

Mobile Application Development

Produced
by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
<http://www.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



Android & Retrofit

A type-safe HTTP client for Android & Java





Retrofit

A type-safe HTTP client for Android and Java

Introduction

Retrofit turns your HTTP API into a Java interface.

```
public interface GitHubService {  
    @GET("users/{user}/repos")  
    Call<List<Repo>> listRepos(@Path("user") String user);  
}
```

The `Retrofit` class generates an implementation of the `GitHubService` interface.

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();  
  
GitHubService service = retrofit.create(GitHubService.class);
```

Each `Call` from the created `GitHubService` can make a synchronous or asynchronous HTTP request to the remote webserver.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

[Introduction](#)

[API Declaration](#)

[Retrofit Configuration](#)

[Download](#)

[Contributing](#)

[License](#)

[Javadoc](#)

[StackOverflow](#)



Agenda & Goals

- ❑ Investigate the use of Retrofit in App Development
- ❑ Be aware of the different Retrofit Annotations and Classes and how, when and where to use them
- ❑ Revisit Java interfaces
- ❑ Understand how to integrate Retrofit into an Android App
- ❑ Refactor our **CoffeeMate** Case Study



What is it?

- ❑ **Retrofit** is a Java Library that turns your REST API into a Java interface
- ❑ Simplifies HTTP communication by turning remote APIs into declarative, type-safe interfaces
- ❑ Developed by Square (Jake Wharton)
- ❑ Retrofit is one of the most popular HTTP Client Library for Android at the moment as a result of its simplicity and its great performance compared to the others (next slide)
- ❑ Retrofit makes use of **OkHttp** (from the same developer) to handle network requests.



Retrofit Performance Analysis

	One Discussion	Dashboard (7 requests)	25 Discussions
AsyncTask	941 ms	4,539 ms	13,957 ms
Volley	560 ms	2,202 ms	4,275 ms
Retrofit	312 ms	889 ms	1,059 ms

<http://instructure.github.io/blog/2013/12/09/volley-vs-retrofit/>



Retrofit Vs Volley

- ❑ **Retrofit** is more about code abstraction on top of a HTTP client which for Retrofit is **OkHttp**.
- ❑ **Retrofit** aims to make it easier to consume RESTful web services where as the goal of **Volley** is to handle all your networking needs for Android specifically.



Why Use it?

- ❑ Developing your own type-safe HTTP library to interface with a REST API can be a burden : you have to handle many functionalities such as making connections, caching, retrying failed requests, threading, response parsing, error handling, and more.
- ❑ **Retrofit**, on the other hand, is very well planned, documented, and tested—a battle-tested library that will save you a lot of precious time and headaches.

The Basics

- ❑ Again, Retrofit2 is a flexible library that uses annotated interfaces to create REST calls. To get started, let's look at our **CoffeeMate** example that makes a **GET** request for Coffees.
- ❑ Here's the **Coffee** class we're using:
- ❑ Much simpler if field names matches server model (but doesn't have to, see later)

```
public class Coffee
{
    public String _id;
    public String name;
    public String shop;
    public double rating;
    public double price;
    public boolean favourite;

    public Coffee() {}

    public Coffee(String name, String shop,
                 double rating, double price,
                 boolean fav) {
        this.name = name;
        this.shop = shop;
        this.rating = rating;
        this.price = price;
        this.favourite = fav;
    }

    @Override
    public String toString() {
        return _id + " " + name + ", " + shop + ", "
               + rating + ", " + price
               + ", fav =" + favourite;
    }
}
```



The Wrapper class *

- Once we've defined the class, we may (we do here) need a 'wrapper' class to represent our JSON Response

```
public class CoffeeWrapper {  
    public int status;  
    public String message;  
    public List<Coffee> data;  
}
```



- Note that the `data` field is a JSON array, this is important for parsing



The Service interface *

- Once we've defined the class (and possible wrapper), we can make a service interface to handle our API. **GET** requests to load all Coffees and a single Coffee could look something like this:

```
public interface CoffeeService
{
    @GET("/coffees")
    Call<CoffeeWrapper> getAll();

    @GET("/coffees/{id}")
    Call<CoffeeWrapper> get(@Path("id") String id);
}
```

- Note that the **@GET** annotation takes the endpoint we wish to hit. As you can see, an implementation of this interface will return a **Call** object containing a **CoffeeWrapper**.



Call

- Models a single request/response pair
- Separates request creation from response handling
- Each instance can only be used once...
- ...instances can be cloned
- Supports both synchronous and asynchronous execution.
- Can be (actually) canceled



Beyond GET – other types of Calls

- ❑ Retrofit2 is not limited to GET requests. You may specify other REST methods using the appropriate annotations (such as @POST, @PUT and @DELETE).
- ❑ Here's another version of our CoffeeService interface (with full CRUD support)

