

# CS 473 - MDP

## Mobile Device Programming

© 2021 Maharishi International University

**All course materials are copyright protected by international copyright laws and remain the property of the Maharishi International University. The materials are accessible only for the personal use of students enrolled in this course and only for the duration of the course. Any copying and distributing are not allowed and subject to legal action.**



Maharishi International  
University

# CS 473 - MDP

## Mobile Device Programming

MS.CS Program  
Department of Computer Science  
**Renuka Mohanraj , Ph.D.**



Maharishi International  
University

# CS 473 – MDP

## Mobile Device Programming

### Lesson 8

## WebView, Preferences and JSON



Maharishi International  
University

# Wholeness of the lesson

- WebView helps to displays web page. WebView class is the basis upon which you can roll your own web browser or simply display some online content within your Activity. *The ultimate provider of tools for the creation of beautiful and functional content is pure intelligence itself; all creativity arises from this field's self-interacting dynamics. How the android WebKit rendering engine to display web pages in this way when the knower's awareness is permanently identified with the field of pure intelligence.*

# Contents

- WebView
  - Introduction
  - Hands on Example
  - Methods in WebViewClient class
- DataStore
  - Introduction
  - Applications of Preferences
  - Steps for creating Preferences DataStore
  - Kotlin Coroutines
  - Internal Storage
- JSON, Gson
- Retrofit to read data from Network - JSON data

# Introduction

- WebView is one of the UI component in Android, which is used to display webpages in your application.
- If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView
- The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout.
- Use WebView
  - want to display third-party web content,
  - want to avoid leaving your app to open the browser

# Working with WebView

**Adding the Widget :** To add a [WebView](#) to your Application, simply include the <WebView> element in your activity layout.

```
<WebView android:id="@+id/webview"/>
```

**Loading Content Via URL :** To load a web page in the [WebView](#), use [loadUrl\(\)](#).

```
var myWebView = binding.webview  
myWebView.loadUrl("http://www.example.com")
```

However, we also must make one change to AndroidManifest.xml, adding a line where we request permission to access the Internet:

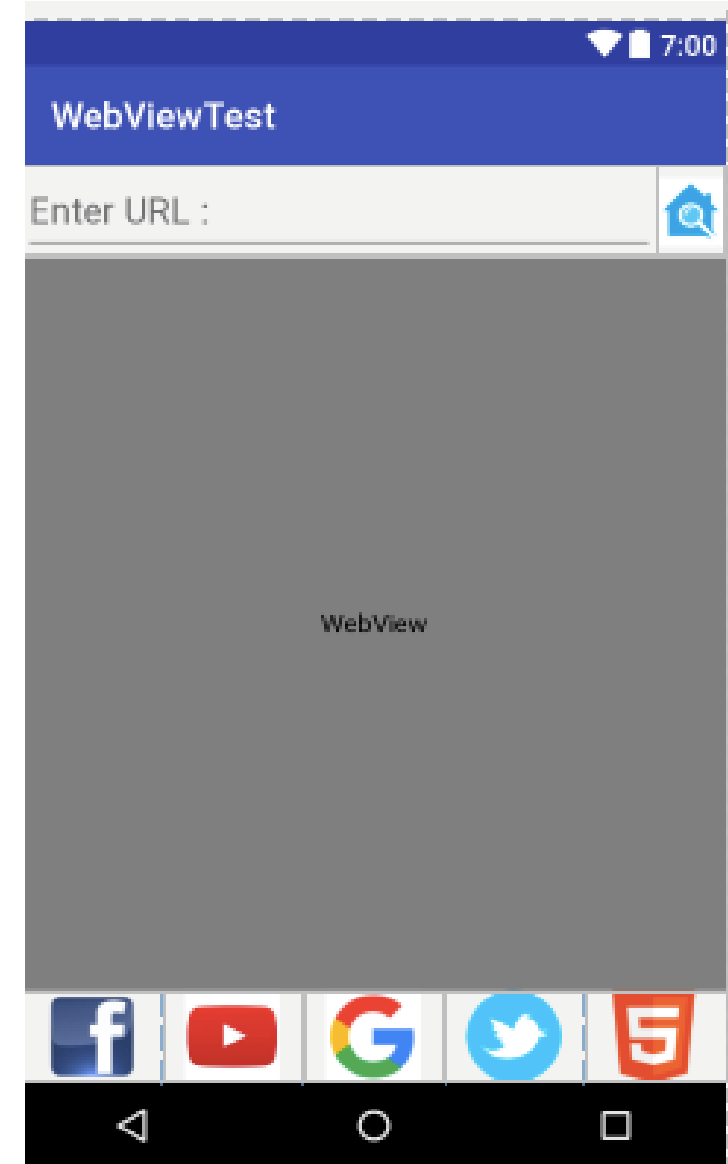
```
<uses-permission android:name="android.permission.INTERNET" />
```

Reading Resource : <https://developer.android.com/guide/webapps/index.html>

# Hands on example-1

**Problem :** Type the expected URL in the EditText UI and click the search button to load the webpage in the WebView component. Below the WebView there are five Images to open the specified URL mostly preferred by the users. Click that image to load the web page in the WebView component.

Refer : Lesson8\WebViewTest





# MainActivity.java

```
/*test method decide action based on the image view selected to load the  
Webpage in WebView UI*/
```

```
fun test(v:View) {
```

```
    when (v.getId()) {
```

```
        R.id.srch -> {
```

```
            /* Load the EditText URL to the WebView UI, it is mendatory to  
mention http:// either here or at the runtime in EditText*/
```

```
            wview.loadUrl("http://" + binding.et1.getText().toString())  
        }
```

```
        R.id.fb -> wview.loadUrl("http://facebook.com")
```

```
        R.id.youtube -> wview.loadUrl("http://youtube.com")
```

```
        R.id.google -> wview.loadUrl("http://google.com")
```

```
        R.id.twitter -> wview.loadUrl("http://twitter.com")
```

```
    } }
```

# WebViewClient

Once you run the app if you click the facebook, youtube, google and twitter image, it will produce the following screen, because just we loaded the URL, but we didn't integrate the browser.

## Handling Page Navigation

- When the user clicks a link from a web page in your WebView, the default behavior is for Android to launch an application that handles URLs.
- Usually, the default web browser opens and loads the destination URL. However, you can override this behavior for your WebView, so links open within your WebView.
- You can then allow the user to navigate backward and forward through their web page history that's maintained by your WebView.



# WebViewClient

- To open links clicked by the user, simply provide a **WebViewClient** for your **WebView** using **setWebViewClient**.
- For example:

```
var myWebView = binding.webview
```

```
myWebView.webViewClient = WebViewClient()
```

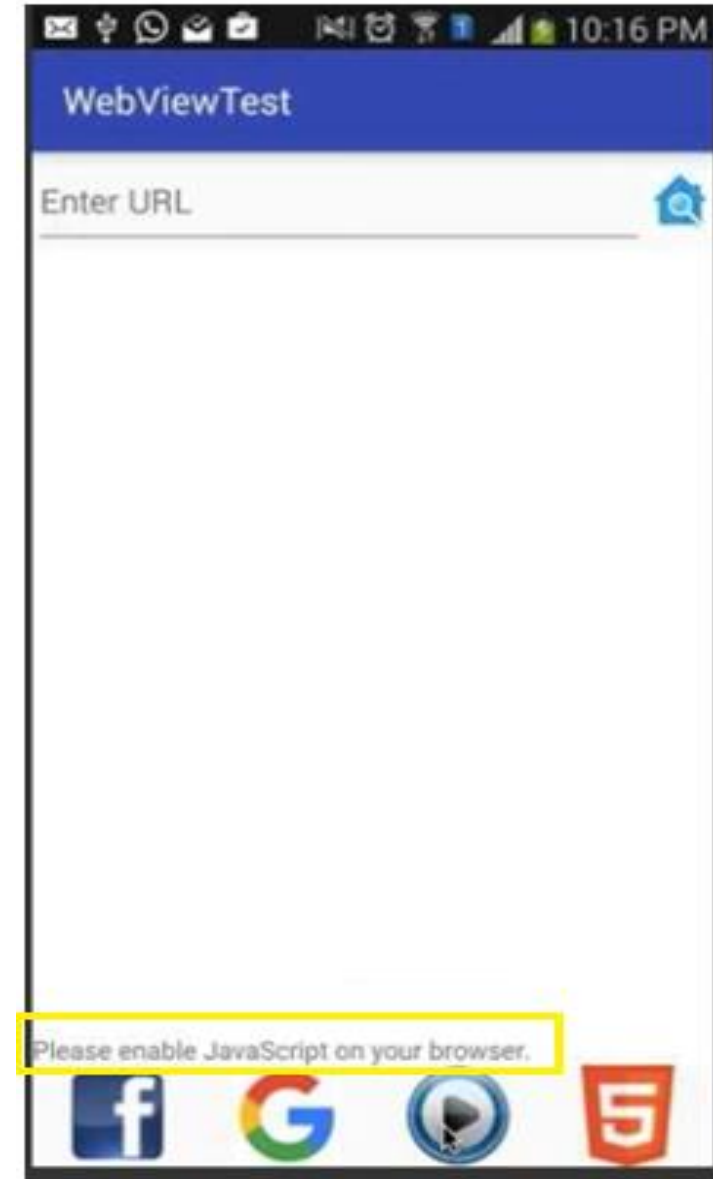
# Supporting JavaScript

- Now, you may be tempted to replace the URL with something else that relies upon JavaScript. You will find that such pages do not work especially well by default.(see the highlighted error).
- That is because, by default, JavaScript is turned off in WebView widgets.
- If you want to enable JavaScript, call the below method on the WebView instance.

