CSCI 578 Final Project

Aditi Gajurel, Felipe Gomez Castano, Rishabh Nevatia,
Tanveesh Singh Chaudhery, Vachirawit Tengchaisri, Yueqin Li

Apr 23, 2024

1 General Introduction

In this project, we designed an android app called PATROL which aims to protect users from pandemic by providing useful information. Users can view crowedness of a location through our real-time heatmap, search for history data of pandemic cases, self-report their symptoms, and checkout newest information from CDC Covid website.

This feasible plan contains both enrichment and trade off decisions we made based on two of our group members' (Yueqin Li and Felipe Castano) prescriptive designs. We chose Yueqin Li's design because we found hers straightforward and feasible, which might be helpful when we work as a team to implement a real running application, given that most of our team members did not have any android development experiences. That said, Yueqin's architecture still has some features such as social platform data collection and analyses work that may not be suitable for fully implementation given the limited resources at hand, and that's where Felipe's design outshines: his "Symptom notification report" functionality is not only helpful in real-life situations, but also enriches user experience without incurring too much workload. Hence, a combination of their designs gives the best of both worlds.

Overall, our new version of PATROL enhances the diversity of information sources and boosts user engagement through the introduction of a notification feature. At the same time, various original designs have been streamlined or substituted with more practical alternatives.

In the following sections, we will be introducing our new design in such a way that the prescriptive design (previous design by Yueqin Li) was listed out first, followed by its implementation status (i.e. not implemented, partially implemented, or fully implemented), and finally if the item was not fully implemented, i.e. either replaced by Felipe's design or abandoned, reasons will be given. As always, for grading conveniences, any deviations will be marked as "*" in the corresponding section's title.

2 Requirements

2.1 Functional Requirements

1 User Registration and Authentication*

- Users should be able to register for an account on the PATROL app.
- The app must authenticate the user when logging in.

Implementation Status:

Not included in our current design.

Reason:

Registration functionality invites too much extra work in both frontend and backend. Since this is only a "nice-to-have" function, given our limited time and resources, we've decided to focus on the main functionalities of PATROL.

2 Access Data*

- The architecture should also support adding and removing social media platforms, and obtain access to the geo-location information available on those. This might involve using the platform's API or any other approved method for accessing data from it.

Implementation Status:

Replaced social platform access with other information sources.

Reason:

We replaced social media platform with our own real time data collection services. The reason behind this is two folds. For one thing, after investigation, we found that social media platforms' APIs are quite different from one to another, and many of them may not directly provide geo-tag information. For instance, in several scenarios, the only way that we can collect real-time geo-location data is through first gathering current posts, then use the user-id parsed from the post data to call the other API in order to acquire the user's geo-location information. Not only is the process complicated to implement, but through the back and forth of data analysis and API calling, latency would become a big issue. For another, it is also tricky about how to handle the huge scale of data we're about to face if combining several social platforms together. Hence, for simplicity and feasibility based on current resources, we've decided to replace this data source with our own real time data collection services.

The way we achieved this is by adding a check-in functionality which will automatically report user's real time location to our backend once user opens the app. While acquiring user's location data, we also made sure to protect user's privacy by collecting nothing other than two figures – latitude and longitude. The backend API for this functionality is "/checkIn".

3 Filter Relevant Data

- Filter out the geo-location data of concern. This could include check-ins, location tags in posts, location data from photos, etc.

Implementation Status:

Implemented. Check-in functionality takes care of collection of geo-location data.

4 Aggregate Data*

- Aggregate the geo-location data to derive meaningful insights. This could involve counting the number of check-ins or posts from different locations, retrieving similar data in the past few days from databases or caches, analyzing trends over time, etc.

Implementation Status:

Partially implemented. One of our fucntionalities – heatmap – replies on both the aggregated geo-location data and the dynamic map supported by google API. To specify, Check-in happens when user opens PATROL, the live data will be sent to the backend and later the aggregated data will be utilized by heat map fragment to indicate the area's crowdedness.

Reason:

While we do have history statistics functionality, the data directly came from official databases instead of our own aggregation algorithms. The reason being that our live data only come from user check-in records, and since the history section is supposed to provide statistics within a relatively arbitrary time range, and that don't want to fake data that we don't really have, we hence switched to official pandemic databases for history case statistics. At the same time, we did not implement future trends for the reason that we are currently not witnessing a pandemic and it would not be appropriate to predict any data without optimal metrics.