```
public interface CoffeeService
{
    @GET("/coffees")
    Call<CoffeeWrapper> getAll();

    @GET("/coffees/{id}")
    Call<CoffeeWrapper> get(@Path("id") String id);

    @DELETE("/coffees/{id}")
    Call<CoffeeWrapper> delete(@Path("id") String id);

    @POST("/coffees")
    Call<CoffeeWrapper> post(@Body Coffee coffee);

    @PUT("/coffees/{id}")
    Call<CoffeeWrapper> put(@Path("id") String id,
                           @Body Coffee coffee);
}
```



Calling the API

- ❑ So how do we use this interface to make requests to the API?
- ❑ Use **Retrofit2** to create an implementation of the above interface, and then call the desired method.
- ❑ Retrofit2 supports a number of converters used to map Java objects to the data format your server expects (JSON, XML, etc). we'll be using the **Gson** converter.

- **Gson**: com.squareup.retrofit2:converter-gson
- **Jackson**: com.squareup.retrofit2:converter-jackson
- **Moshi**: com.squareup.retrofit2:converter-moshi
- **Protobuf**: com.squareup.retrofit2:converter-protobuf
- **Wire**: com.squareup.retrofit2:converter-wire
- **Simple XML**: com.squareup.retrofit2:converter-simplexml
- **Scalars (primitives, boxed, and String)**: com.squareup.retrofit2:converter-scalars



Aside - CoffeeMate Android Client

☐ coffeemate-nodeserver.herokuapp.com api endpoints

```
{ method: 'GET', path: '/coffees', config: CoffeeApi.findAll },
{ method: 'GET', path: '/coffees/{id}', config: CoffeeApi.findById },
{ method: 'POST', path: '/coffees', config: CoffeeApi.addCoffee },
{ method: 'PUT', path: '/coffees/{id}', config: CoffeeApi.updateCoffee },
{ method: 'DELETE', path: /coffees/{id}', config: CoffeeApi.deleteCoffee }
```

☐ Use CoffeeService for

- ☐ Adding / Updating / Deleting a Coffee
- ☐ Listing All Coffees
- ☐ Finding a single Coffee

```
{
  "status": 9,
  "message": "Coffee Successfully Retrieved!",
  "data": [
    {
      "favourite": false,
      "_id": "5bf3aa9fb6fc0561ffcaadd",
      "name": "Standard Black lite",
      "shop": "Tescos",
      "price": 1.99,
      "rating": 2.5
    }
  ]
}
```



Android Networking (Using Retrofit) in CoffeeMate.6b.0



Steps to integrate Retrofit into your App

1. Set up your Project Dependencies & Permissions
2. Create Interface for API and declare methods for each REST Call, specifying method type using Annotations -
 @GET, @POST, @PUT, etc. For parameters use - @Path,
 @Query, @Body
3. Use **Retrofit** to build the service client
4. Make the REST Calls as necessary using the relevant Callback mechanism



1. Project Dependencies & Permissions *

- ❑ Add the required dependencies to your build.gradle

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:28.0.0'  
    implementation 'com.android.support:support-v4:28.0.0'  
    implementation 'com.android.support:design:28.0.0'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'com.android.support.test:runner:1.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
  
    implementation 'com.makeramen:roundedimageview:2.2.1'  
    implementation "com.android.support:recyclerview-v7:28.0.0"  
    implementation "com.android.support:cardview-v7:28.0.0"  
    implementation 'com.android.volley:volley:1.1.1'  
    implementation 'com.google.code.gson:gson:2.8.5' // for Googles Gson JSON Parser  
  
    implementation 'com.squareup.retrofit2:retrofit:2.3.0'  
    implementation 'com.squareup.retrofit2:converter-gson:2.3.0'  
}
```

- ❑ And the necessary permissions to your manifest – BEFORE/OUTSIDE the application tag