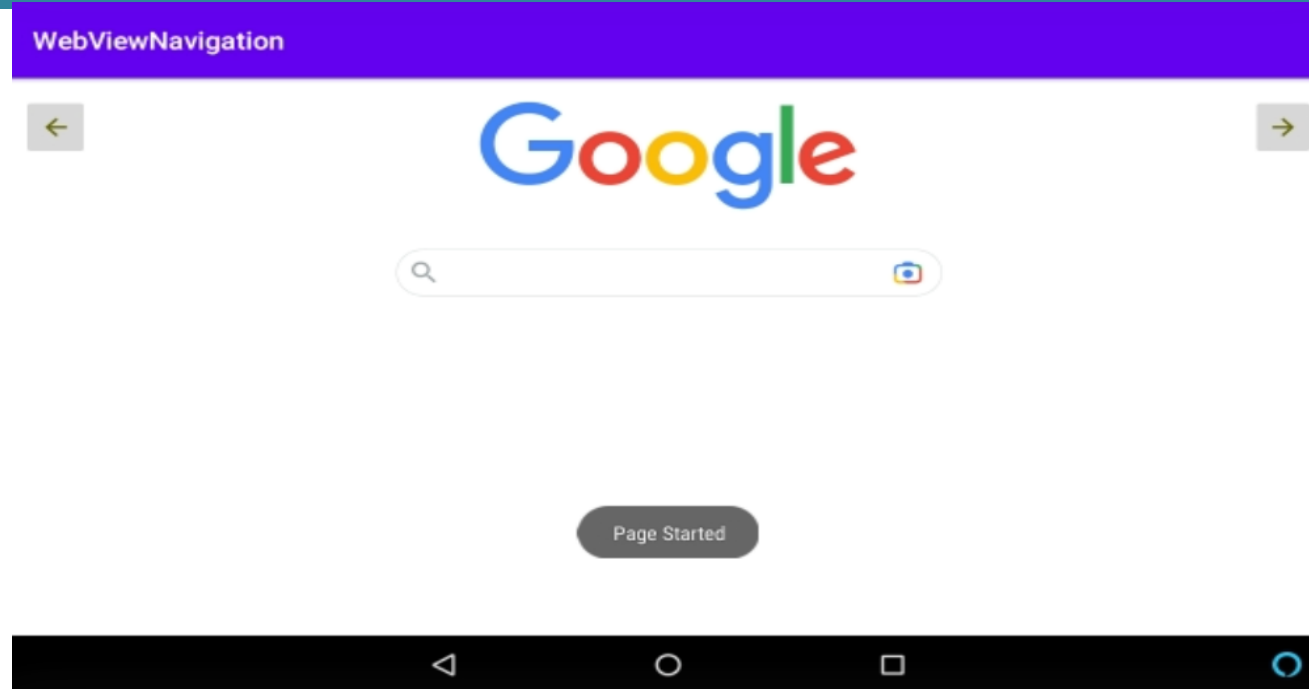
**`wview.settings.javaScriptEnabled = true;`**

- At this point, any JavaScript referenced by your Web page should work normally.
- Sets whether the WebView should use its built-in zoom mechanisms.
- The built-in zoom mechanisms comprise on-screen zoom controls, which are displayed over the WebView's content, and the use of a pinch gesture to control zooming. To enable by giving

**`wview.settings.builtInZoomControls = true;`**



# Hands on Example-2 - Navigation



- The Screen load the url google.com.
- You can browse through this url.
- Click PREVIOUS and NEXT button to navigate.
- If there is no previous and next webpages get a Toast no page available.

Refer : WebPageNavigation

# Methods in WebViewClient class

Create an Inner class with inherit from WebViewClient class, override the below methods according to your application requirements.

1. void onPageStarted () - Notify the host application that a page has started loading.
2. void onPageFinished () - Notify the host application that a page has finished loading.

**For more information refer**

**<https://developer.android.com/guide/webapps/webview#kotlin>**

# WebViewClient Methods

```
inner class MyWebClient : WebViewClient() {  
    // Page Loading started  
    override fun onPageStarted(view: WebView, url: String, favicon: Bitmap?) {  
        Toast.makeText(this@MainActivity, "Page Started",  
            Toast.LENGTH_LONG).show()  
    }  
    // Page Loading finished  
    override fun onPageFinished(view: WebView, url: String) {  
        Toast.makeText(this@MainActivity,  
            "PageFinished", Toast.LENGTH_LONG).show()  
        super.onPageFinished(view, url)  
    }  
}
```

# Navigating web page history

- When your WebView overrides URL loading, it automatically accumulates a history of visited web pages.
- You can navigate backward and forward through the history with `goBack()` and `goForward()`.
- The `canGoBack()` method returns true if there is web page history for the user to visit. Likewise, you can use `canGoForward()` to check whether there is a forward history. If you don't perform this check, then once the user reaches the end of the history, `goBack()` or `goForward()` does nothing.



