Medical Research	Cleveland	99	100	2025-04-07 23:27:17
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36	Diseases, Disorders	Houston	1299	1301	2025-04-07 23:24:26
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36	Mental Health	Philadelphia	405	405	2025-04-07 23:08:58
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36	Animal-Related	New York	140	141	2025-04-07 22:44:41
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36	Animal-Related	NewYork	127	128	2025-04-07 22:44:33
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36	Animal-Related	Pittsburgh	445	448	2025-04-07 22:43:57

7. Cloud Deployment

The web service has been successfully deployed to GitHub CodeSpaces , making it accessible from the Android application.

URLs:

- **Root:** https://orange-fortnight-vrr54g4rpq43xwx5-8080.app.github.dev/
- **Dashboard**: https://orange-fortnight-vrr54g4rpq43xwx5-8080.app.github.dev/dashboard.jsp
- **Example API call from Web Service:** https://orange-fortnight-vrr54g4rpq43xwx5-8080.app.github.dev/charities?city=Houston&category=G



Additional Implementation Details

- **Charity Model**: A Charity class was created to encapsulate charity data, including name, website, and location.
- **CharitySearchServlet**: This servlet handles incoming requests, interacts with the OrgHunter API, logs data to MongoDB, and sends responses back to the client.
- **Logger Utility**: A dedicated utility class for logging interactions to MongoDB.
- **Analytics Dashboard**: A JSP page that retrieves and visualizes data from MongoDB.

Conclusion

The CharitySearch project successfully implements a distributed application with a mobile frontend, cloud-based backend, third-party API integration, and an analytics dashboard retrieving data from a MongoDB cluster.