5 Visualize Data

- Create visualizations such as maps, charts, or graphs to represent the summarized geo-location information effectively.

Implementation Status:

Implemented. We used heat map to demonstrate current crowdedness in a certain area, and bar chart to show history data collected from official Covid-19 database.

6 Analyze Insights*

- Analyze the summarized geo-location information to gain insights. This could include calculating crowd density in specific locations, summarizing history data, detecting trends area population density, etc.

Implementation Status:

Partially implemented. We included calculating crowd density in specific locations and summarizing Covid history data, abandoned trend detection sub-function.

Reason:

As has been mentioned before, geo-location data was collected via user check-in which happens automatically. The first time that the user opens the application. The application will collect the device ID and register it to the database to prevent repeat data in the data store. After that every time the user opens the application the user's location will submit to the database automatically.

When the user searches at the search bar, the application will use all of the check-in location data in the database to depict the intensity of data on the map that is displayed to the user. We also modified the summarizing history data to the real Covid history data to scope our work in the Covid-specific area.

7 Secure Data

- Take appropriate measures to secure the data and protect the privacy of users whose information is being analyzed. This includes anonymizing data whenever possible and storing it securely.

Implementation Status:

This functional requirement was taken care of during our implementations. The details were mentioned in the "Access Data" section.

8 User Interaction*

- When GUI users input a location, a radius and hit submit, the system will return a heat map showing current crowdedness of that area.
- When GUI users select "history" tab, the app will show a chart representing the population record of the area in the past 7 days. Users will be able to checkout each day's hourly population line chart by clicking on the corresponding bar or label.
- When GUI user select the "predict" tab, the app will show a line chart of the predicted crowdedness of that area in the next 24 hours. The x-axis will be the hours ranging from the next hour H to H + 24, y-axis will be the population prediction of the corresponding hour.
- Developers will use the app's API to interact with the app. Depending on user request, the system can return current/historical geo-location aggregation or data/analyzation data, or future trends analysis results, which was the same set of data being used by GUI users, albeit organized and sent in JSON form back to the API caller.

Implementation Status:

Partially implemented with additional features. "Heatmap" was fully implemented as previously designed. "history" was simplified to three buttons: start date choose button, end date choose button, and submit button. The response is a bar chart indicating each day's total case count, no hourly line chart will show up upon clicking each bar. Lastly, as have been discussed before, we don't support "predict" in our current version. On the developers' side, our Patrol application includes the crowdedness heat map and provides API for developer users based on prescriptive design. Furthermore, we have added more features such as Covid case history, and symptom report notification tabs.

Reason:

"History" tab was simplified due to the statistic source we got only contains case count per day, hence we have to limit our design to a certain granularity. The deletion of "prediction" tab roots from our decision to abandon the prediction functionality. On top of these changes, we added a symptom report notification based on Felipe's original design, since we found this functionality is useful, enriches user interactions of our app, and is easy to implement.

2.2 Non-Functional Requirements

1 Security*

- Ensure data encryption and secure transmission of sensitive information.
- Implement robust user authentication mechanisms to prevent unauthorized access.
- Comply with data protection regulations such as GDPR and CCPA.

Implementation Status:

Partially implemented.

Reason:

While we do have mechanisms to protect users' privacy, we didn't implement anything related to data encryption, secure transmission or user authentication. For one thing, due to the measures taken to protect user privacy (which was mentioned in previous sections) and the minimum information we required from the user end, users won't face too much risk in using our app. For another, data encryption, secure transmission and authentication are complicated topics to handle and may not be suitable for implementation at current stage due to the limitation of time and resources.

2 Performance

- The app must handle a large volume of data efficiently without significant latency.
- Ensure scalability to accommodate a growing user base and increasing data sources.

Implementation Status:

Implemented. The application is very salable as it is developed using a auto-scalable and robust backend supported by MongoDB Atlas. It can serve a lot of users and have a high transactions per second.

3 Reliability

- PATROL should be highly available and resilient to ensure users can access critical pandemic information at all times.
- Implement backup and disaster recovery mechanisms to prevent data loss.