# MainActivity.java

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    binding = ActivityMainBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
    url = "http://google.com"  
    binding.webView.loadUrl(url)  
    binding.webView.webViewClient = MyWebClient()  
    binding.webView.settings.javaScriptEnabled = true  
    binding.webView.settings.builtInZoomControls = true  
    binding.btnprev!!.setOnClickListener {  
        if (binding.webView.canGoBack()) {  
            binding.webView.goBack()    }  
        else{  
            Toast.makeText(this,"NO previous history available",Toast.LENGTH_LONG).show() } }  
    btnnext!!.setOnClickListener {  
        if (binding.webView.canGoForward()) {  
            binding.webView.goForward()  }  
        else{  
            Toast.makeText(this,"NO back history available",Toast.LENGTH_LONG).show() }  
    }  
}
```

# Main Point 1

Useful methods in WebViewClient class are onPageStarted() and onPageFinished() helps to perform something when the page is started, new Url is loaded and the page has finished loading. ***Science of Consciousness:*** *Similarly, we start TM by closing eyes, then start thinking Mandra, allow thoughts to come and go along with the mantra for twenty minutes and slowly come out by taking 2-3 minutes of silent to finish meditation.*

# DataStore

# Introduction

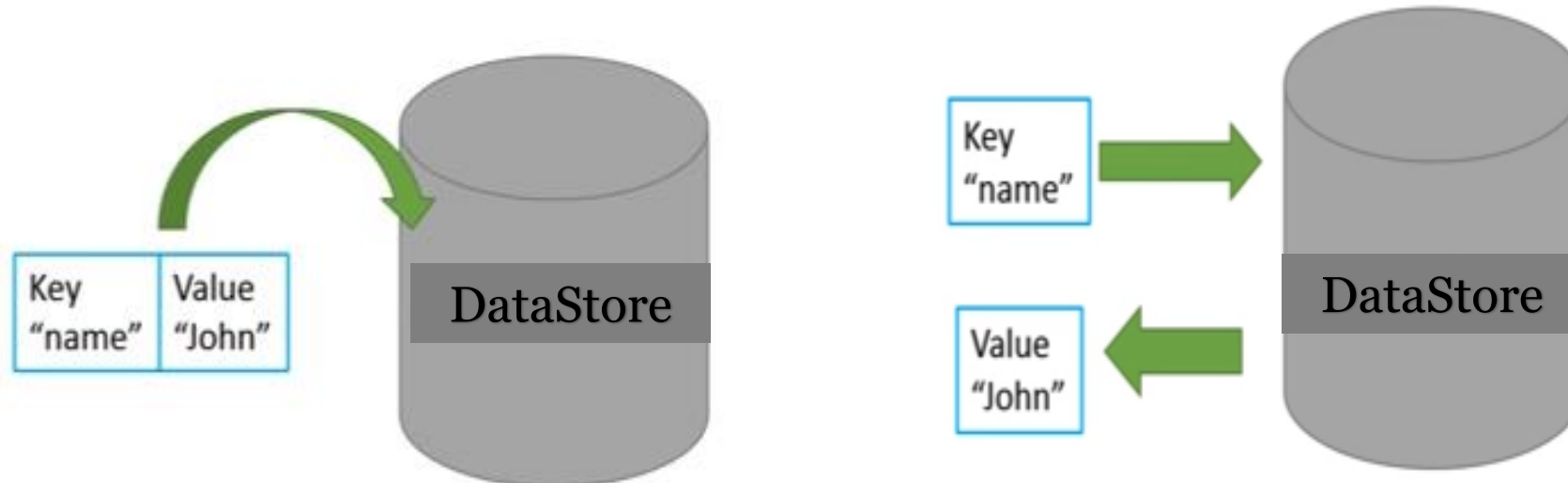
- Android provides several options for you to save your app data:
  - **DataStore Preferences:** Store private, primitive data in key-value pairs.
  - **Internal Storage:** Store data on the device memory. Data will not be accessed by other application.
  - **External Storage:** Store files on the shared external file system. This is usually for shared user files, such as photos. (SD Card)
  - **Database:** Store structured data in a private database using Room.
  - **Network Connection:** Store data on the Web with your own network server.
  - Refer about storage : <https://developer.android.com/guide/topics/data/data-storage.html>

# What is DataStore?

- DataStore is a new and improved data storage solution aimed at replacing SharedPreferences.
- Built on Kotlin coroutines and Flow, DataStore provides two different implementations:
  - **Proto DataStore**, that stores **typed objects for complex data** (backed by protocol buffers), with type safety
  - **Preferences DataStore**, that stores **key-value pairs** and does not offer type safety.
- Data is stored asynchronously, consistently, and transactionally, overcoming some of the drawbacks of SharedPreferences.
- The Preferences DataStore implementation uses the DataStore and Preferences classes to persist simple key-value pairs to disk.

# Preferences DataStore

- Android provides DataStore object to help the developer to save simple application data.
- Using the DataStore object, the developer can save and retrieve the desired data using key/value pairs.



# Difference between SharedPreferences and Preferences DataStore

| Feature           | <u>SharedPreferences</u>   | <u>Preferences DataStore</u>   |
|-------------------|--|--|
| Async API         | only for getting updates on changed values via <u>OnSharedPreferenceChangeListener</u> . It Invoked on Main Thread and possibility to block the UI thread. | Save and retrieve data using the power of Kotlin Coroutines and Flow. Reduce the risk of blocking the thread.                              |
| Synchronous work  | Its synchronous commit function for modifying persisted data may appear safe to call on the UI thread, but it does in fact perform heavier I/O operation   | Does not use synchronous support. Saves the preferences in the file and perform data operations in the background without blocking the UI. |
| Error Handling    | can throw parsing errors as runtime exceptions, leaving your app vulnerable to a crash.  | allowing you to safely catch and handle exceptions when reading or writing data  |
| Data Consistency  | don't guarantee data consistency in a multithreaded environment  | <u>DataStore</u> is a fully transactional API that provides strong consistency guarantees for data operations.                             |
| Migration Support | Does not have built-in migration mechanism   | Allows easy and quick data <u>migrations</u>   |
| Type safety       | Does not offer Type <u>safety</u> protection   | Does not offer Type <u>safety</u> protection   |

# Applications of Preferences DataStore

## Applications

- Last data user entered in your application
- Store the last updates of date and time
- Credentials – Remember user details like username and password
- Location catching – Identify the last location
- Store the login pattern
- Game's high score and current level.

**Not Recommend:** By using Preferences, we can not maintain huge amount of data , but for every value we must give a unique key its difficult to remember & assign new keys for every value , that's why Preferences is preferred to maintain the limited amount of data.

- If you want to maintain huge amount of data, Android is preferred to use Local Databases(Room).



# Steps for creating Preference DataStore

## **Step 1: Add the necessary dependencies.**

Update the build.gradle file to add the following the dependencies. Change the latest version number recommended by the Gradle system.

```
// Preference DataStore
implementation "androidx.datastore:datastore-preferences:1.0.0"

// Coroutines
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.1'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.1'

// Coroutine Lifecycle Scopes
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.0"
implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.6.0"
```

## **Step 2 : Get an instance of Preferences DataStore and declare the Keys to store**

- Get the instance of Preference DataStore with the preference data file name using the code below

```
private val loginDataStore by preferencesDataStore(name = "loginstore")
```

- The preferencesDataStore delegate ensures that we have a single instance of DataStore with that name “loginstore” in our application.
- Declaring Keys, example to store Username and password

