# Creating a Pokemon Info App with Retrofit2 and Glide

This tutorial will guide you through creating an Android app that fetches Pokemon data using Retrofit2 and displays it with the help of Glide. We'll use an empty activity as the starting point and integrate the necessary libraries.

**Prerequisites:**

- Android Studio installed on your computer
- Basic understanding of Android development

**Steps:**

1. **Create a New Project:**
   - Open Android Studio and click "Start a new Android Studio project".
   - Choose "Empty (Views) Activity" as the template and give your project a name (e.g., PokemonInfoApp).
   - Click "Finish".
2. **Add Dependencies:**
   - Open your project's build.gradle file (located in the app module).
   - Inside the dependencies block, add the following libraries:

     ```
     Gradle
     implementation 'com.squareup.retrofit2:retrofit:2.9.0'
     implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
     implementation 'com.github.bumptech.glide:glide:4.14.2'
     ```

   - Click "Sync Now" to download the libraries.

The dependencies used in this project are libraries that help us with specific tasks. Retrofit2 handles network requests, allowing us to fetch data from the Pokemon API. GsonConverterFactory converts JSON data from the API into usable objects. Glide efficiently loads and displays images, like the Pokemon sprites. These dependencies save us time and effort by providing pre-built functionality for common tasks, making our app more efficient and easier to develop.

3. **Define Data Models:**
   - Create a new Kotlin file named PokemonData.kt under your app/java/yourpackagename directory (replace yourpackagename with your actual package name).
   - Paste the following code into the file:

```
data class PokemonData(val name: String,
                       val sprites: PokemonSprite,
                       val types: List<PokemonType>,
                       val abilities: List<PokemonAbility>)

data class PokemonSprite(val front_default: String)
```

```
data class PokemonType(val type: PokemonTypeDetails)

data class PokemonTypeDetails(val name: String)

data class PokemonAbility(val ability: PokemonAbilityDetail)

data class PokemonAbilityDetail(val name: String)
```

These classes represent the structure of the data we'll be fetching from the Pokemon API. This step involves creating Kotlin classes that mirror the structure of the JSON data we expect to receive from the Pokemon API. For instance, the `PokemonData` class has properties like `name`, `sprites`, `types`, and `abilities` to match the corresponding fields in the JSON response. When a network request is made using Retrofit2, the JSON response is parsed by GsonConverterFactory. This library maps the JSON data to the defined data classes, automatically populating the fields of each object. This parsed data can then be used to update the UI elements in the app, such as displaying the Pokemon's name, image, types, and abilities.

4. **Define Network Service Interface:**
   - Create another Kotlin file named PokemonService.kt in the same directory as before.
   - Paste the following code into the file:

   Kotlin
   ```
   import retrofit2.Call
   import retrofit2.http.GET
   import retrofit2.http.Path

   interface PokemonService {
       @GET("pokemon/{name}")
       fun getPokemon(@Path("name") name: String): Call<PokemonData>
   }
   ```
   This interface defines a method getPokemon that retrieves information for a specific Pokemon based on its name.

This step involves creating an interface that defines the network requests we want to make to the Pokemon API. This interface acts as a contract between our app and the API.
In the PokemonService interface, we define a single method getPokemon. This method is annotated with @GET and specifies the endpoint URL with a path parameter /{name}. This parameter will be replaced with the actual Pokemon name when we make the request.
By defining this interface, we're essentially telling Retrofit how to construct the network request: the HTTP method (GET), the URL, and the parameter. Retrofit will then generate the necessary code to make this request when we call the getPokemon method from our activity. This approach makes it easier to work with APIs and simplifies the process of making network requests in our app.

5. **Create the Layout:**
   - Open the activity_main.xml file located in the res/layout directory.
   - Replace the existing code with the following:

In the project, we're using a `ConstraintLayout` to position the elements on the screen. This

layout allows us to create complex layouts by defining relationships between different elements, such as aligning them to each other, centering them, or making them responsive to different screen sizes.

We've defined on the layout:

- **An ImageView** to display the Pokemon's image.
- **Several TextViews** to display the Pokemon's name, types, and abilities.
- **An EditText** for the user to input the Pokemon's name.
- **A Button** to trigger the search for the Pokemon.

By carefully arranging these elements and defining their constraints, we create a user-friendly and visually appealing interface for our Pokemon information app.