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



2. Create interface (and Wrapper) for API

The screenshot shows the Android Studio project structure for 'CoffeeMate.6b.0'. The 'app' module is selected. Inside 'app', the 'src' folder contains 'java' and 'res' subfolders. The 'java' folder has a package named 'ie.cm' which contains 'activities', 'adapters', and 'api'. The 'api' package is expanded, showing two files: 'CoffeeService.java' (marked with an 'I') and 'CoffeeWrapper.java' (marked with a 'C'). Below 'api' are 'fragments', 'main', and 'models' packages, each containing a 'Coffee.java' file (marked with a 'C'). The 'models' package is also expanded. At the bottom of the tree, there are 'ie.cm (androidTest)', 'ie.cm (test)', 'generatedJava', and 'Gradle Scripts'.

```
public interface CoffeeService {
    @GET("/coffees")
    Call<CoffeeWrapper> getAll();

    @GET("/coffees/{id}")
    Call<CoffeeWrapper> get(@Path("id") String id);

    @DELETE("/coffees/{id}")
    Call<CoffeeWrapper> delete(@Path("id") String id);

    @POST("/coffees")
    Call<CoffeeWrapper> post(@Body Coffee coffee);

    @PUT("/coffees/{id}")
    Call<CoffeeWrapper> put(@Path("id") String id,
                           @Body Coffee coffee);
}
```

```
public class CoffeeWrapper {
    public int status;
    public String message;
    public List<Coffee> data;
}
```



3. Build Service Client - CoffeeMateApp *

- ☐ Implement the necessary interface & variables

Our
CoffeeService
instance

```
public class CoffeeMateApp extends Application
{
    //...
    public CoffeeService coffeeService;
    //...
    public String serviceURL = "http://coffeemate-nodeserver.herokuapp.com";
}
```

Note the **serviceURL**



3. Build Service Client - CoffeeMateApp *

- Create the proxy service '**coffeeService**', with the appropriate Gson parsers

```
@Override  
public void onCreate() {  
    super.onCreate();  
  
    //...  
    Gson gson = new GsonBuilder().create();  
  
    OkHttpClient okHttpClient = new OkHttpClient.Builder()  
        .connectTimeout( timeout: 30, TimeUnit.SECONDS )  
        .writeTimeout( timeout: 30, TimeUnit.SECONDS )  
        .readTimeout( timeout: 30, TimeUnit.SECONDS )  
        .build();  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(serviceURL)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .client(okHttpClient)  
        .build();  
  
    coffeeService = retrofit.create(CoffeeService.class);  
  
    Log.v( tag: "coffeemate", msg: "Coffee Service Created" );  
}
```



3. Build Service Client - CoffeeMateApp *

Gson for converting our JSON

OkHttpClient for communication timeouts (optional)

Retrofit.Builder to create an instance of our interface

```
@Override  
public void onCreate() {  
    super.onCreate();  
  
    //...  
    Gson gson = new GsonBuilder().create();  
  
    OkHttpClient okHttpClient = new OkHttpClient.Builder()  
        .connectTimeout( timeout: 30, TimeUnit.SECONDS )  
        .writeTimeout( timeout: 30, TimeUnit.SECONDS )  
        .readTimeout( timeout: 30, TimeUnit.SECONDS )  
        .build();  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(serviceURL)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .client(okHttpClient)  
        .build();  
  
    coffeeService = retrofit.create(CoffeeService.class);  
  
    Log.v( tag: "coffeemate", msg: "Coffee Service Created");  
}
```



4. Calling the API - CoffeeFragment *

- ☐ Implement the necessary interface & variables

```
public class CoffeeFragment extends Fragment implements View.OnClickListener,  
    AdapterView.OnItemClickListener, AbsListView.MultiChoiceModeListener,  
    Callback<CoffeeWrapper>  
{  
    //...  
    public Call<CoffeeWrapper> callRetrieve, callDelete;
```

Note the
Callback
interface

the **call** objects, for
making the requests



4. CoffeeFragment – onResume() *

```
public void getAllCoffees() {  
    activity.showLoader("Downloading Coffees...");  
    callRetrieve = app.coffeeService.getAll();  
    callRetrieve.enqueue(this);  
}
```

```
public void onResume() {  
    super.onResume();  
    getAllCoffees();  
    updateView();  
}
```

```
public void updateView() {  
    listAdapter = new CoffeeListAdapter(activity, this, app.coffeeList);  
    coffeeFilter = new CoffeeFilter(app.coffeeList, "all", listAdapter);  
  
    if (favourites) {  
        coffeeFilter.setFilter("favourites"); // Set the filter text file  
        coffeeFilter.filter(null); // Filter the data, but don't use any  
        // listAdapter.notifyDataSetChanged(); // Update the adapter  
    }  
    setListView(v);  
    setSwipeRefresh(v);  
  
    if (!favourites)  
        getActivity().setTitle("Recently Added Coffee's");  
    else  
        getActivity().setTitle("Favourite Coffee's");  
  
    listAdapter.notifyDataSetChanged(); // Update the adapter  
}
```

enqueue() allows for asynchronous callback to our service



4. CoffeeFragment – onResponse() *

- ❑ Triggered on a successful call to the API
- ❑ Takes 2 parameters
 - ❑ The **Call** object
 - ❑ The expected **Response** object
- ❑ Converted JSON result stored in **response.body()**

```
@Override  
public void onResponse(Call<CoffeeWrapper> call, Response<CoffeeWrapper> response) {  
    CoffeeWrapper cw = response.body();  
    app.coffeeList = cw.data;  
    updateView();  
    checkSwipeRefresh(v);  
    activity.hideLoader();  
}
```



