WeatherCompare

Software Requirements Specification

1.0.1

07/12/2017

Giona Fossati

Lead Software Engineer

Revision History

Date	Description	Author	Comments
20.10.17	First specifications, General Description, Specific Requirements and Appendices.	G	0.0.1
22.10	Added section 3.1-3.2	G	0.0.2
30.10	Corrected Use Case Diagram.	G	0.0.3
05.11	Added Use Case Scenarios 3.2	G	0.0.4
17.11	Added Class Diagram and GitHub repository link.	G	0.0.5
21.11	Added Non-Functional Requirements	G	0.1.0
24.11	Added GUI mockups	G	0.1.1
26.11 - 01.12	Completed the first official version of SRS	G	1.0.0
07.12	Added methods and attributes of Database class	G	1.0.1

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	Giona Fossati	Lead Software Eng.	

Table of Contents

1. Introduction

1.1 Purpose

The main purpose of this SRS Document is to inform and make a solid statement about what WeatherCompare will do, why and what are the user needs its develop is focused on. Also to clarify how
and why certain decision has been made and to inform everyone interested about WeatherCompare
Android app first develops.

1.2 Scope

The Scope of the WeatherCompare is to provide a different way of consulting the weather. Not anymore one-channel based but, thanks to the possibility of gather information from different sources through API requests, will display the forecasts of the interested location provided by different weather clients.
The main benefit from that will be to let people know the reliability of every weather services and let them rate and share their experience. Through this system the application will determine the “weight” of the forecast of every channel, based on the reliability that the users expressed through a 5-star rating system, and will provide to make a “Weighted Average” of the various forecasts. This will let the user know what will be the more reliable forecasts and avoid surprises during their every-day routine.

1.3 Glossary

Term	Definition
API	Set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components.
Weather Provider	Commercial weather service providing real-time weather information via the Internet.
Weighted Average	A method of computing a kind of arithmetic mean of a set of numbers in which some elements of the set carry more importance (weight) than others.

1.4 References

https://goo.gl/jrbwd6 - Informatica ed Elementi di Programmazione II - Sessione 4 - SRS

https://github.com/GionaFossati/WeatherCompare-Android-APP - Project's GitHub Repository

https://goo.gl/s42q4j - "Reading very big JSON files in stream mode with GSON"

https://goo.gl/A9DcjQ - "StringBuffer versus String"

https://www.apixu.com/doc/ - Apixu API Documentation

https://www.wunderground.com/weather/api/ - Weather Underground API Informations

https://openweathermap.org/api - Open Weather Map API Documentation

2. General Description

2.1 Product Perspective

The need behind the development of WeatherCompare is an hitch of an every-day situation experienced by me: checking the weather in the first morning, so I can plan and decide what kind of clothes I'll have to wear during the day and realize, very often too late, that the weather forecasts were wrong. This means that I found myself a lot of times dressed "wrong" for daily weather.
I would like to take care of the need of knowing the reliability of the weather forecast by introducing an App that permits people to have an almost complete trust of their daily forecast.

2.2 Product Functions

- Search for a city weather
- Retrieve and display weather informations about the city searched
- Send feedback about specific weather service
- See feedbacks and ratings about every weather service
- Use feedbacks to make a most accurate prevision of the temperatures cross-referencing the feedbacks and the temperatures.

2.3 User Characteristics

The expected App user doesn't have any particular characteristic. I'll expect the final user to be the average user that doesn't have too much technical knowledge about the smartphone and his use is limited around the basic Applications. The final user doesn't need to have specific knowledge about smartphones to be able to use WeatherCompare.

2.4 General Constraints

- API requests needs a key provided only by the weather service.
- The feedback system based on a relational DB that has to be encrypted to assure the truthfully of the feebacks.
- No particular hardware limitations due to the fact that computation is mostly server-side
- The phone needs Internet connection

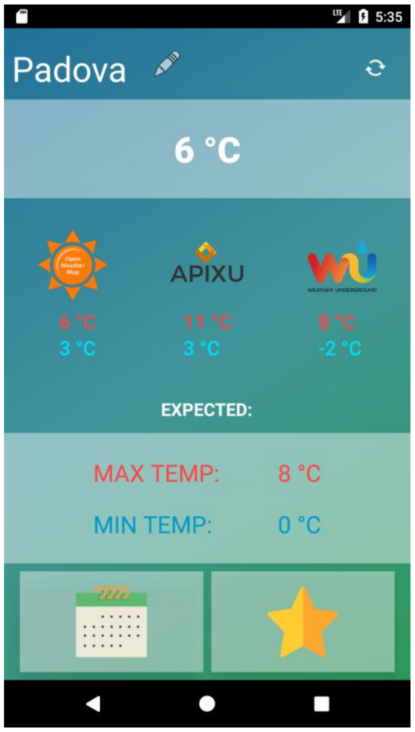
2.5 Assumptions and Dependencies

- Weather Clients' server needs to be up in order the app to work properly
- Feedback's DB needs to run in order to the feedback system to work

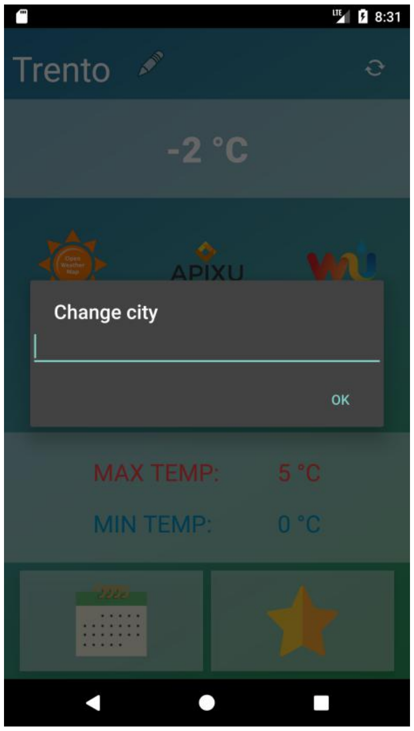
3. Specific Requirements

3.1 User Interfaces

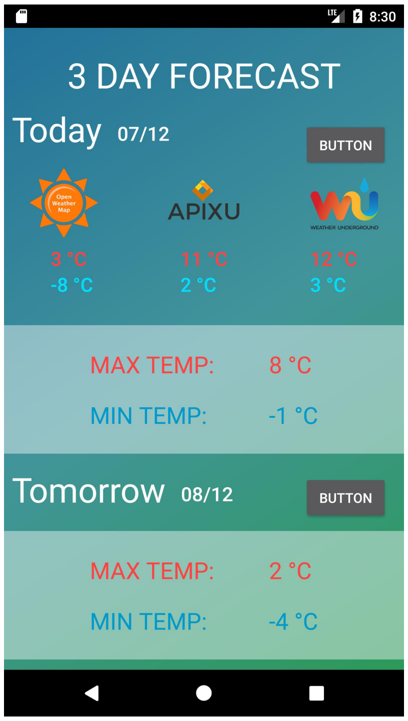
Below are illustrated some mockups of the finished application, colours and style will be defined in the last versions.



