Week 7 Lecture

Making Network Requests & Loading Images

01

Project Demo

What are we building this week?

02

Making Network Requests

Loading remote data from OpenWeatherMap api

03

Loading Remote Images

Loading remote images from a URL

Project Demo

What are we building this week?

Week 7 Project Updates

- Load current forecast data from OpenWeatherApi
- Load 7-day forecast data from OpenWeatherApi
- Load and display forecast icons from a remote URL

Making Network Requests

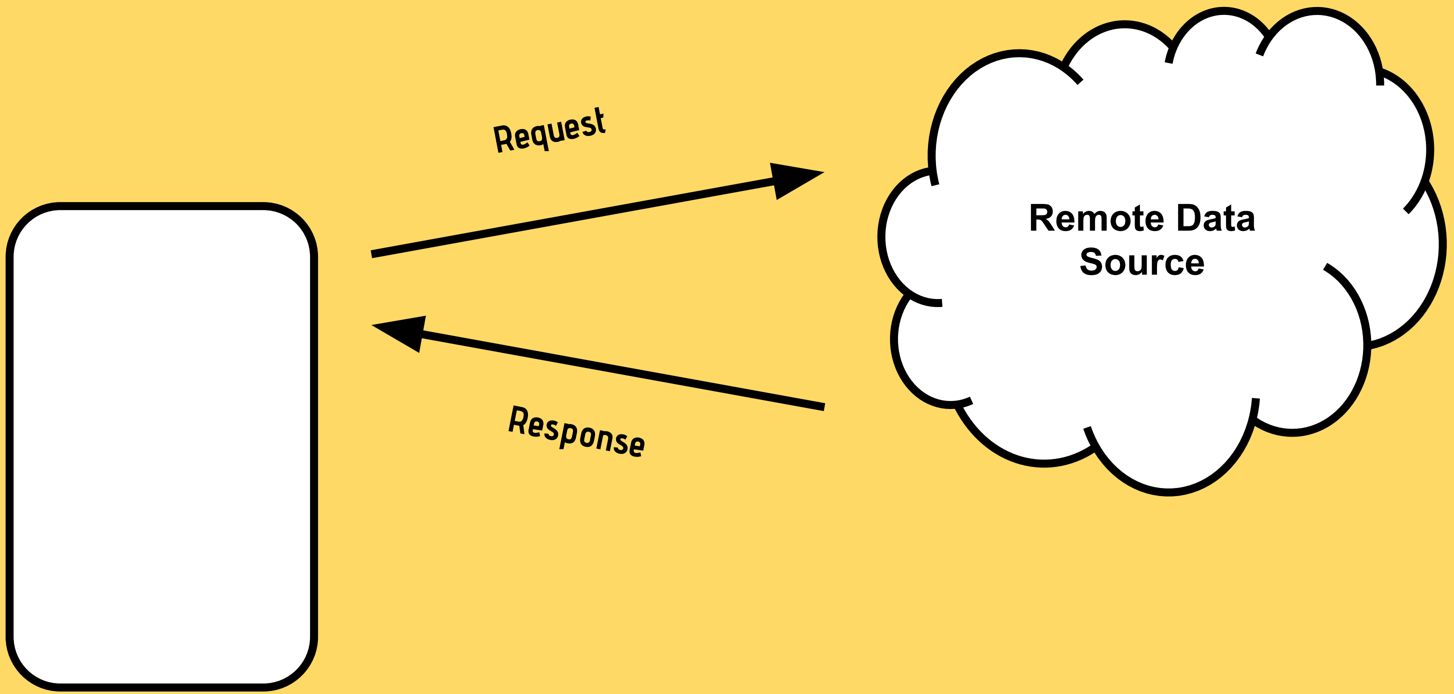
Loading remove data from OpenWeatherMap api

How do we load data from a remote source?