4. CoffeeFragment – onFailure() *

- ❑ Triggered on an unsuccessful call to the API
- ❑ Takes 2 parameters
 - ❑ The **Call** object
 - ❑ A **Throwable** object containing error info
- ❑ Probably should inform user of what's happened

```
@Override  
public void onFailure(Call<CoffeeWrapper> call, Throwable t) {  
    Log.v("coffeemate", "Something Went Wrong : " + t.getMessage());  
    t.printStackTrace();  
    Toast.makeText(getActivity(),  
        "Coffee Service Unavailable. Try again later",  
        Toast.LENGTH_LONG).show();  
}
```



'Add' UseCase *

```
public class AddFragment extends Fragment implements Callback<CoffeeWrapper> {  
  
    private String coffeeName, coffeeShop;  
    private double coffeePrice, ratingValue;  
    private EditText name, shop, price;  
    private RatingBar ratingBar;  
    private Button saveButton;  
    private CoffeeMateApp app;  
    public Call<CoffeeWrapper> callCreate;  
  
    public AddFragment() {}  
  
    public static AddFragment newInstance() {...}  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {...}  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {...}
```

Callback and Call references



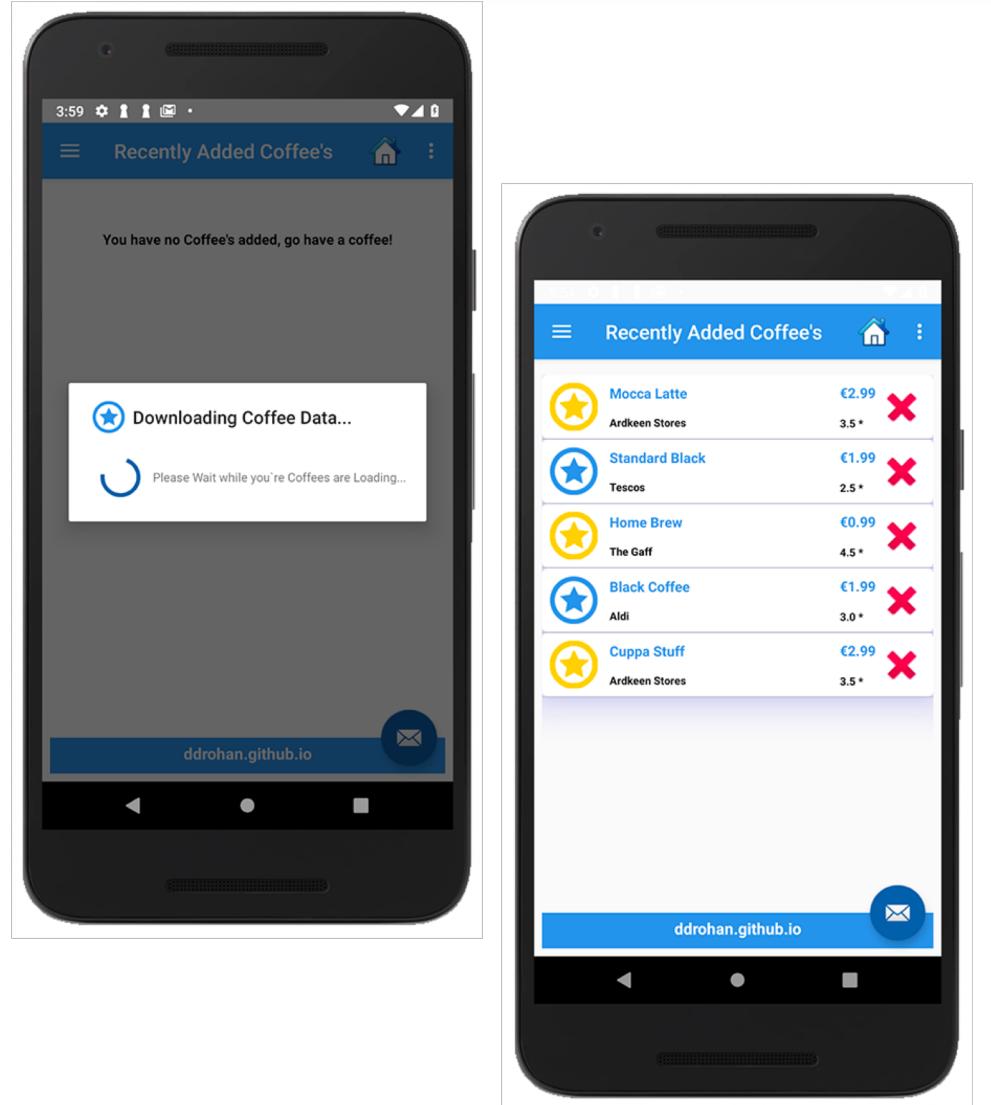
'Add' UseCase *

```
public void addCoffee() {  
    coffeeName = name.getText().toString();  
    coffeeShop = shop.getText().toString();  
    try {  
        coffeePrice = Double.parseDouble(price.getText().toString());  
    } catch (NumberFormatException e) {  
        coffeePrice = 0.0;  
    }  
    ratingValue = ratingBar.getRating();  
  
    if ((coffeeName.length() > 0) && (coffeeShop.length() > 0)  
        && (price.length() > 0)) {  
        Coffee c = new Coffee(coffeeName, coffeeShop, ratingValue,  
            coffeePrice, false);  
  
        callCreate = app.coffeeService.post(c);  
        callCreate.enqueue(this);  
    } else  
        Toast.makeText(this.getActivity(),  
            "You must Enter Something for "  
            + "'Name'", "'Shop'" and "'Price'",  
            Toast.LENGTH_SHORT).show();  
}
```