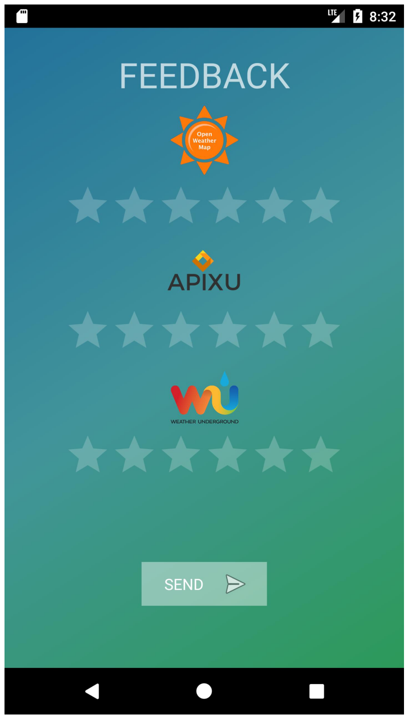
Main



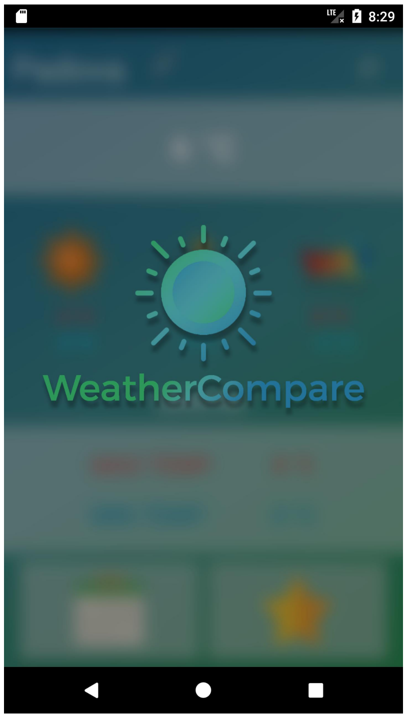
“Change city”Popup



3-day forecast



Send feedbacks



Initial Splash Screen

3.2 Functional Requirements

This section describes specific features of the software project.

3.2.1 Cross-reference datas

Working on the datas retrieved from the API requests the app will cross-reference these with the feedbacks ratings to adjust the expected temperatures towards the temperatures of most accurate weather service.

3.2.2 Save and Change Favourite City

The users doesn't have type his own city every time he enters the application, the last city searched needs to be kept in memory trough SavedPreferences to provide a sufficient good experience of use.

3.2.3 Send Feedbacks about the Services

There will be a dedicated page where the user will be able to easily change the values of the feedback about every service and send them pressing a button

3.2.4 Indexed Italy's Major Cities

The app must be able to provide the weather of at least every major city (50'000+) in Italy to permit possibily the most part of the users to feel the application useful.

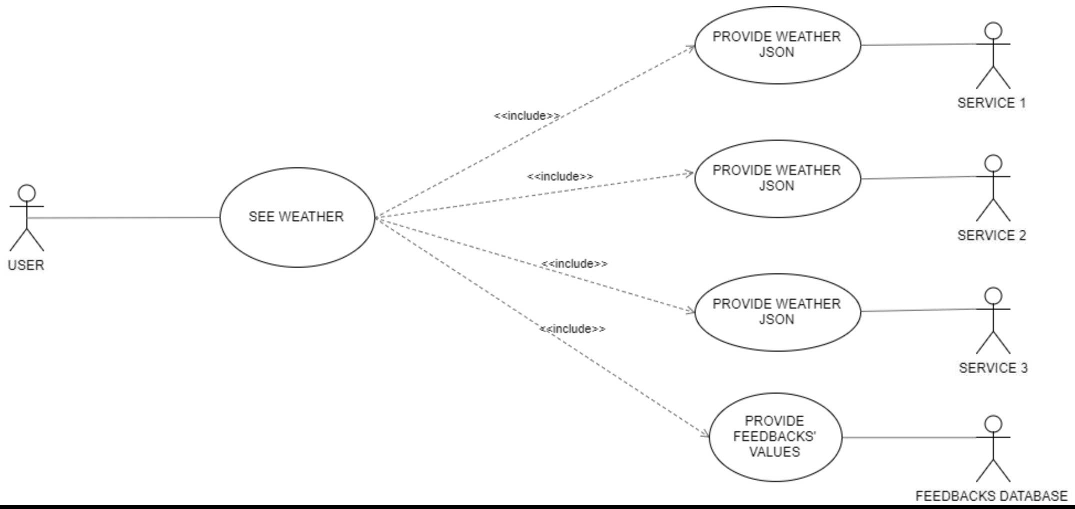
3.2.5 Display Temperatures

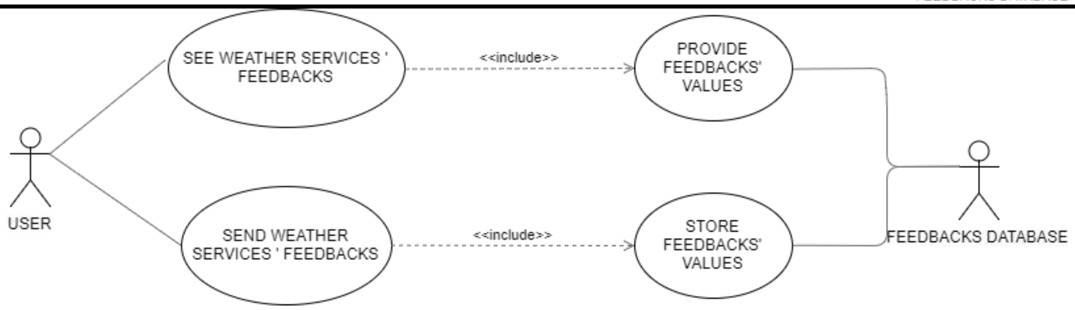
The temperatures are a minimum requirement for the functionality of the app, they need to be displayed every time the user starts the app.

3.2.6 Refresh Weather

Every time a new city is typed the weather must updated the weather according to the new preference.

3.2.7 Use Cases




SEE WEATHER SERVICES ' FEEDBACKS
<<include>>
PROVIDE FEEDBACKS' VALUES
USER
FEEDBACKS DATABASE
SEND WEATHER SERVICES ' FEEDBACKS
<<include>>
STORE FEEDBACKS' VALUES

3.2.8 Use Case Scenarios

Use Case Name	See Weather
Trigger	The user has interest to see the weather
Input	Open app
Basic Path	● Open App ● The app makes API requests to the different providers for the weather JSONs files about the city ● The weather is displayed
Output	The main screen appears with the weather datas

Use Case Name	Send weather services feedbacks
Trigger	The user wants the application to do more accurate previsions
Input	3 different values on a 10-point scale
Basic Path	● Tap on Provider’s Feedbacks button ● Scroll the slider bars to change the values ● Tap “Send”

	● The feedback DB stores the feedback ● A message saying that the valutation have been stored is displayed
Output	Dialog window “Sent sucessfully”

Use Case Name	See weather Services Feedbacks
Trigger	Curiosity about the reliability of each weather service
Input	NA
Basic Path	● Tap on Provider’s Logos ● The Feedback’s DB provides the values ● The values are displayed in a new Dialog Window
Output	A dialog window pops-up with the values