```
private val NAME = stringPreferencesKey("UNAME")
```

```
private val PASS = stringPreferencesKey("UPASS")
```

UNAME and UPASS are the keys store the value of String. Other types are available like booleanPreferencesKey, intPreferencesKey, longPreferencesKey, floatPreferencesKey etc.,

**Step 3: Perform Read and write data operations. It can be done through Kotlin Coroutines.**

# Kotlin Coroutines

- Why Coroutines
  - In Android, it's essential to avoid blocking the main thread.
  - Perform background tasks asynchronously.
  - The main thread is a single thread that handles all updates to the UI.
  - It's also the thread that calls all click handlers and other UI callbacks.
  - As such, it must run smoothly to guarantee a great user experience.
- The recommended way of managing background threads that can simplify code by reducing the need for callbacks.
- **Flow:** Provides efficient, cached (when possible) access to the latest durably persisted state. In coroutines, a flow is a type that can emit values sequentially, as opposed to suspend functions that return only a single value. Kotlin flows are similar to Java streams. Flow is nothing but a stream of values that can be computed asynchronously.

# Coroutine Terminology

- **Coroutine Scope:** (Interface)  
Defines a scope for new coroutines. Every coroutine builder (like launch, async, etc) is an extension on CoroutineScope and inherits its coroutineContext to automatically propagate all its elements and cancellation. Types of Scope are Global, Life Cycle and View Model.
- **A global CoroutineScope** not bound to any job. Global scope is used to launch top-level coroutines which are operating on the whole application lifetime and are not cancelled prematurely.
- **launch** is a function that creates a coroutine and dispatches the execution of its function body to the corresponding dispatcher.

# Coroutine Terminology

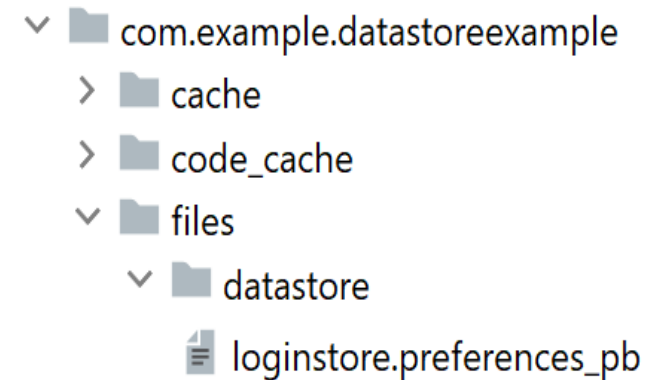
- **Dispatchers**

- Kotlin coroutines use dispatchers to determine which threads are used for coroutine execution.
  - Dispatchers.Main - Use this dispatcher to run a coroutine on the main Android thread. This should be used only for interacting with the UI and performing quick work.
  - Dispatchers.IO - This dispatcher is optimized to perform disk or network I/O outside of the main thread.
  - Dispatchers.Default - This dispatcher is optimized to perform CPU-intensive work outside of the main thread.

- **suspend** keyword enforce a function to be called from within a coroutine or another suspend function. The biggest merit of coroutines is that they can suspend without blocking a thread.

# Where Preferences DataStore file goes?

- Internally SPF will maintain the data in an XML file, we can explore the DataStore file using Android Studio Device File Explorer
- Click View → Tool Windows → Device File Explorer or click the Device File Explorer button in the tool window bar to open the Device File Explorer.
- The file will be saved under your emulator directory  
data/data/<application\_package\_name>/file/datastore folder.
- You can view this file only at the runtime environment.
- You cannot see the stored data once the app is deployed. It's secured.

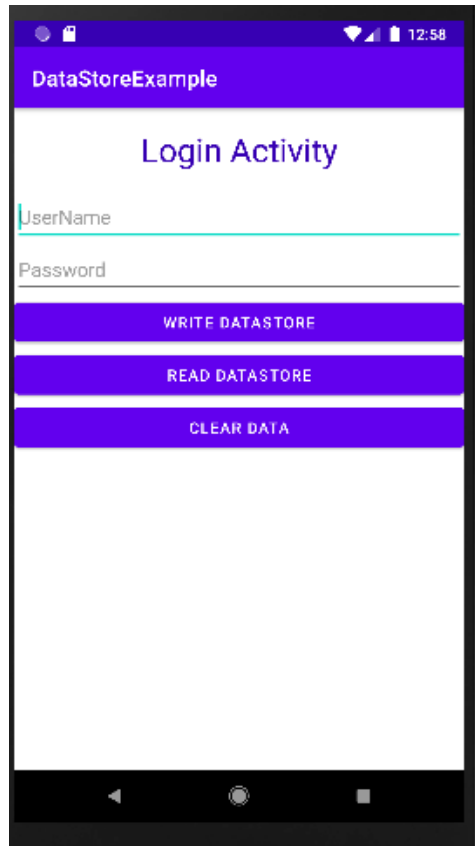


# Hands-on-Examples

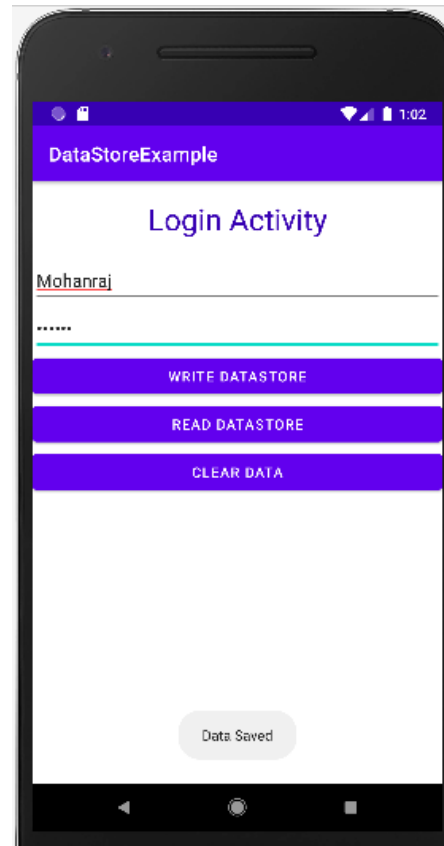
**Problem Requirement :** Design an application which preform to write username and password entered to the Preferences DataStore and read the last stored data from it.

**Refer : DataStoreExample**

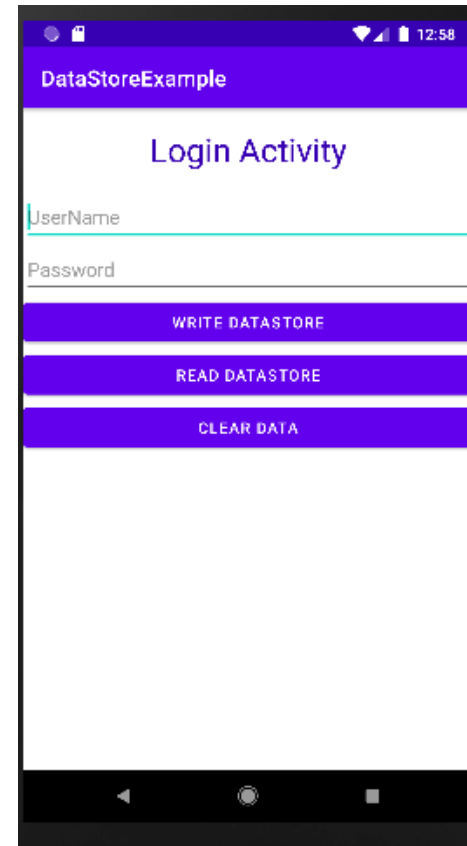
**First Screen**



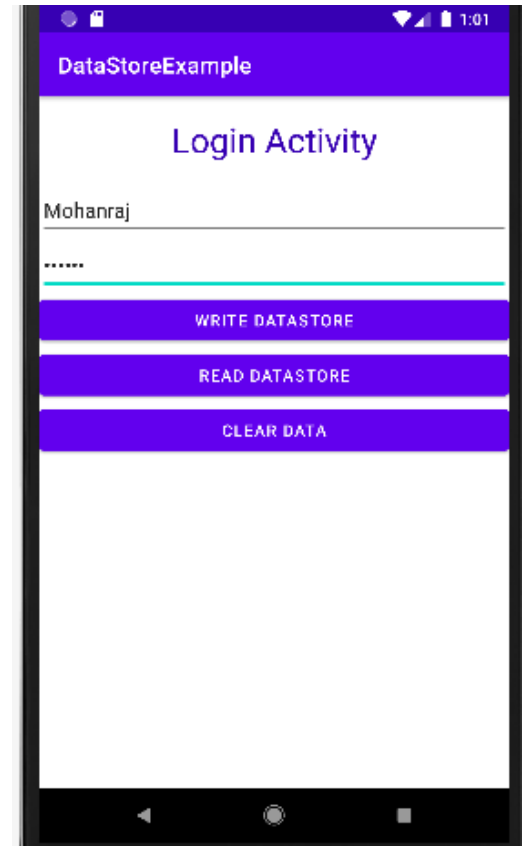
**Write DataStore**



**Clear Data**



**Read DataStore**



# MainActivity.kt

```
class MainActivity : AppCompatActivity() {  
    private lateinit var binding: ActivityMainBinding  
    // Get the instance of Preference DataStore  
    private val loginDataStore by preferencesDataStore(name = "loginstore")  
    // Declaring Keys for the UserName and Password  
    val NAME = stringPreferencesKey("UNAME")  
    val PASS = stringPreferencesKey("UPASS")  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
    }  
}
```



# MainActivity.kt

```
// Read the saved data using coroutine scopes
binding.read.setOnClickListener {
    lifecycleScope.launch {
        val userNameFlow = loginDataStore.data.map {
            it[NAME] ?: "" // Return Flow<String>
        }
        val userPassFlow = loginDataStore.data.map {
            it[PASS] ?: ""
        }
        // Apply the terminal operation first() to get data
        binding.et1.setText(userNameFlow.first().toString())
        binding.et2.setText(userPassFlow.first().toString())
    }
}

// Clear Data
binding.clear.setOnClickListener {
    binding.et1.setText("")
    binding.et2.setText("")
}
```

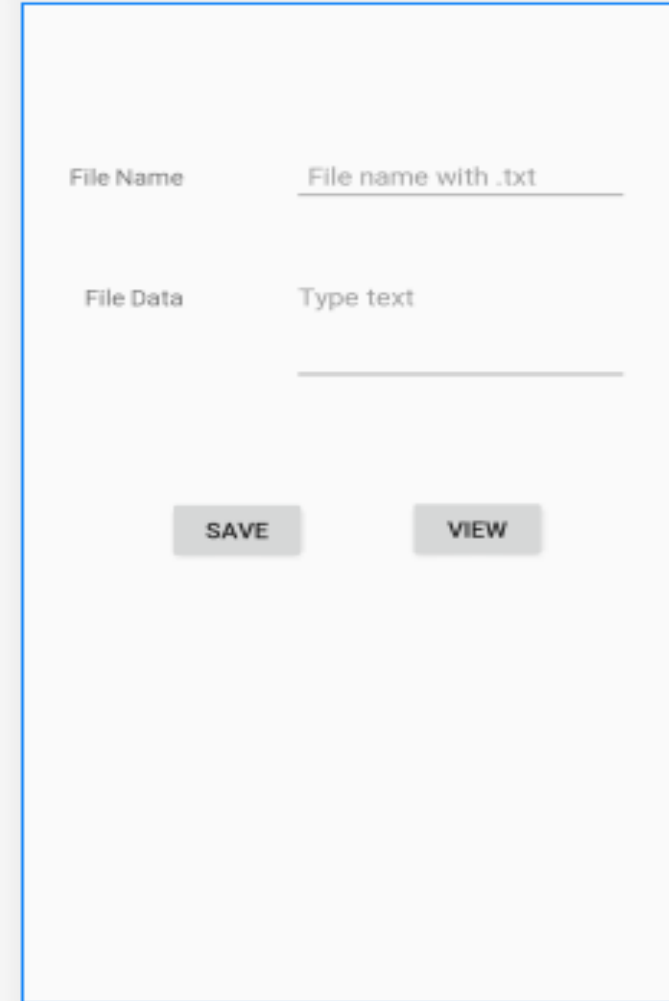
# MainActivity.kt

