

Will The Sun Shine Again?

android weather application

Craig Pirie | Computing Science: Application Software Development

Mobile App Development | 19/11/18

# Data Source

I chose the Dark Sky weather API to receive weather forecasts for my application. I chose this service as it was easy to implement and the information it provides contained all the relevant data necessary to build my app.

The service allows for 1000 free downloads per day which is more than enough to build and run my application for this project. If I reach the 1000 call limit the Dark Sky service will not send any more forecasts until the next day. NO cost will be incurred as I have not given them my bank details.

The data I downloaded was a week’s worth of daily weather forecasts including data regarding:-

* Time
* Max Temperature
* Humidity
* Wind Speed
* Wind Direction
* Summary of the Days Forecast.

To call the Dark Sky service I needed to use a key given to me upon creating an account with them and user given values for the latitude followed by the longitude. The key is a URL containing a randomly generated string. An example is as follows: - [https://api.darksky.net/forecast/8785c1badc7ab3be0e0603fed739a14d/37.8267,-122.4233] .

# Implemented Requirements (Tested)

All of the following requirements were implemented and verified by testing, see the test logs attached for a description of how this was done.

## Interlinking Screens

The application has multiple activities that link to each-other through the use of buttons and the android back button. These work by using intents to switch from one activity to the other. Our screens include the main activity screen (the home-screen), the saved locations (preferences screen), the view forecasts screen and the search screen.

## Store Preferred Locations

The user can enter 10 different geographical locations in the form of co-ordinates in the format as [latitude],[longitude]. These locations are stored separately as preferences and remain on the device even after the application is paused, closed or the device is shut down. The address needed to fetch the forecasts is hard-coded into the class, and the location is fetched from preferences and appended to this string when the API is called.

## Download Data from a Web Service

My application uses a given key which appends each of the users preferred locations to itself in order to request the DarkSky web API for weather data which is returned as JSON. The data returned contains a lot of unwanted data such as hourly forecasts or data regarding different aspects of weather than we need so the data was parsed later to pick out the important bits of data and then saved to a database.

## Parse XML or JSON

Dark Sky returns data as JSON so the application parses the downloaded data for a week’s worth of daily weather forecasts and formats the data into a suitable way for user comprehension. It does this using a JSON handler class which checks the downloaded objects for the data we need and then packages it in a way users can understand. The parsed data was then returned to the main activity where it is saved to a database.

## Manage data in a SQLite database

The data is downloaded to a Room database. There are 3 separate classes used to build my database – ForecastsDAO, Forecasts and ForecastsDatabase.

Forecasts is an Entity used to represent a table in our database, in it contains all the attributes, getters and setters for these attributes and a .toString() method to display the contents of a forecast instance. Each downloaded piece of data that will be displayed to the user is saved to the database alongside a primary key ‘uid’, which auto-generates and alongside the location preference given for that forecast.

ForecastsDAO is our data access object that allows us to interact with the database. It contains various SQL commands to manipulate our database. These commands include SQL queries to retrieve all our forecasts from the database, search our database for certain keys, delete from the database, update our database and insert forecasts into the database.

ForecastsDatabase contains the actual database class that will manage our database. It pulls together the entity and the DAO and sets the conditions to allow our database to run.

The user can browse through all the saved forecasts in the recycler view in the ‘view forecasts’ activity. They can also select any forecast from the recycler view and delete it using the ‘delete’ button.

The user can refresh the screen by clicking the ‘Refresh’ button to retrieve all the forecasts from the database and display them.

## Search

The user can search for a forecast using any key. I used a separate ‘Search Activity’ to handle this as this was neater. An SQL query resides in the DAO that returns a forecast given all of the keys entered by using edit texts on the search activity. The search activity then packages the forecast and sends it as an intent extra back to the forecasts activity where it is displayed. If no forecast is found matching the data then a toast will be displayed alerting the user of this.

## User interface

I used a variety of controls, views and widgets to design my application. These include:-

* Buttons: I used buttons in almost every activity for uses that range to switching activities, activating an SQL command to downloading data.
* Recycler View: I used a recycler view in the ‘View Forecasts’ activity as this is an efficient and sleek way to display the forecasts with limited hardware usage. I needed to implement a RecyclerView adapter interface to allow items to be able to work the view and I also needed to add an RecyclerViewItem.xml file to define the layout of each forecast.
* Edit Texts: These were used in my shared preferences screen to enter each of the users preferred locations as they are the most suitable widget for text input.
* Image Views: These were used as a simple widget that displays images on the screens.
* Text Views: Were another simple and fundamental widget that was used repeatedly throughout this project to display text in all the activities.

# IMPLEMENTED REQUIREMENTS (NON-TESTED)

There were no requirements implemented that were not tested.

# NON-IMPLEMENTED REQUIREMENTS

All requirements were implemented and tested.