Creating the **Call** and triggering the **Callback**



coffeemate-nodeserver + mobile app



coffeemate-nodeserver.herokuapp.com

CoffeeMate Node Server

Our Custom Coffee Web App Routes

- GET – app.get('/coffees', coffees.findAll)
- GET – app.get('/coffees/:id', coffees.findOne)
- POST – app.post('/coffees', coffees.addCoffee)
- PUT – app.put('/coffees/:id', coffees.editCoffee)
- PUT – app.put('/coffees/:id/fav', coffees.setFavourite)
- DELETE – app.delete('/coffees/:id', coffees.deleteCoffee)

Anonymous Callbacks *

Anonymous Callback
allows for multiple calls
in same class



```
callUpdate = app.coffeeService.put(aCoffee._id,aCoffee);
callUpdate.enqueue(new Callback<CoffeeWrapper>() {
    @Override
    public void onResponse(Call<CoffeeWrapper> call, Response<CoffeeWrapper> response) {
        Toast.makeText(getActivity(), "Coffee Updated...", Toast.LENGTH_LONG).show();
        if (getActivity().getSupportFragmentManager().getBackStackEntryCount() > 0) {
            getActivity().getSupportFragmentManager().popBackStack();
        }
    }
    @Override
    public void onFailure(Call<CoffeeWrapper> call, Throwable t) {
        Toast.makeText(getActivity(), "Unable to Update Coffee. Try again later",
                    Toast.LENGTH_LONG).show();
    }
});
```



'Delete' UseCase *

```
public void onCoffeeDelete(final Coffee coffee) {
    String stringName = coffee.name;
    AlertDialog.Builder builder = new AlertDialog.Builder(activity);
    builder.setMessage("Are you sure you want to Delete the 'Coffee' " + stringName + "?");
    builder.setCancelable(false);

    builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id)
        {
            deleteCoffee(coffee._id);
            getAllCoffees();
            listAdapter.notifyDataSetChanged(); // refresh adapter
        }
    }).setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) { dialog.cancel(); }
    });
    AlertDialog alert = builder.create();
    alert.show();
}
```

Anonymous Callbacks
multiple calls here



'Delete' UseCase *

```
public void deleteCoffee(String id) {  
    callDelete = app.coffeeService.delete(id);  
    callDelete.enqueue(new Callback<CoffeeWrapper>() {  
        @Override  
        public void onResponse(Call<CoffeeWrapper> call, Response<CoffeeWrapper> response) {  
            Toast.makeText(getActivity(), "Successfully Deleted Coffee" ,  
                Toast.LENGTH_LONG).show();  
        }  
  
        @Override  
        public void onFailure(Call<CoffeeWrapper> call, Throwable t) {  
            Toast.makeText(getActivity(), "Unable to Delete Coffee : " + "ERROR: " +  
                t.getMessage(), Toast.LENGTH_LONG).show();  
        }  
    });  
}
```