Use Case Name	Change city weather displayed
Trigger	The user has interest to see another city weather
Input	City
Basic Path	● Tap on edit icon ● Type city ● Hit enter ● The app makes API requests to the different providers for the weather JSONs files about the city ● The weather is displayed
Output	The main screen updates with the new weather datas

3.3 Classes

Reference to functional requirements. You may use informal methods (such as CRC) to define preliminary classes

3.3.1 <MainActivity>

3.3.1.1 Attributes

- public String: City
- public String[]: owm[]
- public String[]: apixu[]
- public String[]: wu[]
- public Double[]: feedValues[]

3.3.1.2 Methods

- OnCreate() extends AppCompatActivity
- fetchOwm(String city)
- fetchApixu(String city)
- fetchWu(String city)
- fetchFeedValues()
- SetViewText(String[] owm, String[] apixu, String[] wu,Double[] feedValues)
- buttonsClick()
- showInputDialog()
- changeCity(String city)

3.3.2 <RemoteFetchWu>

3.3.2.1 Attributes

- public String: City
- public String[]: wu[]
- private JSONObject data
- private JSONObject forecastData

3.3.2.2 Methods

- getCurrent(String city)
- getForecast(String city)
- fetchData(String city)
- extractData(JSONObject data)
- fetchForecast(String city)
- extractForecast(JSONObject forecastData)

3.3.3<RemoteFetchApixu>

3.3.3.1 Attributes

- public String: City
- public String[]: apixu[]
- private JSONObject data
- private JSONObject forecastData

3.3.3.2 Methods

- getCurrent(String city)

- getForecast(String city)
- fetchData(String city)
- extractData(JSONObject data)
- fetchForecast(String city)
- extractForecast(JSONObject forecastData)

3.3.4<RemoteFetchOwm>
3.3.4.1 Attributes
- public String: City
- public String[]: owm[]
- private JSONObject data
- private JSONObject forecastData
3.3.4.2 Methods
- getCurrent(String city)
- getForecast(String city)
- fetchData(String city)
- extractData(JSONObject data)
- fetchForecast(String city)
- extractForecast(JSONObject forecastData)

3.3.5<CityPreference>
3.3.5.1 Attributes
- public String: city
- public SharedPreferences: prefs
3.3.5.2 Methods
- Citypreference(Activity activity)
- getCity()
- setCity(String city)

3.3.6 <DatabaseHelper>
3.3.6.1 Attributes
- public Integer[]: newFeedValues
- public Double[]: getFeedValues
- public static final String DATABASE_NAME = "feedback";
- public static final String APIXU_TABLE = "apixu";
- public static final String OWM_TABLE = "owm";
- public static final String WU_TABLE = "wu";
- public static final String COL_1 = "id";
- public static final String COL_2 = "value";
3.3.6.2 Methods
- DatabaseHelper(Context context)
- onCreate(SQLiteDatabase db)
- onUpgrade(SQLiteDatabase db, int i, int i1)
- insertData(Integer[] newFeedValues)
- fetchData()

3.3.7 <ThreeDayForecast>

3.3.7.1 Attributes

- public JSONObject: wuForecast
- public JSONObject: apixuForecast
- public JSONObject: owmForecast
- public Double[]: feedValues[]
- String: city

3.3.7.2 Methods

- OnCreate()
- fetchWuForecast(String city)
- fetchApixuForecast(String city)
- fetchOwmForecast(String city)
- fetchFeedValues()
- SetTextViews(JSONObject wuForecast, JSONObject apixuForecast, JSONObject owmForecast, Double[] feedValues)

—

The Classes:
MinMaxFragment, ServicesTempsTableFragment, SplashScreen and ThreeDayLoading are not listed above because of the minor importance they have in the project. In order to give the SRS a more efficient readability.

3.4 Non-Functional Requirements

3.4.1 Performance

Because the various background activities that WeatherCompare has to perform during the interactions, has to be more attention to these activities. Such as the API requests or the informations fetch from the Database.
The problem that could occur is that at some point, when the app needs to execute more than one background activity, the performances could slow down a lot causing a bad experience and, in the worst case, crash.
The average time of request must be under 2 seconds.
Speaking of timing about these activities, they have to be in the most part not parallel at all. Weather fetch and database queries have to be executed in different moments. The process of fetch needs to be closed before starting every other process.

The API calls need to handle at least every major city in Italy, the errors caused from typing a not-sorted city or an incorrect named needs to be displayed trough a visive warning.

3.4.2 Security

There are no possible information flaws to keep track of. The system does not use the GPS infos so the only personal information stored in the app is the favourite city, not considered a privacy flaw.

3.4.3 Business Rules

The use of the app relies on people’s genuine feedback, so truthful feedbacks from them are required for the app to work properly.
So the user needs to accept some Terms of Use before sending a feedback.

3.4.4 Constraints

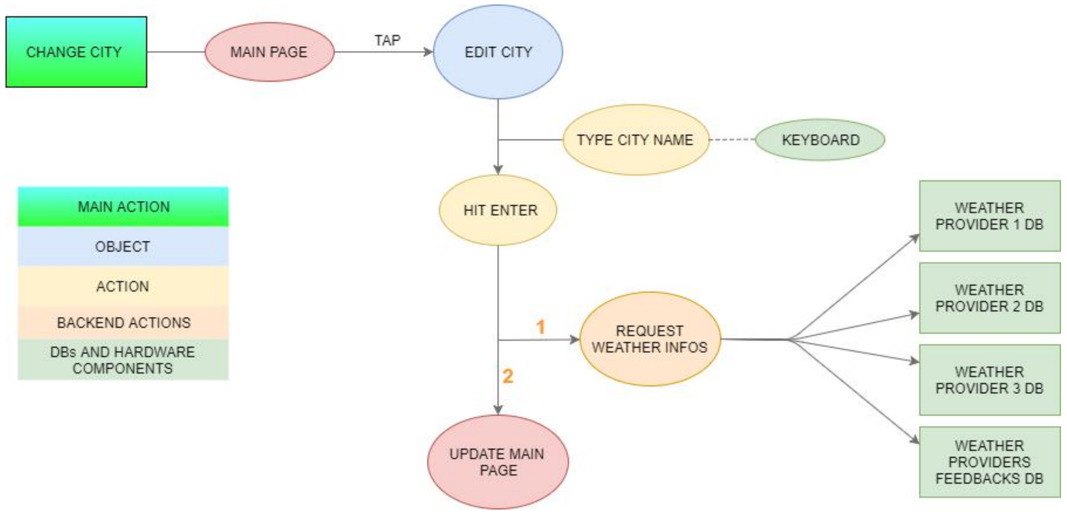
Budget constraints, although the will to develop a fully-functionctional and useful Android Application, budget is an issue. Mostly about the need to buy expensive remote database storage to let the app work properly.

