# ID2012 Ubiquitous Computing Project Report

Sihan Chen, Yuehao Sui, Zidi Chen

May 2021

## 1 Introduction

Our project **RunSpec**, which has a strong relationship with ubiquitous computing, is a client-server application to help people with running. In this section, we will introduce the idea behind **RunSpec** and the relationship with ubiquitous computing.

Github repo: https://github.com/Spycsh/RunSpec

### 1.1 Idea and Purpose

Even before we started learning this course, we three had an idea to develop an application to help people to run or jog. We thought we could do something interesting and different with the collected data during people jogging. This course offers us an opportunity and knowledge to continue with this idea.

Our basic idea is that the application use android front end as sensor. We get location and send the data to the back end. The back end application handles these data and gives proper response when needed. During users' running, we could keep passing sensor data to the back end every 5 seconds, so that we can track the running.

To clarify our ideas, the basic functionalities are as follows:

- Before running, it should provide users with current weather and air quality conditions. Users can decide based on these conditions whether to run or not.
- We have a few predefined spots (Point of Interest, aka POI) in the application. When the user run nearby the spots during running, we count. In the back end, we could see how many times all the users pass the spots, which could be considered as the popularity of the spots among all users.
- Users could get top five hot spots in the area before running, so that they can plan their route.
- When one user is running, we plan to display run steps, and kilometers (the basic information from android sensor). In addition, we also display which spots the user have passed by after the user finish running.

To summarize, we want to provide an client-server application which could give necessary advise information before running, and we could monitor spots in the back end. In addition, the user could also get top five hot spots before running, and get whether they pass the hot spots after running, which should give a feel like a checklist functionality.

## 1.2 Relationship with Ubiquitous Computing

Our project can be considered as an Ubiquitous Computing application. From the course content, we know that in ubiquitous systems, normally we need sensors for interaction between human and machine. In our project, we use android sensors to detect latitude and longitude, as well as steps and distance. We also call public open weather map API for current weather, weather within future two hours, and current air quality.

Furthermore, there is an important concept in Ubiquitous Computing, which is context-awareness. We applied context-awareness in the project when we handling the count of spots. Figure 1 shows the way to count how many times the users pass spot A **without considering context-awareness**. We use the latitude and longitude to check if the users are in the radius of spot A. Since the front end will send the location every 5 seconds, the count of spot A will keep increasing if the users stay around spot A. This is actually not desired, because we do not want to count when the runner just stop at the spot A for a few minutes or run very slowly around spot A.

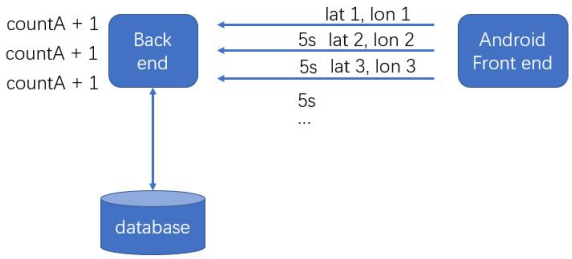


Figure 1: Spot count without context-awareness

What we have done, **considering context awareness**, as shown in figure 2, is that we use two additional parameters: user id and trip id, together with the POI id to count once that one user is near one point of interest in one run trip. Each record of user location with user id and trip id is stored in database so when the back end receives a new record, it checks if the record with the same user id and the same trip id and the same POI id exists in the database. Under some circumstances, one user in one trip may just run inside the radius of

a POI for a long time and produce multiple such records into the database but we only need to count one record of them. The advantage is that by filtering out redundant data we are able to accurately measure the overall popularity of POI.

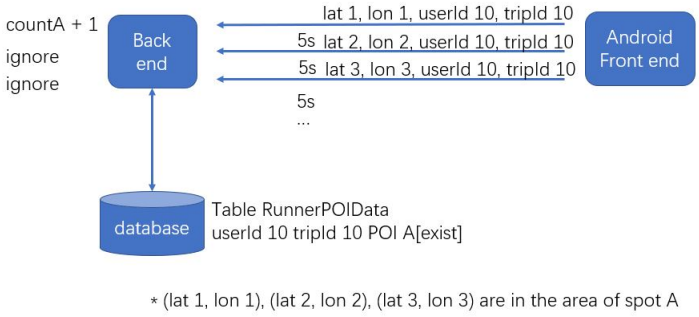


Figure 2: Spot count with context-awareness

Overall, our project is a ubiquitous application with necessary sensors from android, public weather and air API, and our project does consider context-awareness.