Implementation Status:

Implemented. The backend is very reliable. It has fail-over mechanisms in case of errors. So, in case of any crashes on the backend, a new server is spun up and served, keeping reliability.

4 Usability*

- Design an intuitive user interface that is easy to navigate and understand, especially for users in stressful situations.
- Provide multi-lingual support to cater to diverse user demographics.

Implementation Status:

Partially implemented. Our UI design was very simple and user-friendly with intuitive icons, clear text information and simple UI components such as buttons and text input boxes.

Reason:

We did not implement the second item because multi-language support is not considered to be a major functionality of PATROL. Due to limited resources and time constraints, we've decided to focus on more important features.

3 Android Architectural Model[1][2][3][4][5]

1 Use Case Diagram*

Target users can be divided into two groups: android app users and developers. The former can access information provided by our app via android UI, and latter will do so via calling our backend APIs. Both groups will be able to use current crowdedness heat map, search for history case count, and self-report functionalities. And our android app users will be entitled to an extra feature, which is to view CDC Covid information through clicking on the floating Covid button in the map screen.

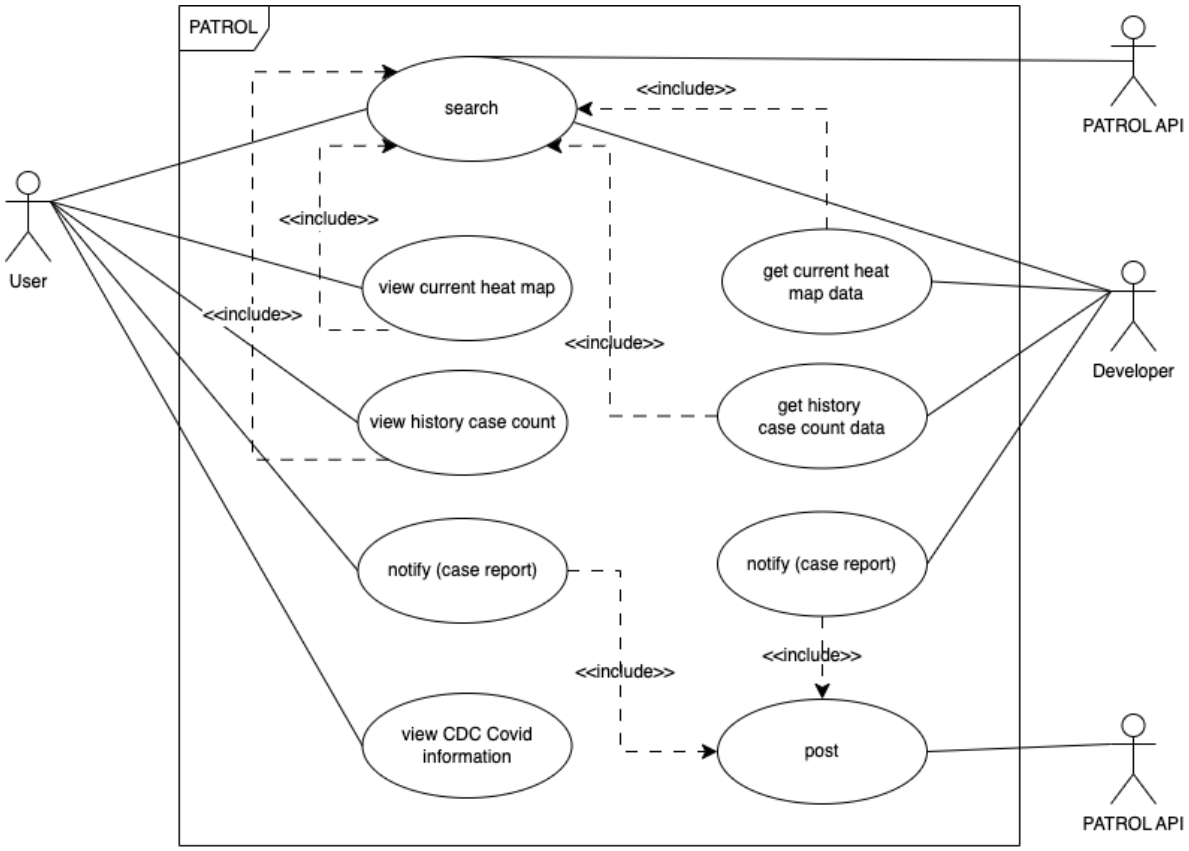

Figure 1: Use Case Diagram

2 Class Diagram*

In the Class Diagram, the classes are demonstrated along with their relationship within them to depict the structure of the PATROL application. Sticking to the MVVM architecture, we have three categories of the classes i.e., Model which is essentially the structure of the data, View Model which is about binding the data from Model to the View and View is the User Interface. The View classes are connected to View Model classes and all View Model classes are connected to the Patrol Repository.