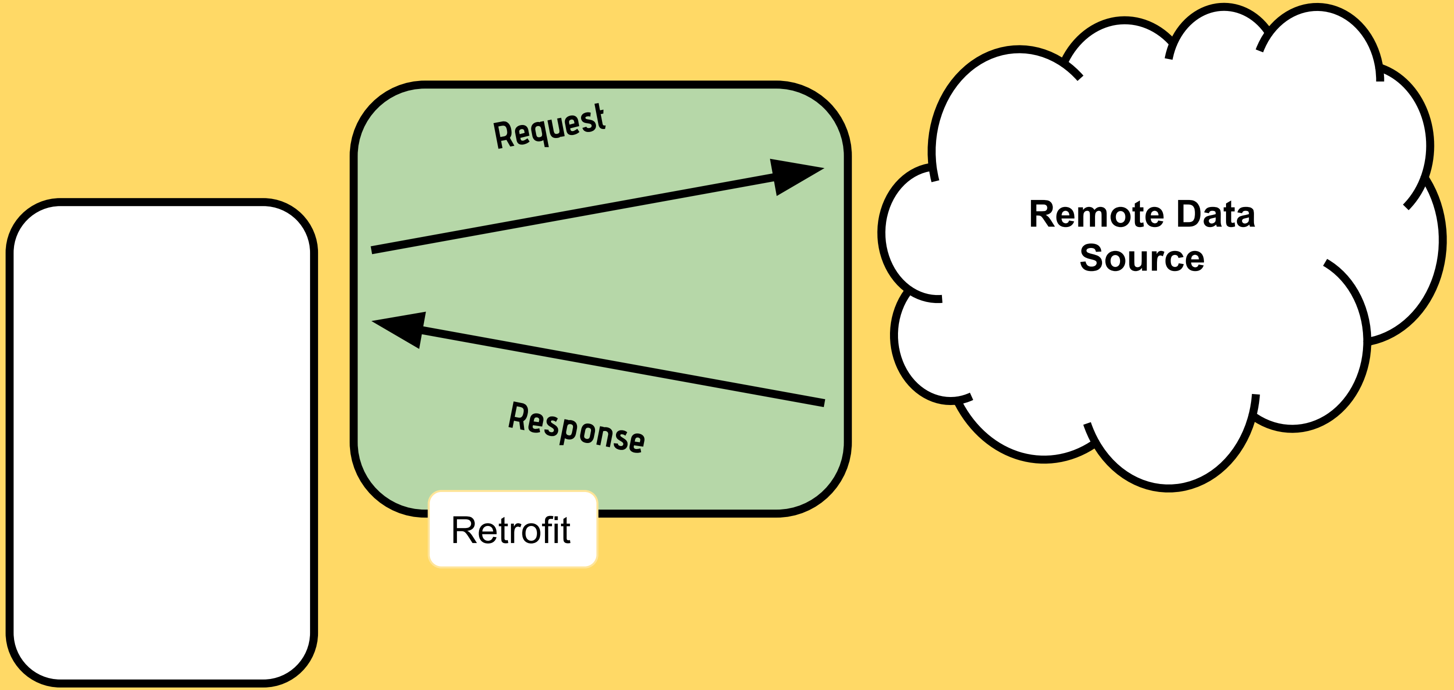
Retrofit

- HTTP client for Android & Java
- Open-source library from Square

Retrofit



Retrofit



Retrofit

Api Interface

Define your HTTP api using a Kotlin/Java interface

Retrofit.Builder

The Retrofit.Builder class will help generate an implementation of your api service

Call

The Call class lets you make synchronous & asynchronous requests to your api

Define Your API Interface

```
data class Forecast(val temp: Float)
data class Coordinates(val lat: Float, val lon: Float)

/**
* Api response for OpenWeatherMap's /weather endpoint
*/
data class CurrentWeather(
   val name: String,
   val coord: Coordinates,
   @field:Json(name = "main") val forecast: Forecast
)
```

Define Your API Interface

```
/**
* http://api.openweathermap.org/data/2.5/weather?zip=98119&units=imperial&appid=<apikey>
*/
@GET("/data/2.5/weather")
fun currentWeather(
   @Query("zip") zipcode: String,
   @Query("appid") apiKey: String,
   @Query("units") units: String
) : Call<CurrentWeather>
```