# 2 Implementation

We have one android front end, one back end connecting with database, and one statistics board to monitor POI.

## 2.1 Android Front End

The Android App for this project could be run on any Android 8.1 or higher version. The development of the Android project follows the latest Google standards and MVVM (Model–view–viewmodel) software architectural pattern.

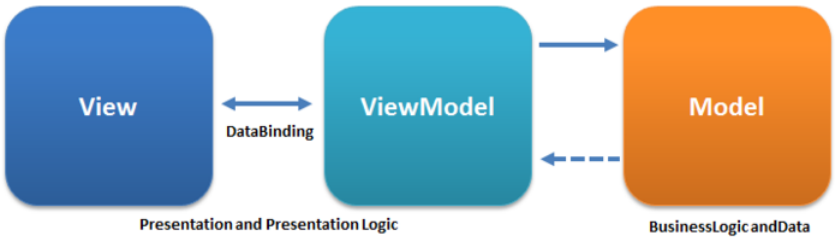


Figure 3: MVVN software architectural pattern

### 2.1.1 View

This App has only a single Activity, which is made up of a Fragment container with a bottom control bar. The content displayed in the container is switched by touching the different tabs on the control bar. The app has three different Fragments, which are:

- Home: Shows current location, weather, air quality and most popular locations.
- Dashboard: Controls to start or stop running, shows the number of steps, the distance and the places already travelled.
- Settings: Change your name and server address.

### 2.1.2 ViewModel

Dynamic binding of views and data is provided through the official Android library.

Create multiple LiveData data containers in the ViewModel and encapsulate important data in the LiveData data container. Once the containers have observed the changes of the internal data they can perform the specified logic, for example updating the corresponding components in the View.

For example, if GPS information is available, when the Model gets the latest location via GPS, it will update the variable data within the ViewModel, and the data on the view presented to the user will also be updated at the same time. Thus the user will always know their exact location.

### 2.1.3 Model

The app uses Google's Location Services SDK to get the user's location. The advantage of Google's fusion location service is that it can provide rough location based on network information and Bluetooth even when GPS signals are weak.

App and server exchange data via RESTful Http requests. Here I used the open source Volley library recommended by Google. Volley offers the following advantages:

- Automatic network request scheduling.
- Multiple concurrent network connections.
- +Transparent disk and in-memory response caching with standard HTTP cache coherency.

## 2.2 Back End

In the back end, we have four modules for different usage: adviser, producer, processor and statistics board. Figure 4 shows the project structure with technologies.

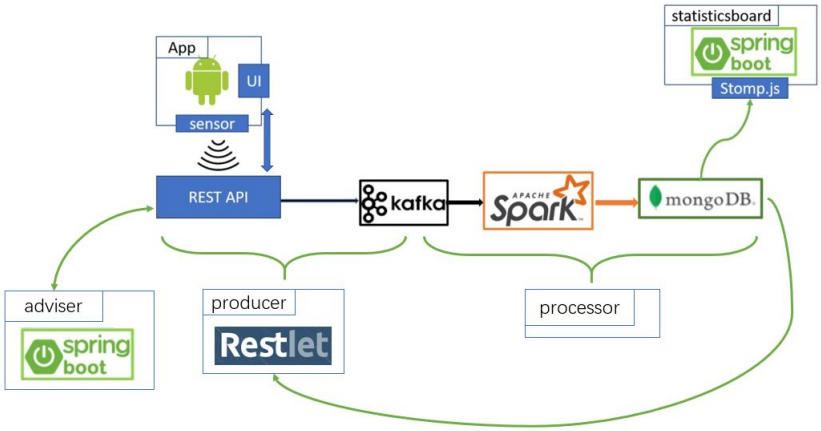


Figure 4: Project structure

### 2.2.1 adviser

Adviser module is responsible for calling open APIs to get weather and air quality information by latitude and longitude. We used "one call" and "air pollution" APIs from open weather map. We decided to just present these information to users, so that they can decide to run or not, based on themselves' preference.

The other three modules (producer, processor and statics board) are used for handling the user data during the run and after the run.

### 2.2.2 producer

Producer contains three REST APIs for front end and also provides the functionality to serialize and send runner data to Kafka, a popular distributed message queue to handle voluminous data.

- /producer/runningData: for android to pass runner location during trip. (should be called every 5 seconds during the run)
- /producer/returnTripData: return the spots where the current user passed in the current running trip. (should be called after the trip)
- /producer/hotSpot: return top five hot spots. (should be called before the run)

### 2.2.3 processor

Processor contains the functionality of Kafka consumer, which will consume the runner data from the message queue in order and relieve the pressure of read/write operations of MongoDB. Here we also use Spark streaming to handle the streaming runner data. Spark offers plenty of functional operations of streams such as map, filter etc. Data will be ensured to be processed fluently and saved as new tables in MongoDB.