```
// Writing/Save to Data Store Preferences
binding.write.setOnClickListener {
    lifecycleScope.launch {
        //You can write a separate suspend function or directly implement the logic inside Coroutine Scope
        write(binding.et1.text.toString(), binding.et2.text.toString())
    }
    Toast.makeText(this, "Data Saved", Toast.LENGTH_LONG).show()
}
}

private suspend fun write(name: String, pwd: String) {
    // To store the data using edit
    loginDataStore.edit {
        it[NAME] = name
        it[PASS] = pwd
    }
}
}
```

# Hands on Example - File Read and Write Internal Storage

- Refer InternalStorageDemo
- Give a Filename and type the multiple lines of text data.
- Click the SAVE button to store on the device Internal storage.
- Give the filename to view and Click VIEW to retrieve the data from the file which displayed on the File data UI.
- To see the files from Internal storage,
- Click View → Tools Window → Device File Explorer
- Select the Device you installed
- Click data → data → goto the right package → files



The screenshot shows a mobile application interface with a light gray background. At the top, there is a label "File Name" followed by a text input field containing the text "File name with .txt". Below this, there is a label "File Data" followed by a text input field containing the text "Type text". At the bottom of the interface, there are two buttons: "SAVE" on the left and "VIEW" on the right. Both buttons are gray with white text.

# Main Point 2

Android provides Preferences DataStore to help the developer to save and retrieve simple application data through the use of key/value pairs. ***Science of Consciousness:*** *Transcendental Consciousness provides simple and useful foundation for developing any skill, because regular experience of this field stimulates the flow of thought and action from the total potential of our creativity and intelligence.*

# JSON

- JSON stands for JavaScript Object Notation. It is an independent data exchange format and is the best alternative for XML.
- A modifiable set of name/value mappings. Names are unique, non-null strings. Values may be any mix of JSONObjects, JSONArrays, Strings, Booleans, Integers, Longs, Doubles. Values may not be null.
- It is easy for machines to parse and generate ( for *more info* : [www.json.org](http://www.json.org))
- JSON object represented using { } braces. JSON arrays represented using [ ] braces.

# XML & JSON Format of Employee record

## XML Format

```
<employees>
  <employee>
    <id> 123 </id>
    <name> Renuka </name>
    <desig> AP </desig>
    <dept> CS </dept>
  </employee>
  <employee>
    <id> 125 </id>
    <name> Mohanraj </name>
    <desig> GD </desig>
    <dept> CS </dept>
  </employee>
</employees>
```

## JSON Format