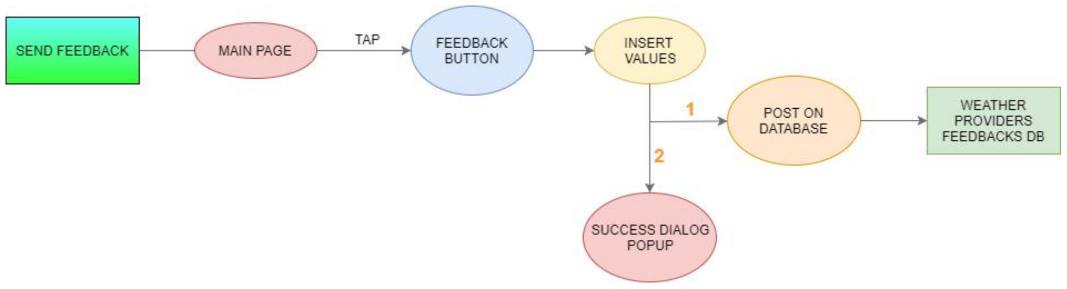
A. Appendices

A.1 Appendix 1 - Analysis Models

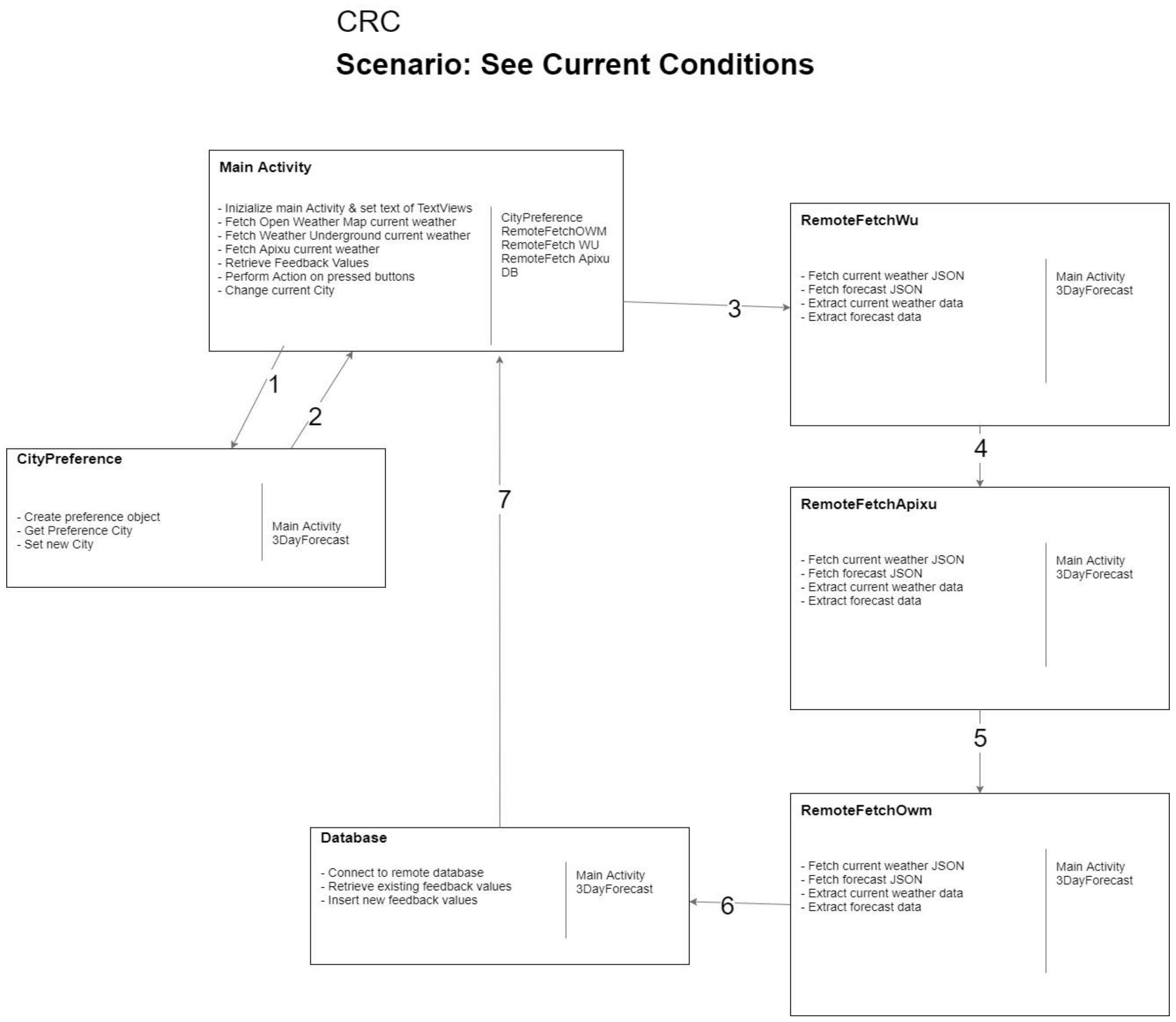
Use Case Diagrams with the support of the UML - *Unified Modeling language* - in order to highlight every actor, and micro-actions related to them, involved in every macro-action that the user could possibly execute.
CRC Cards - *Class Responsabilities Collaborators Cards* - to help understanding how to structure the Object Oriented Programming language - Java - development. Expliciting every relationship between objects and actions helps code first realization and development.