### 2.2.4 statics board

We have a statistics board to monitor all the POI statistics. As shown in Figure 5, on the left side we show the map of the POI with the numbers of cases that the runners have passed by, which can directly reflect the hot extent of the POI. On the right side we show the relevant detailed records.

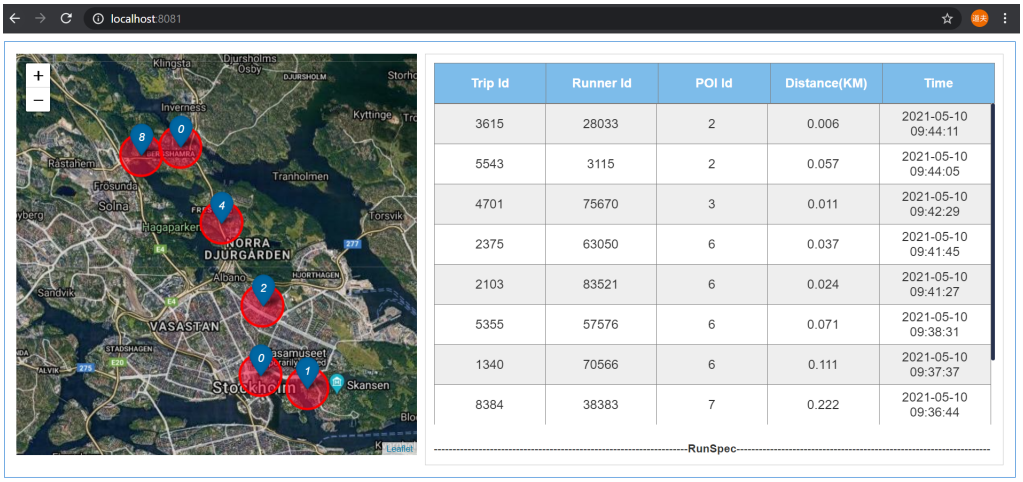


Figure 5: statistics board

## 2.3 Database

Our database is MongoDB. Figure 6 shows the database structure. We have three tables. *POI* means points of interest, which stores all predefined spots.

*RunnerData* is to store the message of user location during the run from front end. *RunnerPOIData* stores the which spots the user passes in the run.

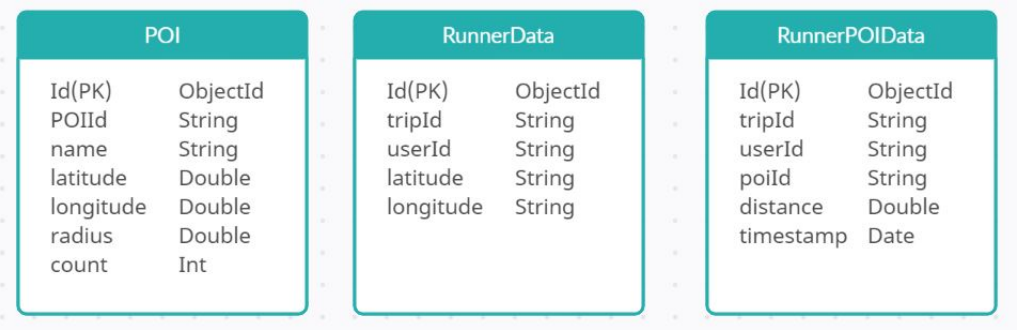


Figure 6: Database schema

# 3 Discussion and Evaluation

Here we evaluate our project and list what we can do in the future.

## 3.1 evaluation

Overall, we realized the highlight functionalities in our plan. We use android sensor to get runner data, use open API to get weather and air information, and we handle data with the consideration of context-awareness. This is a ubiquitous project to help users to have a better and different experience of running.

## 3.2 Future work

Our project could be extended in many aspects. The basic extension could be user account management. In addition, we could enable users to add more spots (POI). We could also show the history about which spots the user have passed in all trips. Furthermore, we have tested that our project could tolerant around 200 calls on 1 API a time. If we want this application to be used in practice, maybe we need to make our application tolerant high concurrency.

# 4 Team Work

- Sihan Chen: processor module, producer module, statistics board module
- Yuehao Sui: Android front end
- Zidi Chen: adviser module, APIs in producer module, POI table generation in database