Anonymous Callback

```
public void getAllCoffees() {  
    activity.showLoader("Downloading Coffees...");  
    callRetrieve = app.coffeeService.getAll();  
    callRetrieve.enqueue(this);  
}
```

Standard Callback



Bridging the Gap Between Your Code & Your API

❑ Variable Names

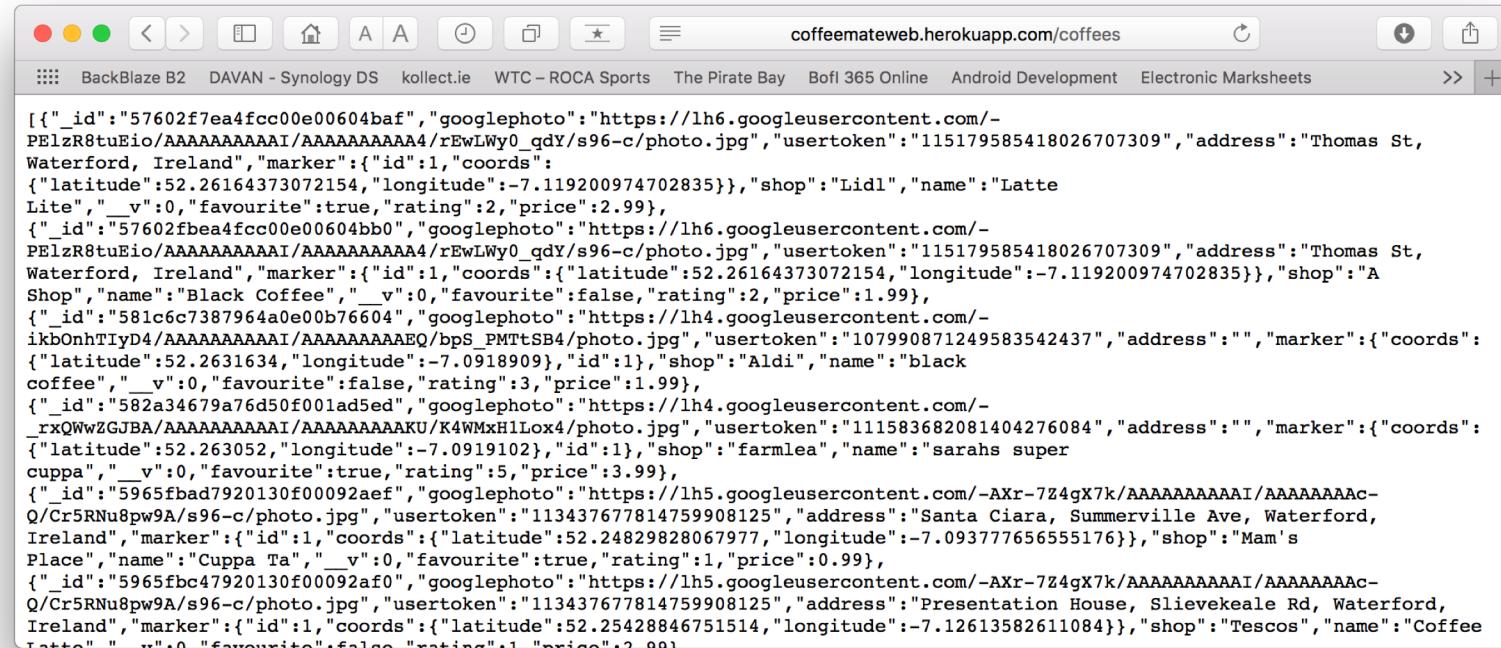
- ❑ In the previous examples, we assumed that there was an exact mapping of instance variable names between the Coffee class and the server. This will often not be the case, especially if your server uses a different spelling convention than your Android app.
- ❑ For example, if you use a Rails server, you will probably be returning data using `snake_case`, while your Java probably uses `camelCase`. If we add a `dateCreated` to the Coffee class, we may be receiving it from the server as `date_created`.
- ❑ To create this mapping, use the `@SerializedName` annotation on the instance variable. For example:

```
@SerializedName("date_created")
private Date dateCreated;
```