A1.1 Representation of Problem Decomposition

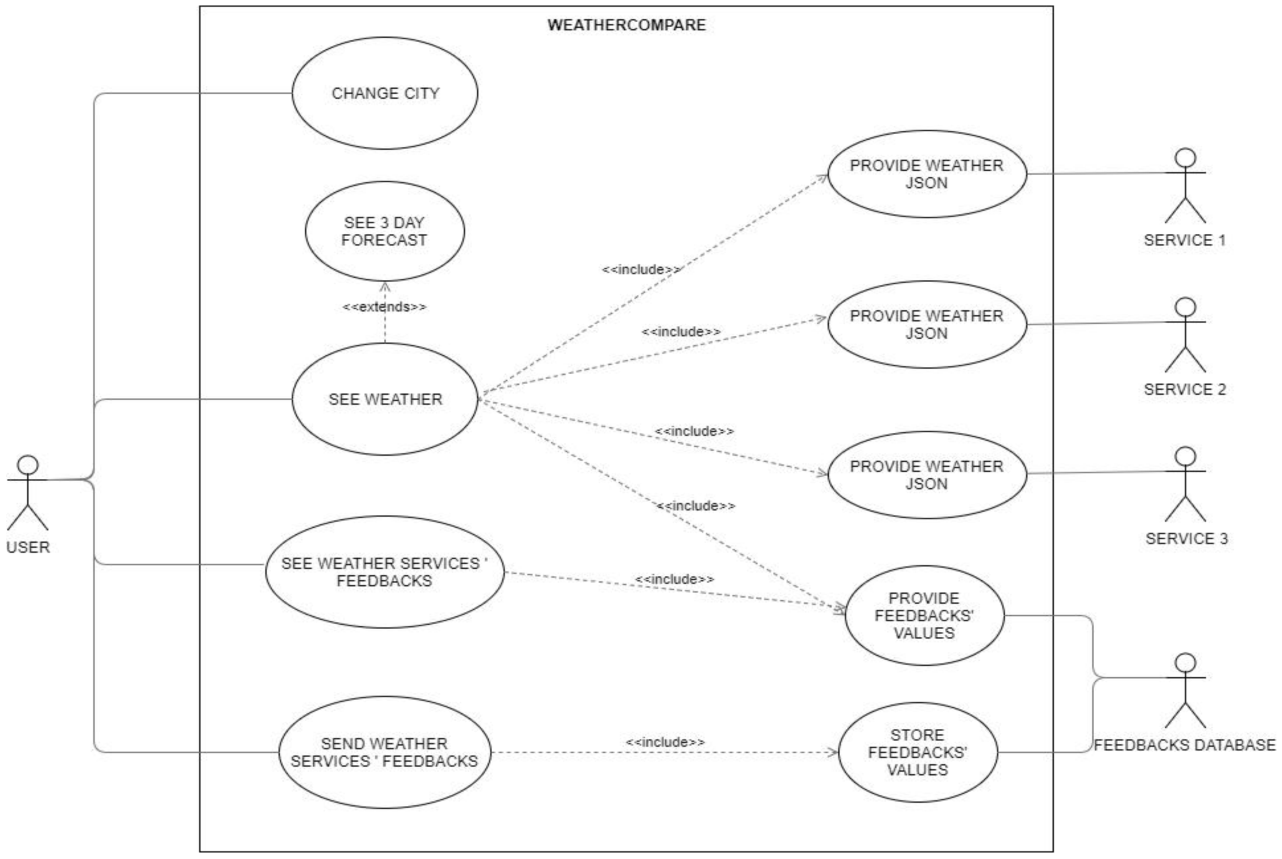


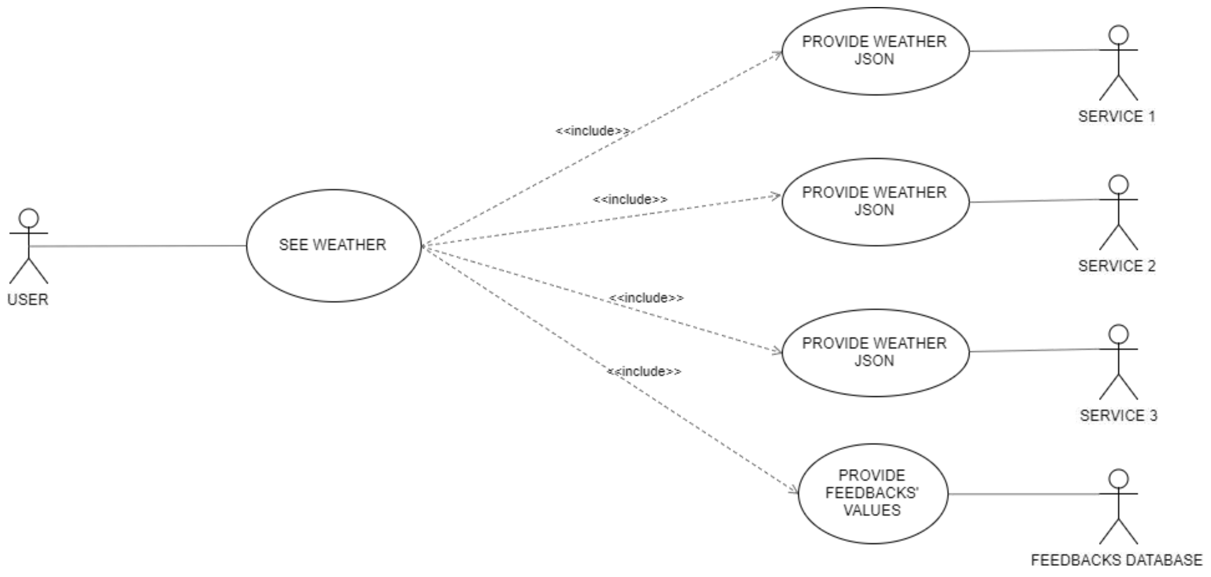


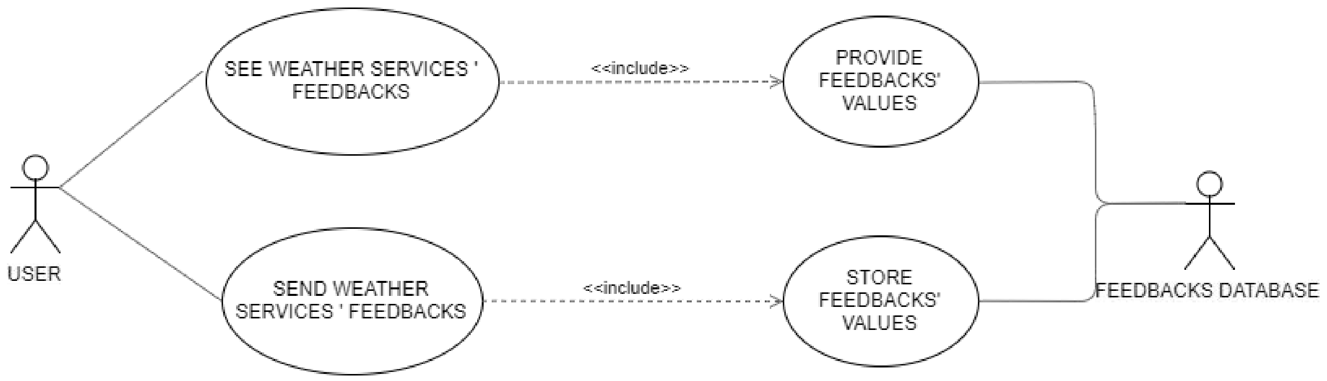
A1.2 CRC


CRC
Scenario: See Current Conditions
Main Activity
- Inizialize main Activity & set text of TextViews
- Fetch Open Weather Map current weather
- Fetch Weather Underground current weather
- Fetch Apixu current weather
- Retrieve Feedback Values
- Perform Action on pressed buttons
- Change current City
CityPreference
RemoteFetchOWM
RemoteFetch WU
RemoteFetch Apixu
DB
RemoteFetchWu
- Fetch current weather JSON
- Fetch forecast JSON
- Extract current weather data
- Extract forecast data
Main Activity
3DayForecast
CityPreference
- Create preference object
- Get Preference City
- Set new City
Main Activity
3DayForecast
RemoteFetchApixu
- Fetch current weather JSON
- Fetch forecast JSON
- Extract current weather data
- Extract forecast data
Main Activity
3DayForecast
RemoteFetchOwm
- Fetch current weather JSON
- Fetch forecast JSON
- Extract current weather data
- Extract forecast data
Main Activity
3DayForecast
Database
- Connect to remote database
- Retrieve existing feedback values
- Insert new feedback values
Main Activity
3DayForecast
1
2
3
4
5
6
7