```
{“employees”:[
  {
    “id” : 123,
    “name”: “Renuka”,
    “desig” : “AP”,
    “dept”:”CS”
  },
  {
    “id” : 125,
    “name”: “Mohanraj”,
    “desig” : “GD”,
    “dept”:”CS”
  }
]}
```

# XML vs JSON

- Both are used to transfer data from one technology to another technology.
- JSON occupies less space and load faster than XML.
- Information is represented as a collection of key/value pairs, and that each key/value pair is grouped into an ordered list of objects. JSON can provide data type like Integer, String.
  - “id” : 123
  - “name”: “Renuka”
- JSON parsing is simple. Converting object to JSON and JSON to object.
- Android has a build-in support for JSON parsing.
- Third party libraries are available in the market for parsing such as GSON, Retrofit, Jackson & Jettison etc.
- In this course we will use GSONS and Retrofit.

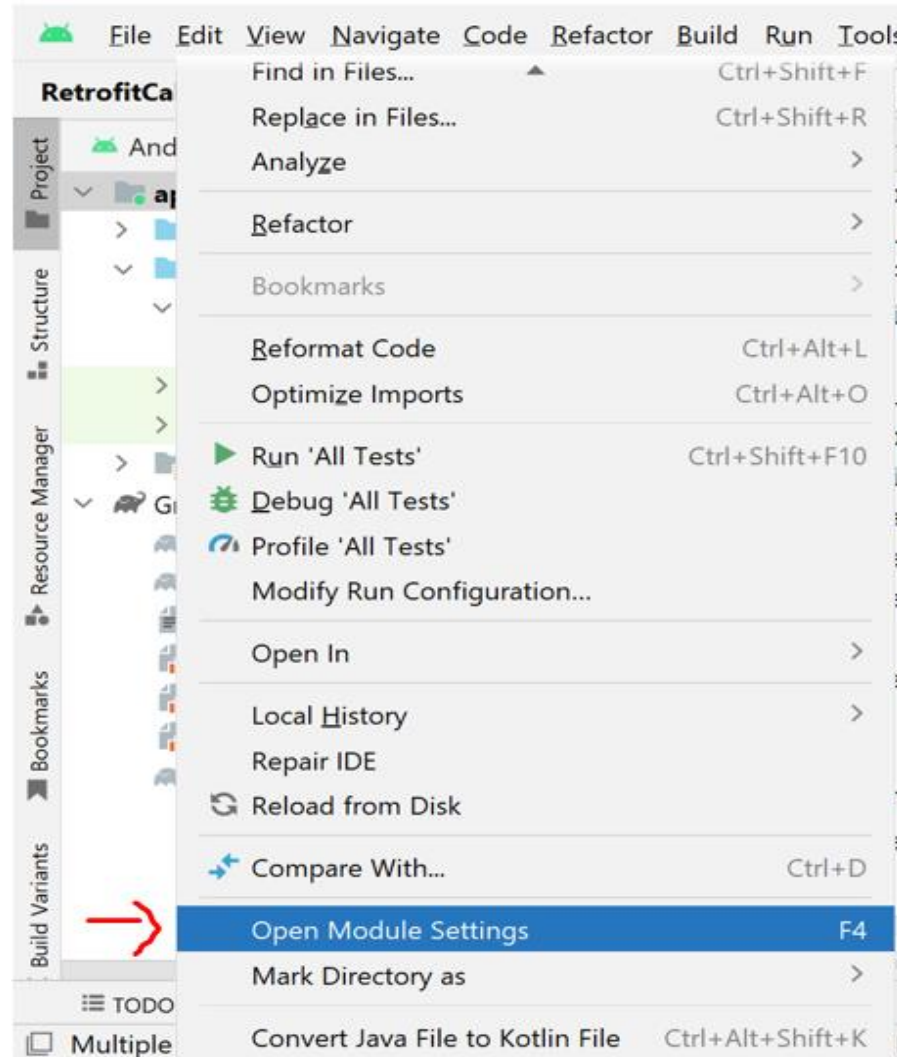
# Google Gson (Gson)

- Gson is a Java library that can be used to convert Java(POJO)/Kotlin(POKO) Objects into their JSON representation.
  - POJO – Plain Old Java Object, POKO – Plain Old Kotlin Object
- It can also be used to convert a JSON string to an equivalent Java/Kotlin object.
- It has extensive support for generics.
- To use Gson in Android add the below dependencies add directly in the gradle Module:app or follow the next few slides to add dependency.  
**implementation 'com.google.code.gson:gson:2.10.1'**
- Square's Retrofit, including its Gson-based converter code, for retrieving JSON data from Web services.



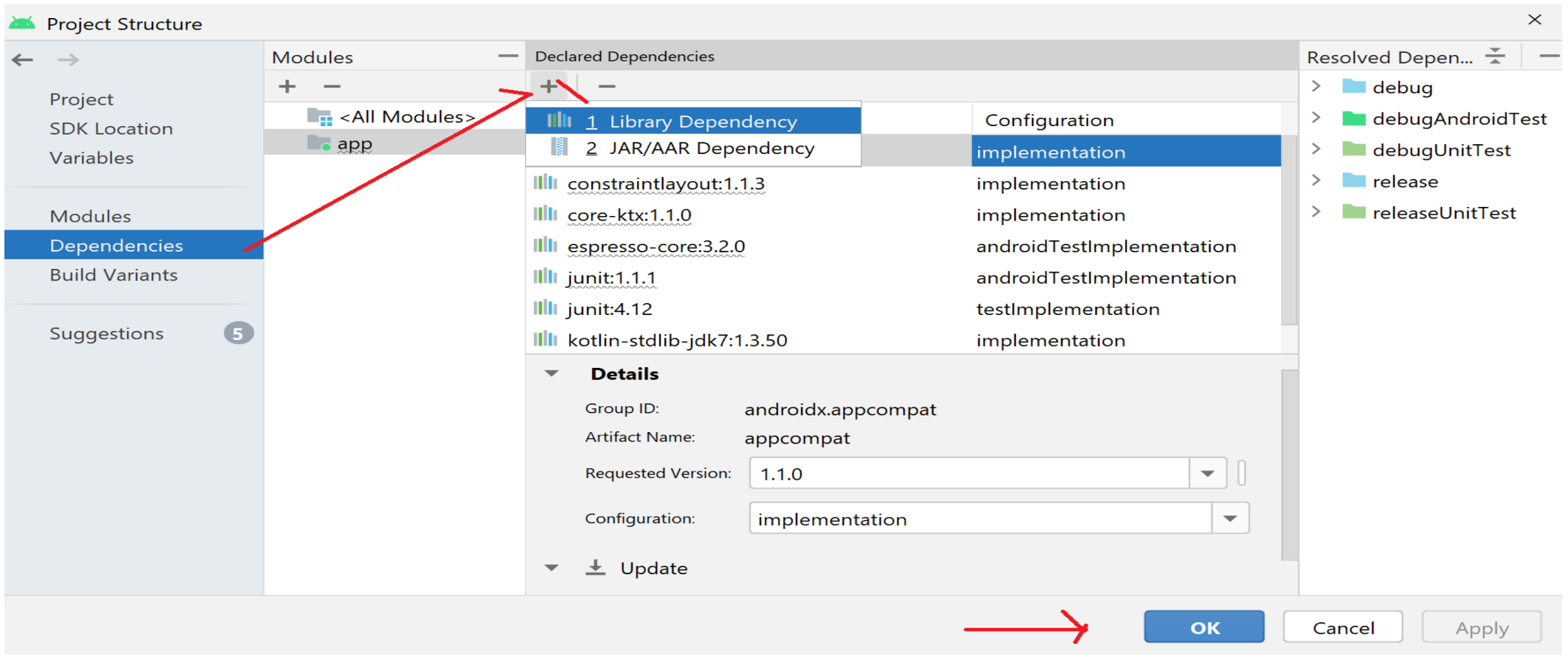
# Add Gson dependencies

Step 1 : Right click app-> Open Module Settings



# Add Gson dependencies

Step 2 : Choose the Dependencies Tab and click + icon to add dependency by selecting Library Dependencies



# Add Gson dependencies

Step 3 : Search gson dependency and select the highlighted dependency and click ok will automatically add **implementation 'com.google.code.gson:gson:2.10.1'** on Gradle.

**Add Library Dependency**

**Module 'app'**

**Step 1.**  
Use the form below to find the library to add. This form uses the repositories specified in the project's build files (Google, Maven Central)

gson Search

Enter a search query or fully-qualified coordinates (e.g. guava\* or com.google.\*:guava\* or com.google.guava:guava:26.0)

| Group ID                            | Artifact Name | Repository    | Versions |
|-------------------------------------|---------------|---------------|----------|
| com.google.code.gson                | gson          | Maven Central | 2.10.1   |
| at.stefangeyer.challonge.serializer | gson          | Maven Central |          |
| com.centurylink.mdw.assets          | gson          | Maven Central |          |
| com.episode6.typed2                 | gson          | Maven Central |          |

Library: com.google.code.gson:gson:2.10.1

**Step 2.**  
Assign your dependency to a configuration by selecting one of the configurations below.  
[Open Documentation](#)

implementation

**OK** **Cancel**

# JSON data into Kotlin data class

- Go to File → Settings → Plugins

The screenshot shows the IntelliJ IDEA Settings window with the 'Plugins' tab selected. The left sidebar lists various settings categories, with 'Plugins' highlighted. The main area displays the 'Marketplace' tab, where a search for 'json to kotlin' has been performed. The search results list several plugins, with 'JSON To Kotlin Class (JsonToKotlinClass)' at the top, marked as 'Installed'. Other plugins like 'RoboPOJOGenerator', 'Generate Kotlin data classes...', 'JSON To Dart Class (JsonTo...', and 'Pojo Generator' are listed below with 'Install' buttons. The right pane shows the details for the 'JSON To Kotlin Class (JsonToKotlinClass)' plugin, including its version (3.7.4), release date (Jun 05, 2022), and a description: 'Plugin for Kotlin to convert Json String into Kotlin data class code quickly'. It also lists features like generating Kotlin classes from JSON strings or responses.

Settings

Plugins

Marketplace Installed 1

Search json to kotlin

Search Results (26) Sort By: Relevance

**JSON To Kotlin Class (JsonToKotlinClass)** Installed

↓ 681.9K ☆ 4.79 Seal

Code Tools 3.7.4 Jun 05, 2022

[Plugin homepage](#)

Plugin for Kotlin to convert Json String into Kotlin data class code quickly

Fast use it with short cut key ALT + K on Windows or Option + K on Mac

**Features:**

Generating Kotlin class from any legal JSON string/JSONSchema or an API response

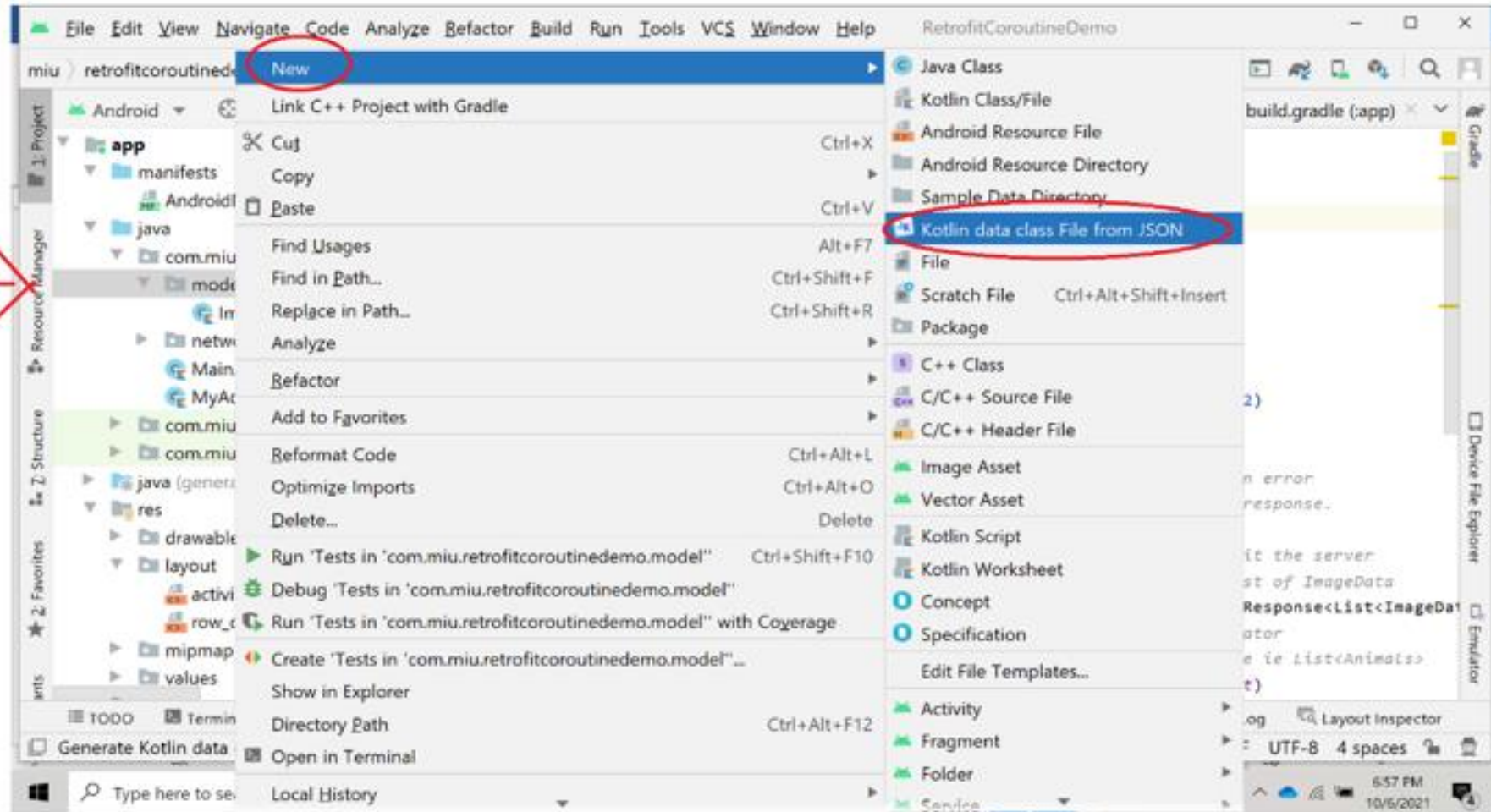
Generating Kotlin class from any legal JSON text when right click on selected JSON

OK Cancel Apply

# JSON data into Kotlin data class

Step 2: Follow as per the screenshot

Right click  
which package  
you need



# JSON data into Kotlin data class

**Step 3:** Paste your JSON Data and give your class name, then click Generate. Class will be created and stored in your package.

Generate Kotlin Data Class Code

Please input the JSON String and class name to generate Kotlin data class

JSON Text: Tips: you can use JSON string, http urls or local file just right click on text area Format