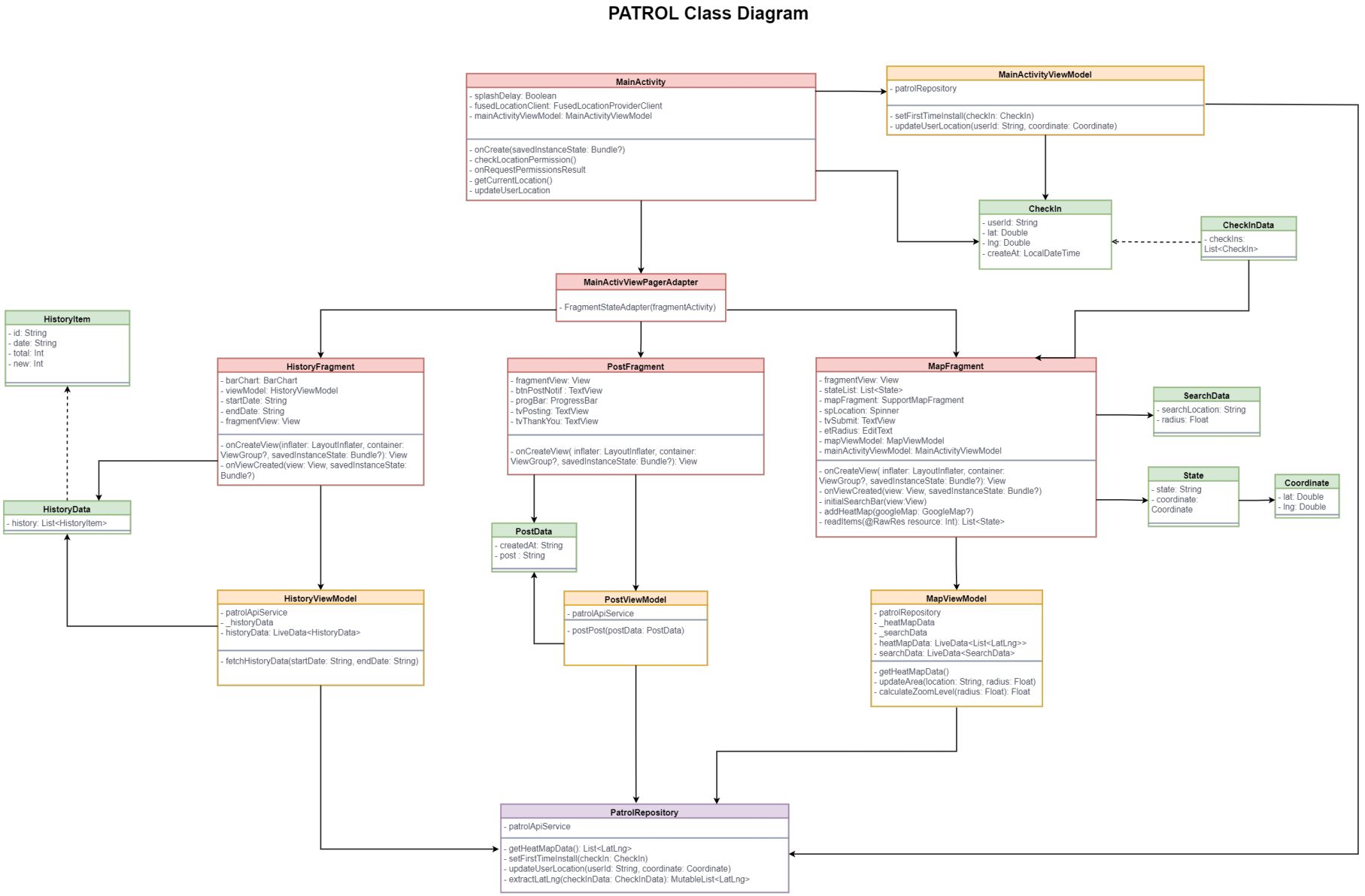

Figure 2: Class Diagram

3 Activity Diagram*

The Activity diagram (Fig. 3) depicts the project's flow and the application's behavior when the user opens the application. When the user opens the application, it checks the device ID and registers relevant information in the database. The user can input the location and radius to view the heat map for a particular region. The user can traverse the US government website to get the latest official information. The user can also view the past data by generating the chart for that particular timeframe and also add in any reports using the reporting feature.