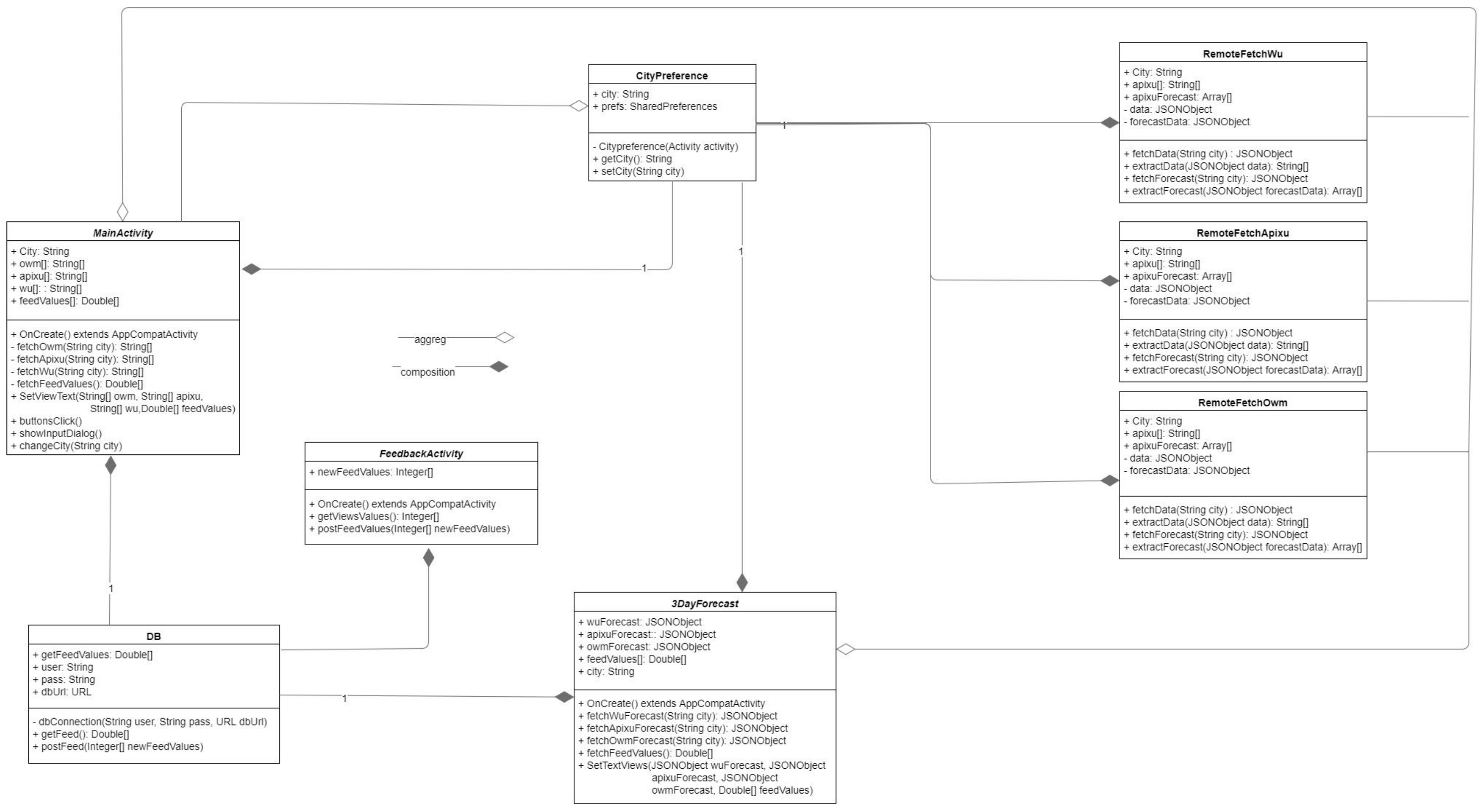
A1.3 Use Case Diagram





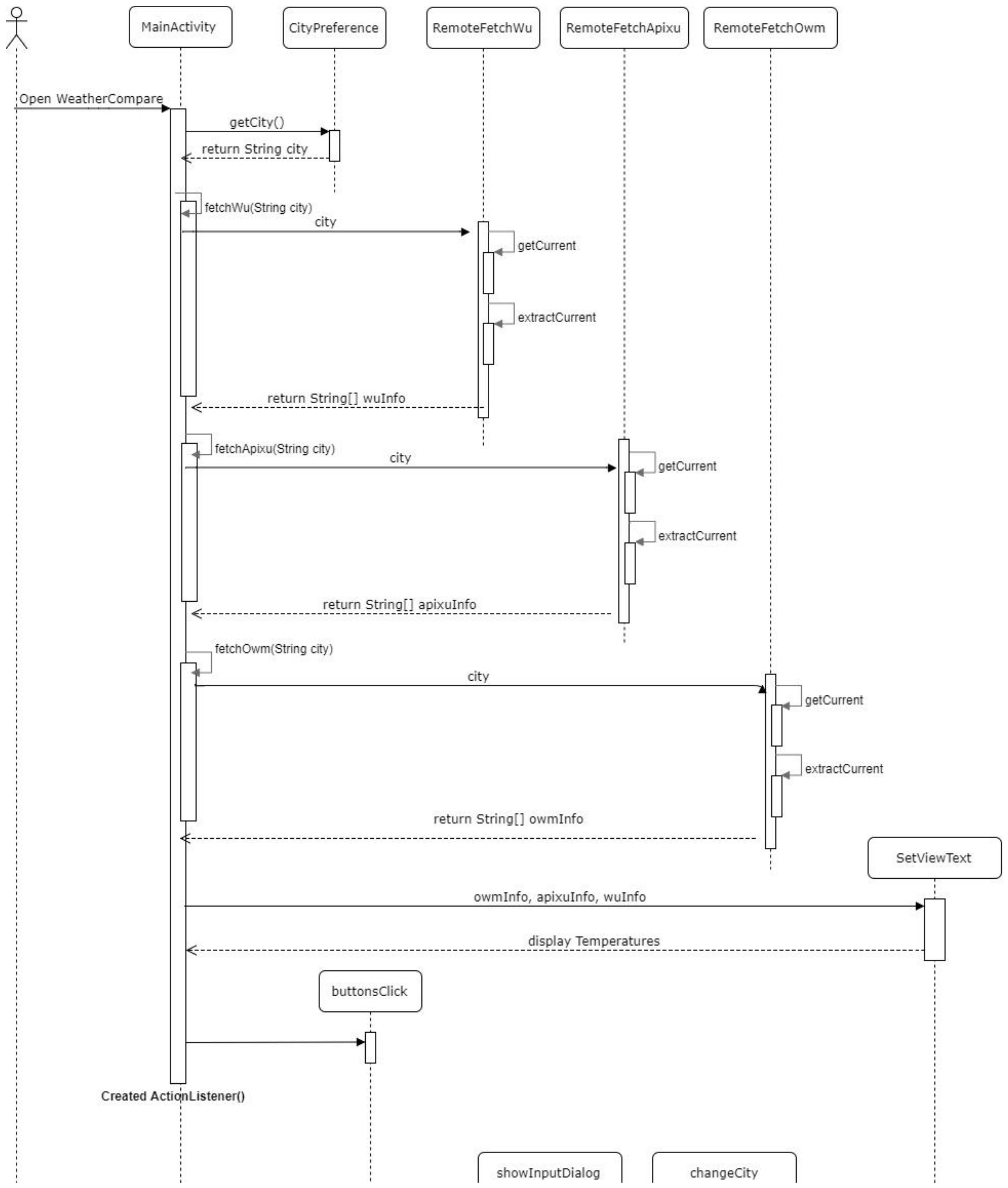

SEE WEATHER SERVICES ' FEEDBACKS
<<include>>
PROVIDE FEEDBACKS' VALUES
USER
FEEDBACKS DATABASE
SEND WEATHER SERVICES ' FEEDBACKS
<<include>>
STORE FEEDBACKS' VALUES