```
{
  "id": "28",
  "name": "Voyager A",
  "propellant": "Warp Drive",
  "destination": "Alpha Centauri A",
  "imageurl": "https://raw.githubusercontent.com/OclemysSampleJSON",
  "technologyexists": "1"
}
```

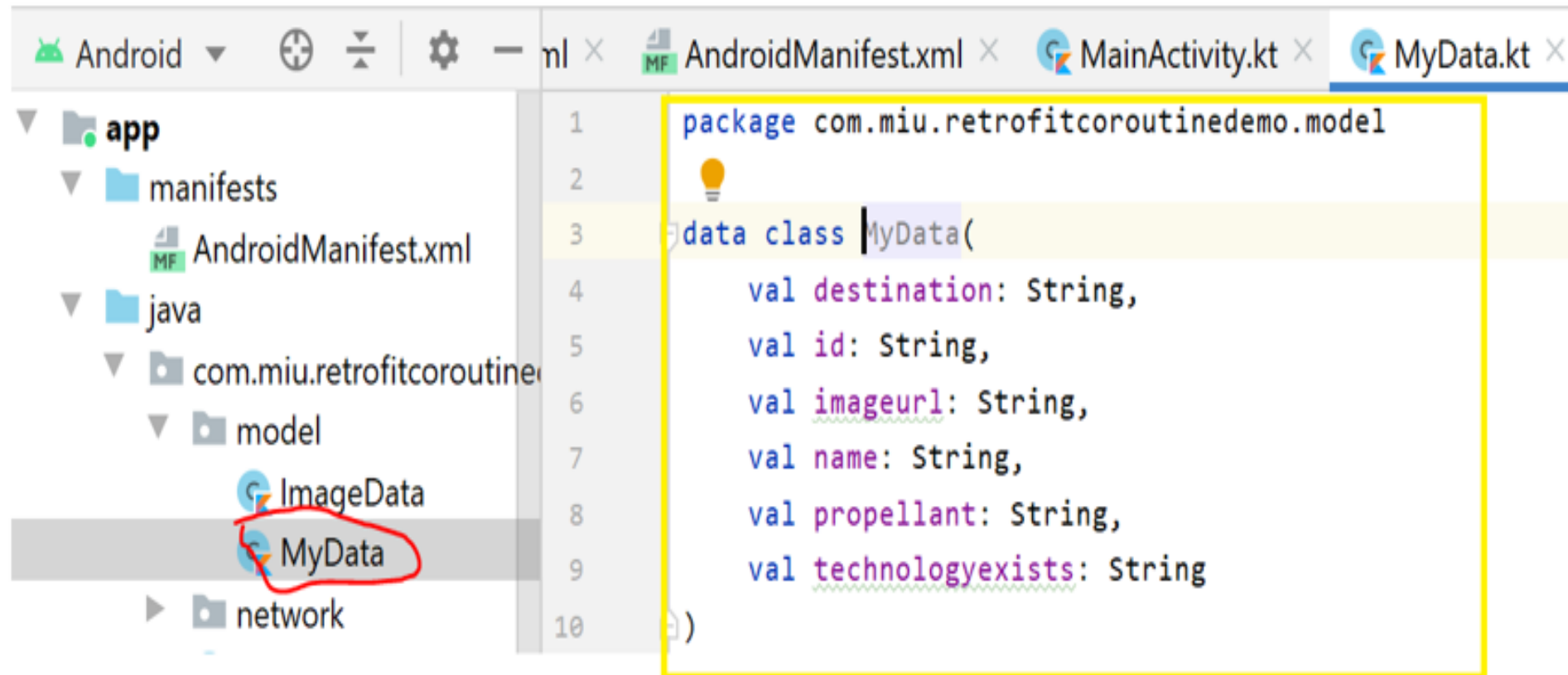
Class Name:

Advanced Like this version? Please star here: <https://github.com/wuseal/JsonToKotlinClass>

Generate Cancel

# JSON data into Kotlin data class

**Step 4:** You can see the created Class as mentioned below.





# Retrofit

- **Two Web Services**
  - **SOAP** stands for Simple Object Access Protocol.
  - **REST** stands for REpresentational State Transfer
- Most popular Webservice is Retrofit(REST). It's a REST Client for Java and Android. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST based webservice. Developed by Square(Third party Library).
- In **Retrofit** you should configure which converter is used for the data serialization.(like JSON, Gson)
- **Retrofit** automatically serializes and desterializes the JSON data as a type of POJO/POKO
- Best Example for WebService: Finding places using Google Maps
- Read more about from : <https://square.github.io/retrofit/>



# Hands on Example

- **Problem Requirement** : Read the JSON Data from this url:  
<https://raw.githubusercontent.com/Oclemey/SampleJSON/338d9585/spacecrafts.json> using Retrofit Library and show the data in the RecyclerView
- Need to add the following dependencies in build.gradle (Module:app)  
  
**// Retrofit and the Gson converter dependency**  
implementation 'com.squareup.retrofit2:retrofit:2.6.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.6.0'  
  
**// Glide support for Image downloading from the third party**  
implementation 'com.github.bumptech.glide:glide:4.8.0'  
  
**// Add permission in your Manifest**  
<uses-permission android:name="android.permission.INTERNET"></uses-permission>

# Steps to follow

1. Add Dependencies to the app-module build.gradle file:
2. Design an UI to retrieve network data probably ListView, GridView and RecyclerView
3. Create Data Class
4. Create the API interface which we will use to make requests and get responses via retrofit. [ Other requests – Post, Delete etc.,]
5. Create a Client call to get the response from the url using Retrofit object.

```
val api = Retrofit.Builder()
    .baseUrl(base_url)
    .addConverterFactory(GsonConverterFactory.create())
    .build()
    .create(APIInterface::class.java)
```

# Steps to follow

**// To get the methods automatically press CTRL + SHIFT +Space key, inside enqueue.**  
6. Then call enqueue() and override the following two methods to retrieve the data into your view object.