Create Your API Implementation

```
val retrofit = Retrofit.Builder()
   .baseUrl("http://api.openweathermap.org")
   .addConverterFactory(MoshiConverterFactory.create())
   .build()

return retrofit.create(OpenWeatherMapService::class.java)
```

Create Your API Implementation

```
val call = weatherService.currentWeather(zipcode, “some api key”, "imperial")
call.enqueue(object : Callback<CurrentWeather> {
   override fun onFailure(call: Call<CurrentWeather>, t: Throwable) {
       Log.e(ForecastRepository::class.java.simpleName, "error loading current weather", t)
   }

   override fun onResponse(call: Call<CurrentWeather>, response: Response<CurrentWeather>) {
       val weatherResponse = response.body()
       if (weatherResponse != null) {
           // handle the loaded weather
       }
   }
})
```

Threading On Android

Threading On Android

- A Thread is an independent path of execution within your code
- Instructions are run 1 by 1
- An instruction can’t start until the previous one is finished; in this case, the 2nd instruction is “blocked” by the first

Threading On Android

- Android apps run on the Main Thread by default
- Every 16ms the UI will draw itself
- Block the Main Thread, and you will cause slowdowns and jank in your UI
- Block the Main Thread for too long, and you'll receive an "Activity Not Responding" dialog

Threading On Android

- To avoid blocking the Main Thread, long running tasks should be run on a background thread
- Threads, Executors, Runnables, AsyncTask, RxJava, Coroutines, Work Manager - a lot of ways to do background work
- We will use Retrofit for this week's assignment

Retrofit Call Callbacks

- The Retrofit Call class enables us to asynchronously load data
- Pass a Callback to be notified when your request is complete
- The work will be done on a background thread and you will be notified on the Main Thread

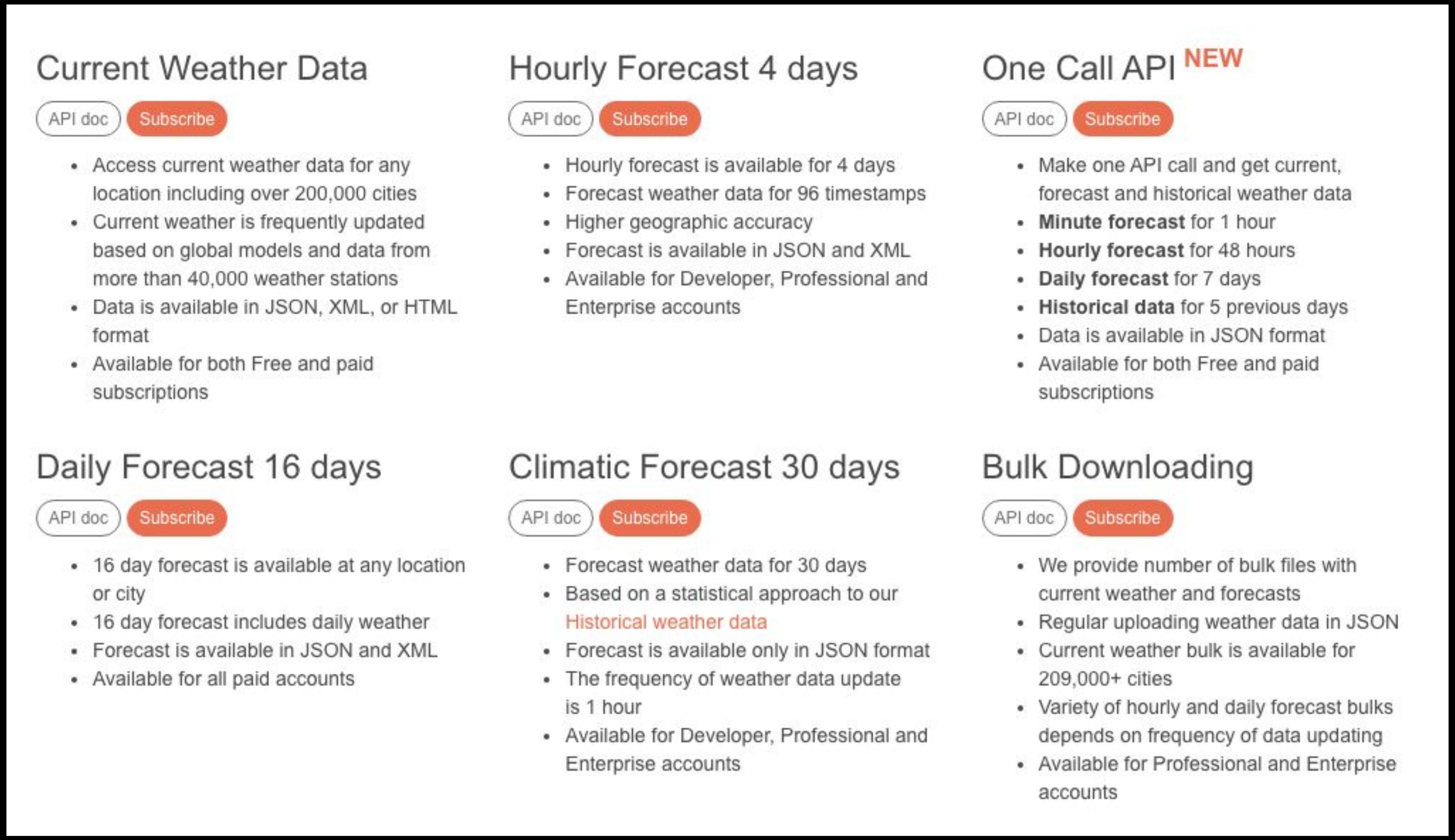
Create Your API Implementation

```
val call = weatherService.currentWeather(zipcode, “some api key”, "imperial")
call.enqueue(object : Callback<CurrentWeather> {
   override fun onFailure(call: Call<CurrentWeather>, t: Throwable) {
       Log.e(ForecastRepository::class.java.simpleName, "error loading current weather", t)
   }

   override fun onResponse(call: Call<CurrentWeather>, response: Response<CurrentWeather>) {
       val weatherResponse = response.body()
       if (weatherResponse != null) {
           // handle the loaded weather
       }
   }
})
```