A1.4 Class Diagram

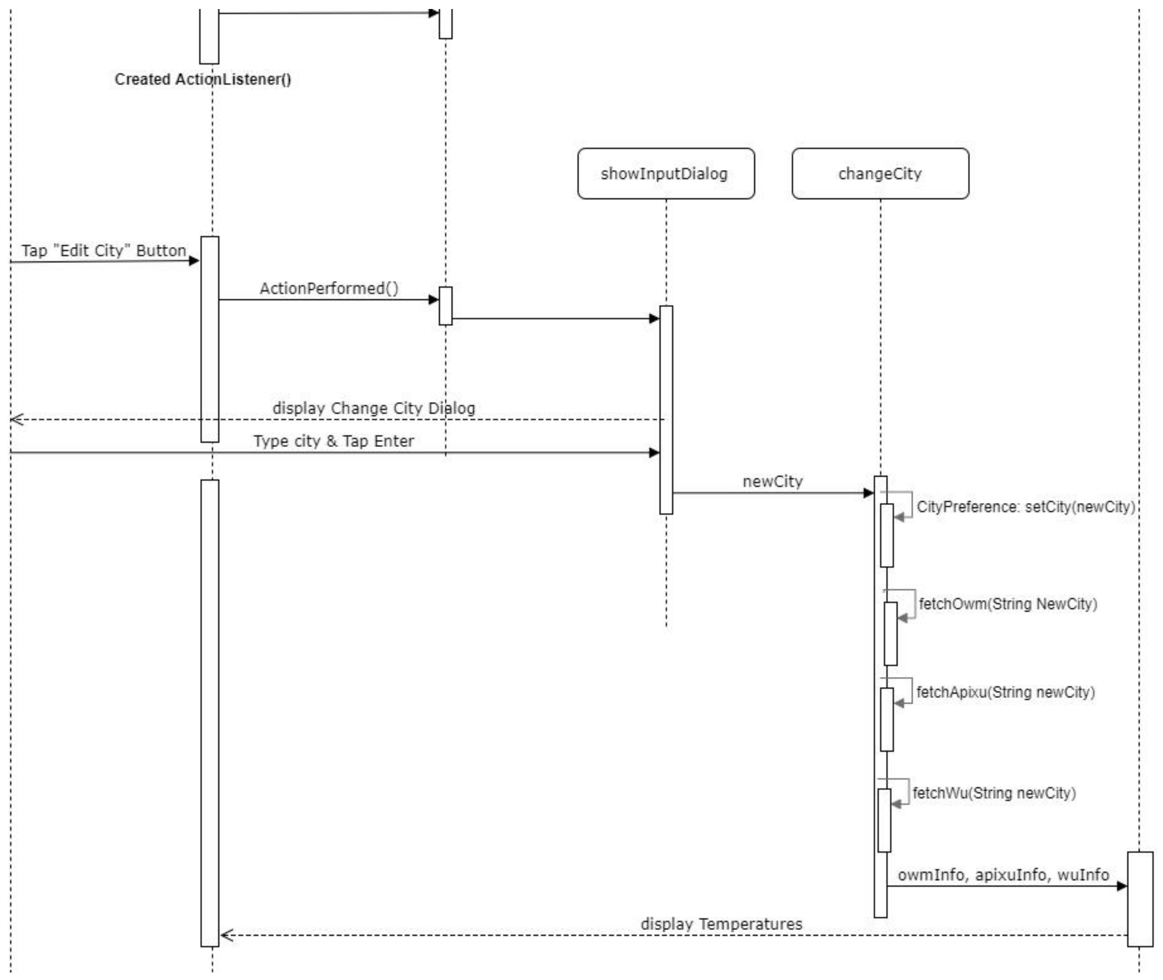

CityPreference
+ city: String
+ prefs: SharedPreferences
- Citypreference(Activity activity)
+ getCity(): String
+ setCity(String city)
RemoteFetchWu
+ City: String
+ apixu[]: String[]
+ apixuForecast: Array[]
- data: JSONObject
- forecastData: JSONObject
+ fetchData(String city) : JSONObject
+ extractData(JSONObject data): String[]
+ fetchForecast(String city): JSONObject
+ extractForecast(JSONObject forecastData): Array[]
RemoteFetchApixu
+ City: String
+ apixu[]: String[]
+ apixuForecast: Array[]
- data: JSONObject
- forecastData: JSONObject
+ fetchData(String city) : JSONObject
+ extractData(JSONObject data): String[]
+ fetchForecast(String city): JSONObject
+ extractForecast(JSONObject forecastData): Array[]
RemoteFetchOwm
+ City: String
+ apixu[]: String[]
+ apixuForecast: Array[]
- data: JSONObject
- forecastData: JSONObject
+ fetchData(String city) : JSONObject
+ extractData(JSONObject data): String[]
+ fetchForecast(String city): JSONObject
+ extractForecast(JSONObject forecastData): Array[]
MainActivity
+ City: String
+ owm[]: String[]
+ apixu[]: String[]
+ wu[]: : String[]
+ feedValues[]: Double[]
+ OnCreate() extends AppCompatActivity
- fetchOwm(String city): String[]
- fetchApixu(String city): String[]
- fetchWu(String city): String[]
- fetchFeedValues(): Double[]
+ SetViewText(String[] owm, String[] apixu, String[] wu,Double[] feedValues)
+ buttonsClick()
+ showInputDialog()
+ changeCity(String city)
aggreg
composition
FeedbackActivity
+ newFeedValues: Integer[]
+ OnCreate() extends AppCompatActivity
+ getViewsValues(): Integer[]
+ postFeedValues(Integer[] newFeedValues)
DB
+ getFeedValues: Double[]
+ user: String
+ pass: String
+ dbUrl: URL
- dbConnection(String user, String pass, URL dbUrl)
+ getFeed(): Double[]
+ postFeed(Integer[] newFeedValues)
3DayForecast
+ wuForecast: JSONObject
+ apixuForecast:: JSONObject
+ owmForecast: JSONObject
+ feedValues[]: Double[]
+ city: String
+ OnCreate() extends AppCompatActivity
+ fetchWuForecast(String city): JSONObject
+ fetchApixuForecast(String city): JSONObject
+ fetchOwmForecast(String city): JSONObject
+ fetchFeedValues(): Double[]
+ SetTextViews(JSONObject wuForecast, JSONObject apixuForecast, JSONObject owmForecast, Double[] feedValues)
1
1
1
1
1