```
override fun onResponse(call: Call<List<POKO Type>?>?, response:
    Response<List<POJO Type>?>?) {
    if( response!!.isSuccessful){ // Check using non null !! operator
        /*The deserialized response body of a successful response and set into your adapter */
    }
}
call.enqueue(object : Callback<List< POKO Type >> {
    override fun onFailure(call: Call<List< POKO Type >?>?, t: Throwable?) {
        // The localized description of this throwable, like getErrorMessage from throwable
        Toast.makeText(applicationContext,"An Error Occured
        ${t?.localizedMessage}",Toast.LENGTH_LONG).show()
    }
})
```

# Complete Picture of the Implementation

Make a call through Retrofit

Getting response from the Network as JSON

POJO class of each entity in the JSON

```
← → ↻ raw.githubusercontent.com/Oclemly/SampleJSON/338d9585/spacecrafts.json

[
  {
    "id": "28",
    "name": "Voyager A",
    "propellant": "Warp Drive",
    "destination": "Alpha Centauri A",
    "imageUrl": "https://raw.githubusercontent.com/Oclemly/SampleJSON/master/spacecrafts/voyager.jpg",
    "technologyexists": "1"
  },
  {
    "id": "29",
    "name": "Voyager B",
    "propellant": "Plasma Ions",
    "destination": "Proxima Centauri",
    "imageUrl": "https://raw.githubusercontent.com/Oclemly/SampleJSON/master/spacecrafts/challenger.jpg",
    "technologyexists": "0"
  },
  {
    "id": "30",
    "name": "Apollo",
    "propellant": "Chemical Energy",
    "destination": "Venus",
    "imageUrl": "https://raw.githubusercontent.com/Oclemly/SampleJSON/master/spacecrafts/apollo.jpg",
    "technologyexists": "1"
  },
  {
    "id": "31",
    "name": "Pioneer",
    "propellant": "Laser Beam",
    "destination": "Saturn",
    "imageUrl": "https://raw.githubusercontent.com/Oclemly/SampleJSON/master/spacecrafts/pioneer.jpg",
    "technologyexists": "0"
  }
],
```

```
class ImageData(var id:Int,
                var name:String,
                var propellant:String,
                var destination:String,
                var imageUrl:String,
                technologyexists:Int)
```

Call to retrieve the ImageData from the Network

From MainActivity invoke the APIClient and make call.enqueue() to get onResponse()/onFailure from network using Retrofit

APIClient can make a Request to the Interface

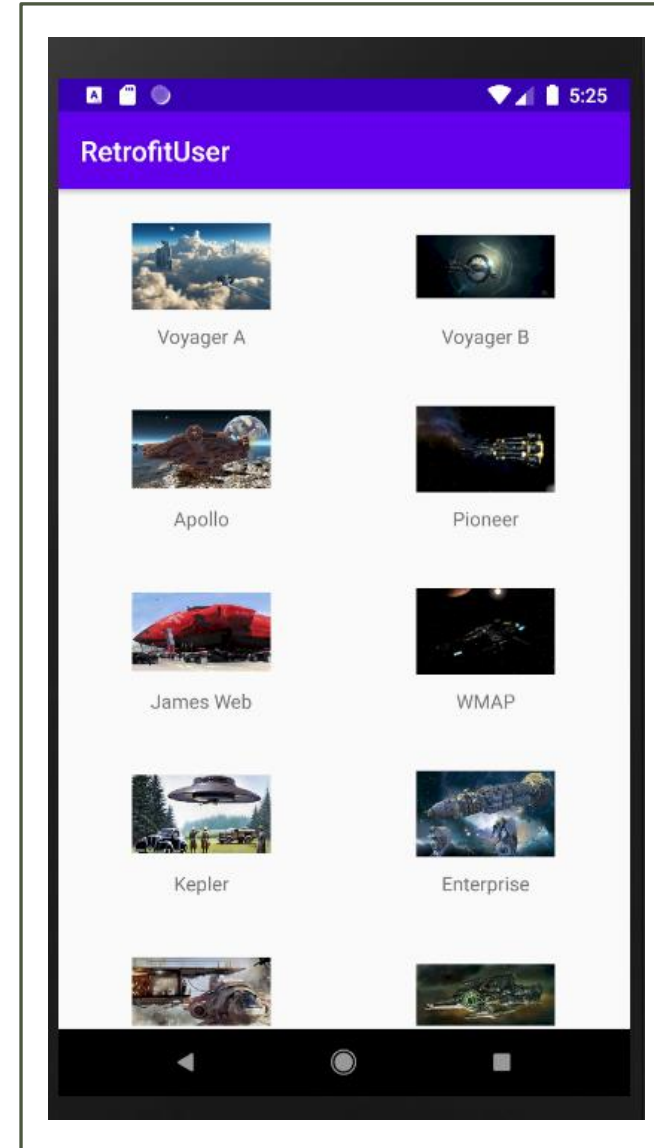
```
// Define the tasks you want to perform
interface APIInterface {
    @GET("/Oclemly/SampleJSON/338d9585/spacecrafts.json")
    fun getImages(): Call<List<ImageData>>
}
```

# Expected Outcome

- Read image data from the given URL JSON data and displayed in the GridView.
- **Refer Demo: TestRetrofit**

## Workflow

- Client Request → send through Retrofit → Network Server
- Network Server → Send responses in JSON Format → Response converted using GSON into object format → retrieve as POJO/POKO Type



# Retrofit using Coroutines

- Coroutines are a Kotlin feature that converts async callbacks for long-running tasks, such as database or network access.
- Kotlin coroutines let you convert callback-based code to sequential code and keeps Main safe.
- The previous example modified using Kotlin Coroutines.
- Add the below dependencies in the build.gradle module

**// Glide support for Image downloading from the third party**

**implementation 'com.github.bumptech.glide:glide:4.8.0'**

**// Coroutine Dependency**

**implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4'**

**// Retrofit and the Gson converter dependency**

**implementation 'com.squareup.retrofit2:retrofit:2.6.0'**

**implementation 'com.squareup.retrofit2:converter-gson:2.6.0'**

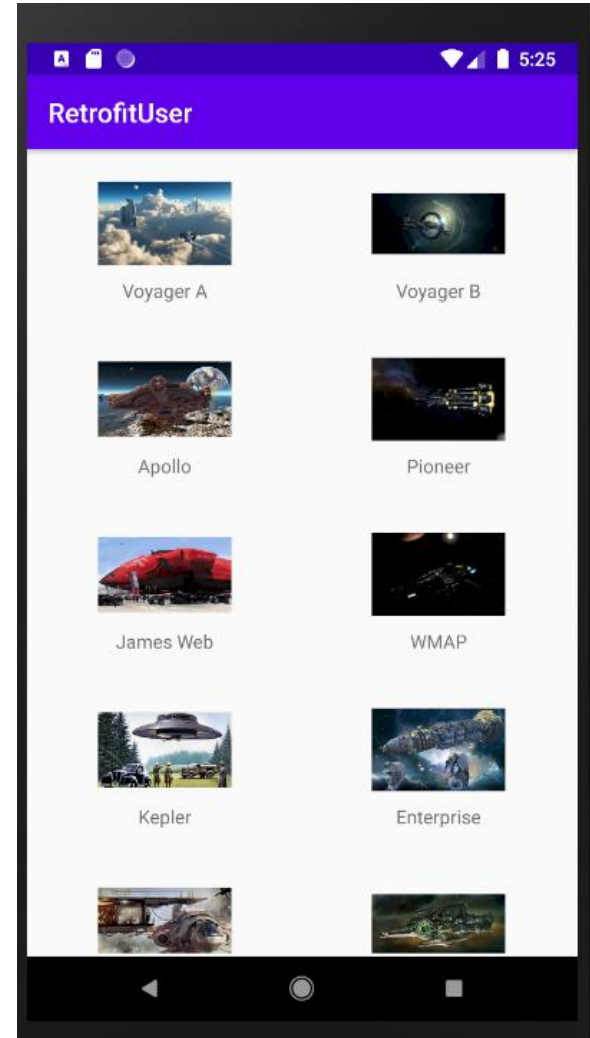
**// Lifecycle scope dependency**

**implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.6.1"**

# Modified Retrofit Example using Coroutines

```
// lifecycleScope.launch { // This is recommended to go
GlobalScope.launch(Dispatchers.Main) { // Example to show GlobalScope
    val response = api.getImages()
    if( response!!.isSuccessful){ // Check using non null !! operator
        response.body().let {
            if (it != null) {
                var items = it
                myAdapter.list = items
                myAdapter.notifyDataSetChanged()
            }
        }
    }
    else{
        Toast.makeText(applicationContext,"Failed to Retrieve ",
        Toast.LENGTH_LONG).show()
    }
}
```

**let** is a Kotlin Scope function, can make your code more concise and readable.  
**Refer : RetrofitCoroutine**



# Main Point 3

JSON has become more widely used for data representations than XML because JSON is easier to write and read and is almost identical to Java object literal syntax. Gson is a Java library to convert Kotlin Objects into their JSON representation. ***Science of Consciousness:** Enjoy greater efficiency and accomplish more by using JSON android APIs.*



# UNITY CHART

## CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Infinite creativity

1. WebView is one of the UI components in Android, which is used to display webpages in our application.
2. Android provides Shared Preferences object to help the developer to save simple application data as key/value pair.

- 
3. *Transcendental Consciousness: TC is the self-referral field of all possibilities.*
  4. *Impulses within the Transcendental field: These impulses are extremely subtle movements, transformations, within the ocean of Silence.*
  5. *Wholeness moving within Itself: In Unity Consciousness, all activity is appreciated as the self-referral dynamics of one's own Self.*