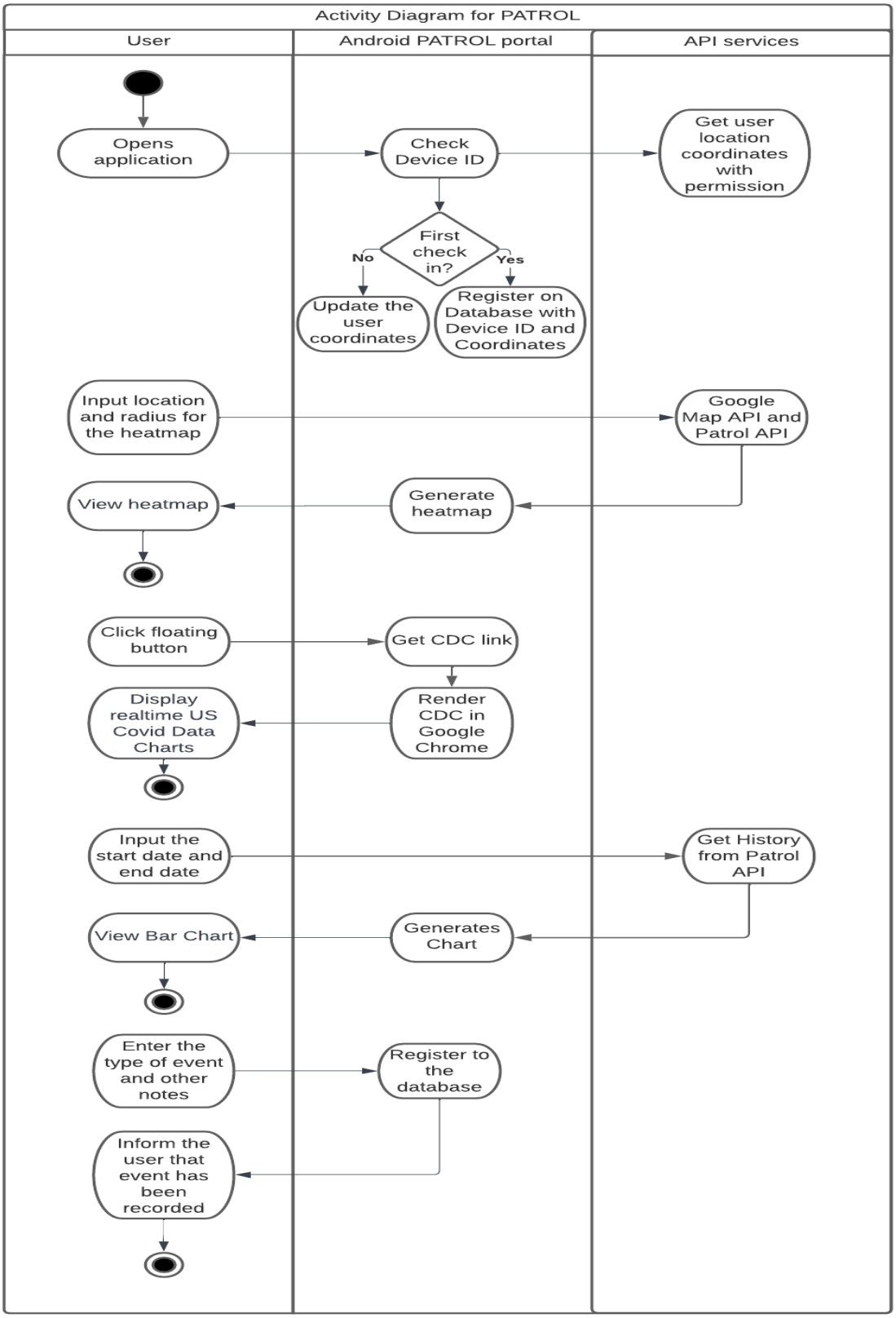

Figure 3: Activity Diagram

4 Deployment Diagram*

The deployment diagram shows the physical component of the system design. The application can provide access to both user and developer users through an Android Patrol application using APK for the installation and Patrol API call. For the backend, the backend code is installed on the container styles cloud server which improves the scalability and availability. Moreover, this system selects MongoDB as the database to maximize the query performance and the ability to handle large data.

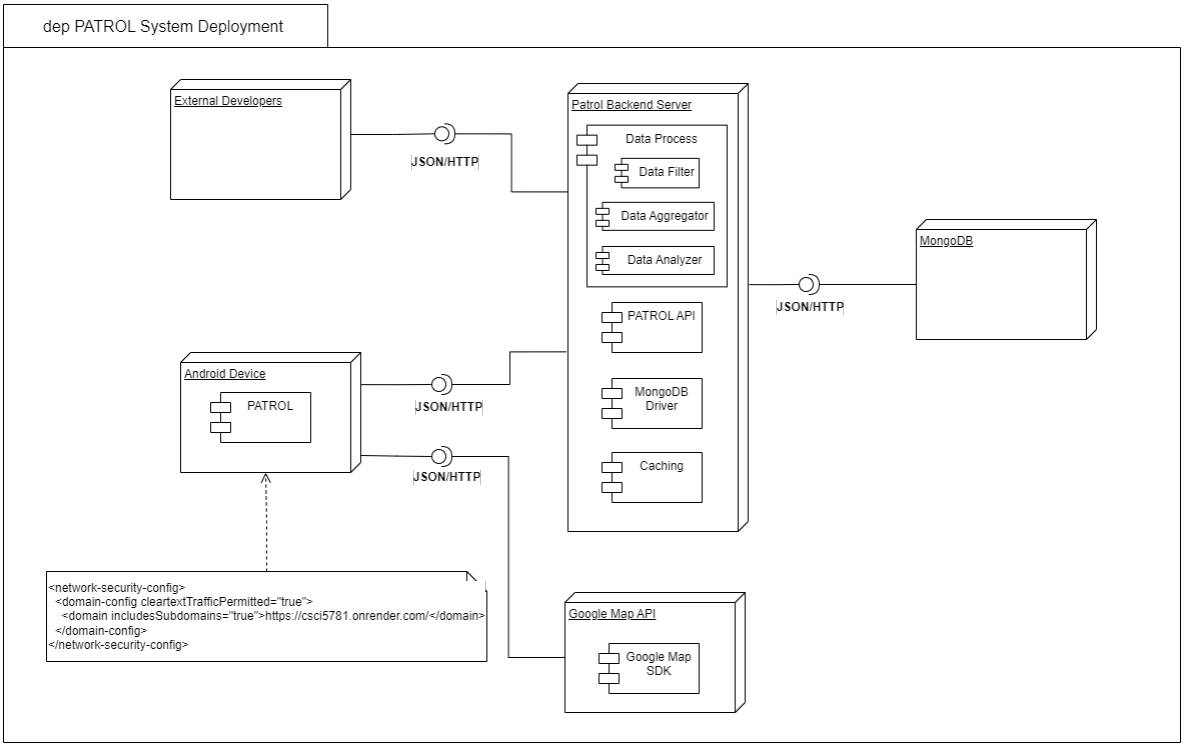

Figure 4: Deployment Diagram

References

[1] developer.android.com. *Recommendations for Android architecture*. URL: https://developer.android.com/topic/architecture/recommendations.

[2] Kirill Fakhroutdinov. *Deployment Diagrams Overview*. URL: https://www.uml-diagrams.org/deployment-diagrams-overview.html.

[3] *Google Maps Android Heatmap Utility*. URL: https://developers.google.com/maps/documentation/android-sdk/utility/heatmap.

[4] *Maps Android KTX*. URL: https://github.com/googlemaps/android-maps-ktx.

[5] *Retrofit: A type-safe HTTP client for Android and Java*. URL: https://square.github.io/retrofit/.