A1.5 Sequence / Collaboration Diagrams

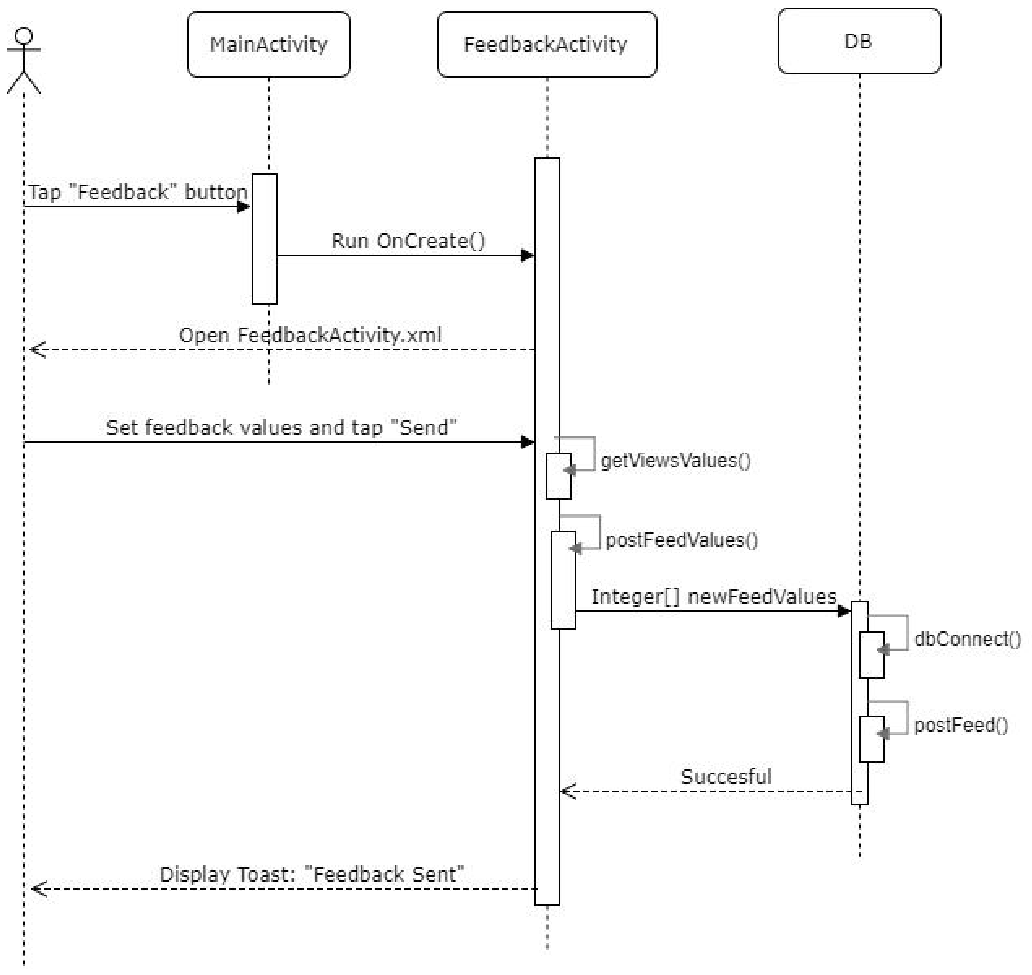
A1.5.1 - Actual Weather



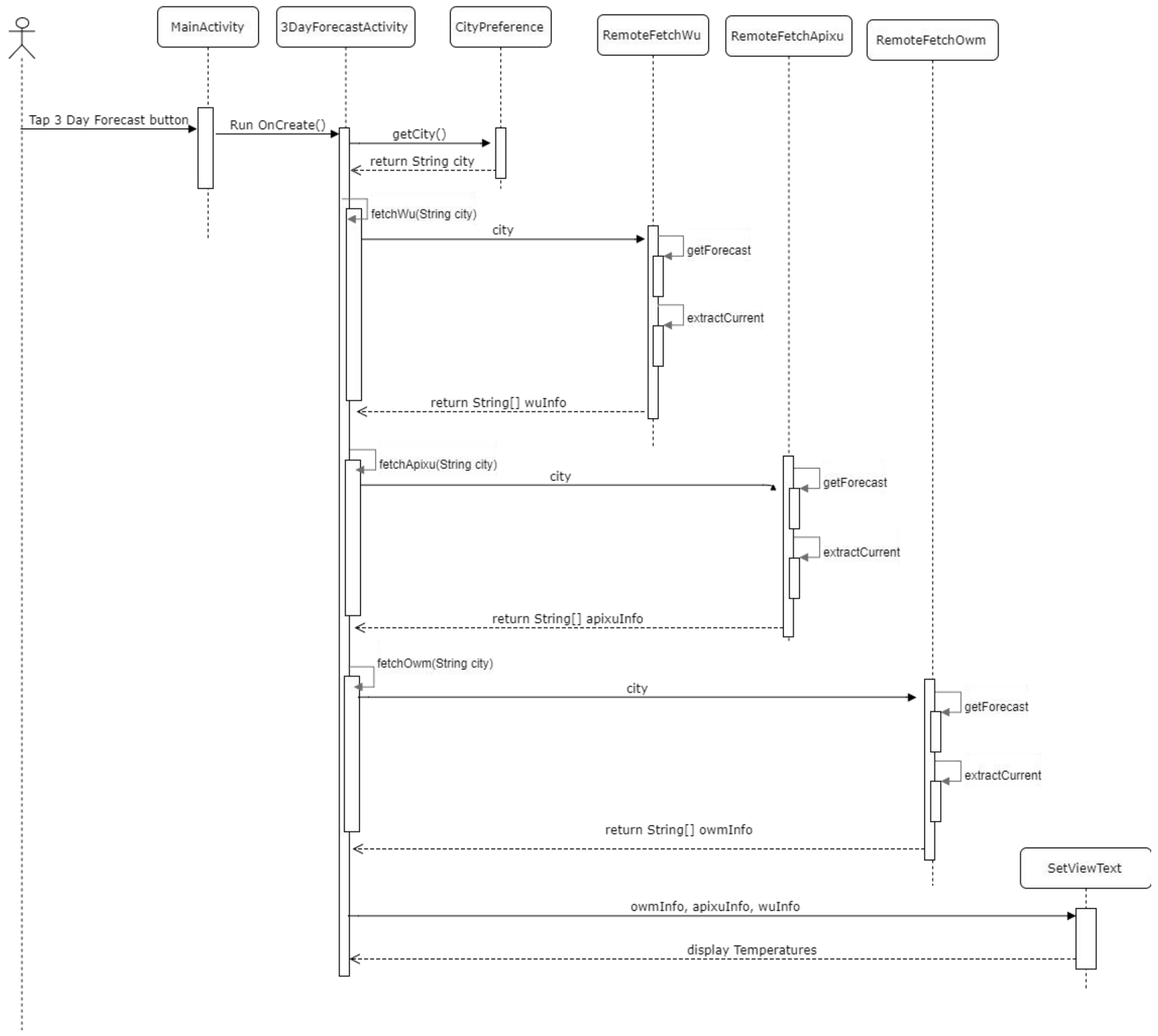
A1.5.2 - Edit City



A1.5.2 - Send Feedback



A1.5.2 - See 3-day forecast


MainActivity
3DayForecastActivity
CityPreference
RemoteFetchWu
RemoteFetchApixu
RemoteFetchOwm
Tap 3 Day Forecast button
Run OnCreate()
getCity()
return String city
fetchWu(String city)
city
getForecast
extractCurrent
return String[] wuInfo
fetchApixu(String city)
city
getForecast
extractCurrent
return String[] apixuInfo
fetchOwm(String city)
city
getForecast
extractCurrent
return String[] owmInfo
SetViewText
owmInfo, apixuInfo, wuInfo
display Temperatures