Bridging the Gap Between Your Code & Your API

- You can also create your models automatically from your JSON response data by leveraging a very useful tool:
jsonschema2pojo - <http://www.jsonschema2pojo.org>
- Grab your JSON string, visit the above link and paste it in, like so



```
[{"_id": "57602f7ea4fcc00e00604baf", "googlephoto": "https://lh6.googleusercontent.com/-PElzRtuEio/AAAAAAAAAAI/AAAAAAA4A/rEwLWy0_qdY/s96-c/photo.jpg", "usertoken": "115179585418026707309", "address": "Thomas St, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.26164373072154, "longitude": -7.119200974702835}}, "shop": "Lidl", "name": "Latte Lite", "v": 0, "favourite": true, "rating": 2, "price": 2.99}, {"_id": "57602fbea4fcc00e00604bb0", "googlephoto": "https://lh6.googleusercontent.com/-PElzRtuEio/AAAAAAAAAAI/AAAAAAA4A/rEwLWy0_qdY/s96-c/photo.jpg", "usertoken": "115179585418026707309", "address": "Thomas St, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.26164373072154, "longitude": -7.119200974702835}}, "shop": "A Shop", "name": "Black Coffee", "v": 0, "favourite": false, "rating": 2, "price": 1.99}, {"_id": "581c6c7387964a0e00b76604", "googlephoto": "https://lh4.googleusercontent.com/-ikbOnhTIyD4/AAAAAAAAAAI/AAAAAAAEGQ/bpS_PMTtSB4/photo.jpg", "usertoken": "107990871249583542437", "address": "", "marker": {"coords": {"latitude": 52.2631634, "longitude": -7.0918909}, "id": 1}, "shop": "Aldi", "name": "black coffee", "v": 0, "favourite": false, "rating": 3, "price": 1.99}, {"_id": "582a34679a76d50f001ad5ed", "googlephoto": "https://lh4.googleusercontent.com/-rxQWwZGJBA/AAAAAAAAAAI/AAAAAAAAKU/K4WMxH1Lox4/photo.jpg", "usertoken": "11583682081404276084", "address": "", "marker": {"coords": {"latitude": 52.263052, "longitude": -7.0919102}, "id": 1}, "shop": "farmlea", "name": "sarabs super cuppa", "v": 0, "favourite": true, "rating": 5, "price": 3.99}, {"_id": "5965fbad7920130f00092aef", "googlephoto": "https://lh5.googleusercontent.com/-AXr-7Z4gX7k/AAAAAAAAAAI/AAAAAAAAC-Q/Cr5RNu8pw9A/s96-c/photo.jpg", "usertoken": "113437677814759908125", "address": "Santa Ciara, Summerville Ave, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.24829828067977, "longitude": -7.093777656555176}}, "shop": "Mam's Place", "name": "Cuppa Ta", "v": 0, "favourite": true, "rating": 1, "price": 0.99}, {"_id": "5965fbc47920130f00092af0", "googlephoto": "https://lh5.googleusercontent.com/-AXr-7Z4gX7k/AAAAAAAAAAI/AAAAAAAAC-Q/Cr5RNu8pw9A/s96-c/photo.jpg", "usertoken": "113437677814759908125", "address": "Presentation House, Slievekeale Rd, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.25428846751514, "longitude": -7.12613582611084}}, "shop": "Tescos", "name": "Coffee Tattoo", "v": 0, "favourite": false, "rating": 1, "price": 2.99}
```

Bridging the Gap Between Your Code & Your API

Your JSON goes here

The screenshot shows the jsonschema2pojo website interface. On the left, there is a large text input area labeled "Your JSON goes here" with a black arrow pointing towards it from the left side of the slide. The main content area displays a JSON snippet:

```
[{"_id": "57602f7ea4fcc00e00604baf",  
 "googlephoto": "https://lh6.googleusercontent.com/-PEIzR8tu  
 "usertoken": "115179585418026707309"  
 "address": "Thomas St, Waterford, Ireland",  
 "marker": {"id": 1, "coords": {"latitude": 52.26164373072154, "l  
 "shop": "Lidl",  
 "name": "Latte Lite",  
 "v": 0,  
 "favourite": true,  
 "rating": 2,  
 "price": 2.99},  
 {"_id": "57602fbea4fcc00e00604bb0", "googlephoto": "https://lh6.  
 "usertoken": "115179585418026707309"}]
```

To the right of the JSON input, there are several configuration fields:

- Package: `ie.cm`
- Class name: `Coffee`
- Target language:
 - Java
 - Scala
- Source type:
 - JSON Schema
 - JSON
 - YAML Schema
 - YAML
- Annotation style:
 - Jackson 2.x
 - Jackson 1.x
 - Gson
 - Moshi
 - None
- Checkboxes for options:
 - Generate builder methods
 - Use primitive types
 - Use long integers
 - Use double numbers
 - Use Joda dates
 - Use Commons-Lang3
 - Include getters and setters
 - Include constructors
 - Include `hashCode` and `equals`
 - Include `toString`
 - Include JSR-303 annotations
 - Allow additional properties
 - Make classes serializable
 - Make classes parcelable
 - Initialize collections
- Property word delimiters: `- _`

At the bottom of the form are two buttons: "Preview" and "Zip". A "Fork me on GitHub" button is located in the top right corner of the page.