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="8dp"
        android:contentDescription="@string/pokemon_image_text"
        android:scaleType="fitXY"
        app:layout_constraintBottom_toTopOf="@+id/nameTextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button"
        tools:srcCompat="@tools:sample/avatars" />

    <TextView
        android:id="@+id/nameTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:text="@android:string/unknownName"
        android:textSize="34sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/typeTextView"
        app:layout_constraintEnd_toEndOf="@+id/imageView"
        app:layout_constraintHorizontal_bias="0.4"
        app:layout_constraintStart_toStartOf="@+id/imageView" />
```

```
    <TextView
        android:id="@+id/typeTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:text="@android:string/unknownName"
        app:layout_constraintBottom_toTopOf="@+id/abilitiesTextView"
        app:layout_constraintEnd_toEndOf="@+id/nameTextView"
        app:layout_constraintStart_toStartOf="@+id/nameTextView" />

    <TextView
        android:id="@+id/abilitiesTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:text="@android:string/unknownName"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/typeTextView"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="@+id/typeTextView" />

    <EditText
        android:id="@+id/searchEditText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:hint="@string/pokemon_name_text"
        android:importantForAutofill="no"
        android:inputType="textPersonName"
        android:minHeight="48dp"
        android:textColorHint="#78909C"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="@string/search_text"
        app:layout_constraintEnd_toEndOf="@+id/searchEditText"
        app:layout_constraintStart_toStartOf="@+id/searchEditText"
        app:layout_constraintTop_toBottomOf="@+id/searchEditText" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

6. **Implement the Activity:**
   - Open the MainActivity.kt file.
   - Replace the existing code with the following:

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.ImageView
import android.widget.TextView
import com.bumptech.glide.Glide
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class MainActivity : AppCompatActivity() {
   lateinit var searchEditText: EditText
   lateinit var button: Button
   lateinit var imageView: ImageView
   lateinit var nameTextView: TextView
   lateinit var typeTextView: TextView
   lateinit var abilitiesTextView: TextView
   lateinit var pokemonData: PokemonData
   val pokemonService = Retrofit.Builder()
           .baseUrl("https://pokeapi.co/api/v2/")
       .addConverterFactory(
           GsonConverterFactory.create()
       ).build()
       .create(PokemonService::class.java)

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
       imageView = findViewById(R.id.imageView)
       nameTextView = findViewById(R.id.nameTextView)
       typeTextView = findViewById(R.id.typeTextView)
       abilitiesTextView = findViewById(R.id.abilitiesTextView)
       searchEditText = findViewById(R.id.searchEditText)
       button = findViewById(R.id.button)
       button.setOnClickListener {
           val searchText = searchEditText.text.toString()
           fetchPokemon(searchText)
       }
   }
   fun fetchPokemon(name: String){
      pokemonService.getPokemon(name)
          .enqueue(object: Callback<PokemonData>{
              override fun onResponse(call: Call<PokemonData>, response:
Response<PokemonData>) {
                  if (response.isSuccessful){
                      val pokemon = response.body()!!
                      Glide.with(this@MainActivity)
                          .load(pokemon.sprites.front_default)
                          .into(imageView)
                      nameTextView.text= pokemon.name
```

```
                    typeTextView.text = pokemon.types.joinToString(","){
it.type.name}
                    abilitiesTextView.text =
pokemon.abilities.joinToString(","){it.ability.name}
                }
            }
            override fun onFailure(call: Call<PokemonData>, t: Throwable) {
                Log.e("Pokemon",t.localizedMessage)
            }

        })
    }
}
```

The `MainActivity` class is the core of the app. It's responsible for:

1. **Initializing UI Elements:** It finds references to the UI elements like the `EditText`, `Button`, `ImageView`, and `TextViews` defined in the layout XML.
2. **Setting Up Button Click Listener:** It listens for clicks on the button and, when clicked, triggers the `fetchPokemon` method.
3. **Fetching Pokemon Data:** The `fetchPokemon` method:
    - Uses Retrofit to make a network request to the Pokemon API with the provided Pokemon name.
    - Asynchronously handles the response using the `enqueue` method.
    - If the request is successful:
        - Extracts the Pokemon data from the response.
        - Loads the Pokemon's image using Glide.
        - Updates the text views with the Pokemon's name, types, and abilities.
    - If the request fails:
        - Logs an error message and potentially displays an error message to the user.

By combining these steps, the `MainActivity` provides a seamless user experience where users can input a Pokemon name, fetch its information, and view it on the screen.