OpenWeatherMap Api

Load dynamic weather data



Current Weather Data

- Access current weather data for any location including over 200,000 cities
- Current weather is frequently updated based on global models and data from more than 40,000 weather stations
- Data is available in JSON, XML, or HTML format
- Available for both Free and paid subscriptions

Hourly Forecast 4 days

- Hourly forecast is available for 4 days
- Forecast weather data for 96 timestamps
- Higher geographic accuracy
- Forecast is available in JSON and XML
- Available for Developer, Professional and Enterprise accounts

One Call API NEW

- Make one API call and get current, forecast and historical weather data
- **Minute forecast** for 1 hour
- **Hourly forecast** for 48 hours
- **Daily forecast** for 7 days
- **Historical data** for 5 previous days
- Data is available in JSON format
- Available for both Free and paid subscriptions

Daily Forecast 16 days

- 16 day forecast is available at any location or city
- 16 day forecast includes daily weather
- Forecast is available in JSON and XML
- Available for all paid accounts

Climatic Forecast 30 days

- Forecast weather data for 30 days
- Based on a statistical approach to our Historical weather data
- Forecast is available only in JSON format
- The frequency of weather data update is 1 hour
- Available for Developer, Professional and Enterprise accounts

Bulk Downloading

- We provide number of bulk files with current weather and forecasts
- Regular uploading weather data in JSON
- Current weather bulk is available for 209,000+ cities
- Variety of hourly and daily forecast bulks depends on frequency of data updating
- Available for Professional and Enterprise accounts

OpenWeatherMap API

- https://openweathermap.org/api
- /weather - returns current weather
- /onecall - returns many things including 7-day forecast

Loading Remote Images

Loading remote images from a URL

Loading Remote Images

- App resources can't all be packaged with an app
- Images/Video/Music/etc must be loaded from the network
- Multiple libraries available to load remote images in our app

Image Loading Libraries

- Glide
- Picasso
- Coil

Loading An Image With Coil

```
imageView.load("<image url>")
```

Demo