Bridging the Gap Between Your Code & Your API

Your annotated class

The screenshot shows a Mac OS X browser window displaying the jsonschema2pojo website at www.jsonschema2pojo.org. The page title is "jsonschema2pojo" and the subtitle is "Generate Plain Old Java Objects from JSON or JSON-Schema". A JSON schema is pasted into the input area, and a modal dialog titled "Preview" displays the generated Java code for a "Coffee" class. The Java code includes annotations like @SerializedName and @Expose. The "Copy to Clipboard" button is visible in the top right of the preview dialog. At the bottom of the dialog, there are "Preview" and "Zip" buttons, and checkboxes for "Initialize collections" and "Property word delimiters". The URL bar shows the site's address.

```
1 [ {"_id": "57602f7ea4",  
2   "googlephoto": "http://",  
3   "usertoken": "1157",  
4   "address": "Thomas",  
5   "marker": { "id": 1,  
6     "shop": "Lidl",  
7     "name": "Latte Lit",  
8     "v": 0,  
9     "favourite": true,  
10    "rating": 2,  
11    "price": 2.99},  
12   {"id": "57602fbbe4",  
13     "id": "57602f7ea4",  
14     "googlephoto": "http://",  
15     "usertoken": "1157",  
16     "address": "Thomas",  
17     "marker": { "id": 1,  
18       "shop": "Lidl",  
19       "name": "Latte Lit",  
20       "v": 0,  
21       "favourite": true,  
22       "rating": 2,  
23       "price": 2.99},  
24   } ]  
-----  
ie.cm.Coffee.java-----  
-----  
package ie.cm;  
  
import com.google.gson.annotations.Expose;  
import com.google.gson.annotations.SerializedName;  
  
public class Coffee {  
  
    @SerializedName("_id")  
    @Expose  
    public String id;  
    @SerializedName("googlephoto")  
    @Expose  
    public String googlephoto;  
    @SerializedName("usertoken")  
    @Expose  
    public String usertoken;  
    @SerializedName("address")  
    @Expose  
    public String address;  
    @SerializedName("marker")  
    @Expose  
    public Marker marker;  
    @SerializedName("shop")  
    @Expose  
    public String shop;  
    @SerializedName("name")  
    @Expose  
    public String name;  
    @SerializedName("__v")  
    @Expose  
    public Integer v;  
    @SerializedName("favourite")  
    @Expose  
    public Boolean favourite;  
    @SerializedName("rating")  
    @Expose  
    public Integer rating;  
    @SerializedName("price")  
    @Expose  
    public Integer price;  
  
    /**  
     * No args constructor for use in serialization  
     *  
     */  
    public Coffee() {  
    }  
  
    /**  
     * @param id  
     * @param v  
     * @param shop  
     * @param name  
     * @param googlephoto  
     * @param usertoken  
     * @param address  
     * @param rating  
     * @param price  
     * @param favourite  
     */  
}
```





Bridging the Gap Between Your Code & Your API

□ Date Formats

- ❑ Another potential disconnect between the app and the server is the way they represent date objects.
- ❑ For instance, Rails may send a date to your app in the format "yyyy-MM-dd'T'HH:mm:ss" which Gson will not be able to convert to a Java Date. We have to explicitly tell the converter how to perform this conversion. In this case, we can alter the Gson builder call to look like this:

```
Gson gson = new GsonBuilder()
    .setDateFormat("yyyy-MM-dd'T'HH:mm:ss")
    .create();
```



Additional Info - Headers

- ❑ If you wish to add headers to your calls, you can annotate your method or arguments to indicate this.
- ❑ For instance, if we wanted to add a content type and a specific user's authentication token, we could do something like this:

```
@POST("/coffees")
@Headers({ "Content-Type: application/json" })
Call<Coffee> createCoffee(@Body Coffee coffee,
                           @Header("Authorization") String token);
```



Full List of Retrofit Annotations

- @POST
- @PUT
- @GET
- @DELETE
- @Header
- @PATCH
- @Path
- @Query
- @Body



References

- ❑ <http://square.github.io/retrofit/>
- ❑ <https://spin.atomicobject.com/2017/01/03/android-rest-calls-retrofit2/>
- ❑ <https://code.tutsplus.com/tutorials/getting-started-with-retrofit-2--cms-27792>

