

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
- ❑ Recap on Java interfaces
- ❑ Understand how to integrate Retrofit into an Android App
- ❑ Revisit our Donation 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 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



Why Use it?

- ❑ Developing your own type-safe HTTP library to interface with a REST API can be a real pain: 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 Donation example that makes a **GET** request for Donations.
- ❑ Here's the **Donation** class we'll be using:
- ❑ Much simpler if field names matches server model (but doesn't have to, see later)

```
public class Donation
{
    public String _id;
    public int amount;
    public String paymenttype;
    public int upvotes;
    public double latitude;
    public double longitude;

    public Donation (int amount, String paymenttype,
                    int upvotes, double lat, double lng)
    {
        this.amount = amount;
        this.paymenttype = paymenttype;
        this.upvotes = upvotes;
        this.latitude = lat;
        this.longitude = lng;
    }

    public Donation ()
    {...}

    public String toString()
    {...}
}
```



The Service interface *

- Once we've defined the class, we can make a service interface to handle our API. A GET request to load all Donations could look something like this:

```
public interface DonationService
{
    @GET("/donations")
    Call<List<Donation>> getAllDonations();
}
```

- 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 list of Donations.



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



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



Calling the API *

- ❑ First, implement the necessary interface & variables

```
public class DonationApp extends Application
    implements Callback<List<Donation>>
{
    public final int      target      = 10000;
    public int            totalDonated = 0;
    public List <Donation> donations   = new ArrayList<Donation>();
    //public DBManager dbManager;

    public DonationService donationService;
    public boolean        donationServiceAvailable = false;
    public String         service_url   = "http://donationweb-4-0.herokuapp.com/";
    //public String         service_url   = "http://10.0.2.2:4000";
    //Standard Emulator IP Address
}
```

Note the
Callback
interface

Note the **service_url**



DonationApp - onCreate()

- ❑ Create the proxy service ‘donationService’, with the appropriate Gson parsers

```
@Override  
public void onCreate()  
{  
    super.onCreate();  
    Gson gson = new GsonBuilder().create();  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(service_url)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .build();  
    donationService = retrofit.create(DonationService.class);  
  
    Log.v("Donation", "Donation App Started");  
}
```



Calling the API – onCreate() *

Gson for converting our JSON

```
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();
```

OkHttpClient for communication timeouts (optional)

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(service_url)
    .addConverterFactory(GsonConverterFactory.create(gson))
    .client(okHttpClient)
    .build();
```

Retrofit.Builder to create an instance of our interface

```
donationService = retrofit.create(DonationService.class);
```

enqueue() allows for asynchronous callback to our service

```
Log.v( tag: "Donate", msg: "Donation Service Created" );
```

```
Call<List<Donation>> call1 = (Call<List<Donation>>) donationService.getAllDonations();
call1.enqueue( callback: this );
```



Calling the API – 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<List<Donation>> call, Response<List<Donation>> response) {  
    serviceAvailableMessage();  
    Log.v( tag: "retrofit", msg: "JSON = " + response.raw());  
    donations = response.body();  
    donationServiceAvailable = true;  
}
```



Calling the API – onFailure() *

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

```
@Override  
public void onFailure(Call<List<Donation>> call, Throwable t) {  
    donationServiceAvailable = false;  
    Log.v( tag: "retrofit", t.getMessage());  
    serviceUnavailableMessage();  
}
```



Calling the API – Helper methods

- ❑ Our 2 helper methods for user feedback

```
void serviceUnavailableMessage()
{
    Toast toast = Toast.makeText( context: this,
        text: "Donation Service Unavailable. Try again later",
        Toast.LENGTH_LONG );
    toast.show();
}

void serviceAvailableMessage()
{
    Toast toast = Toast.makeText( context: this,
        text: "Donation Contacted Successfully",
        Toast.LENGTH_LONG );
    toast.show();
}
```



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 DonationService interface

```
public interface DonationService {  
    @GET("/donations")  
    Call<List<Donation>> getAllDonations();  
  
    @GET("/donations/{id}")  
    Call<List<Donation>> getDonation(@Path("id") String id);  
  
    @DELETE("/donations/{id}")  
    Call<Donation> deleteDonation(@Path("id") String id);  
  
    @POST("/donations")  
    Call<Donation> createDonation(@Body Donation donation);  
  
    @PUT("/donations/{id}/votes")  
    Call<Donation> upVoteDonation(@Path("id") String id,  
                                    @Body Donation donation);  
}
```



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 user's authentication token, we could do something like this:

```
@POST("/donations")
@Headers({ "Content-Type: application/json"})
Call<Donation> createDonation(@Body Donation donation,
                                @Header("Authorization") String token);
```



Bridging the Gap Between Your Code & Your API

❑ Variable Names

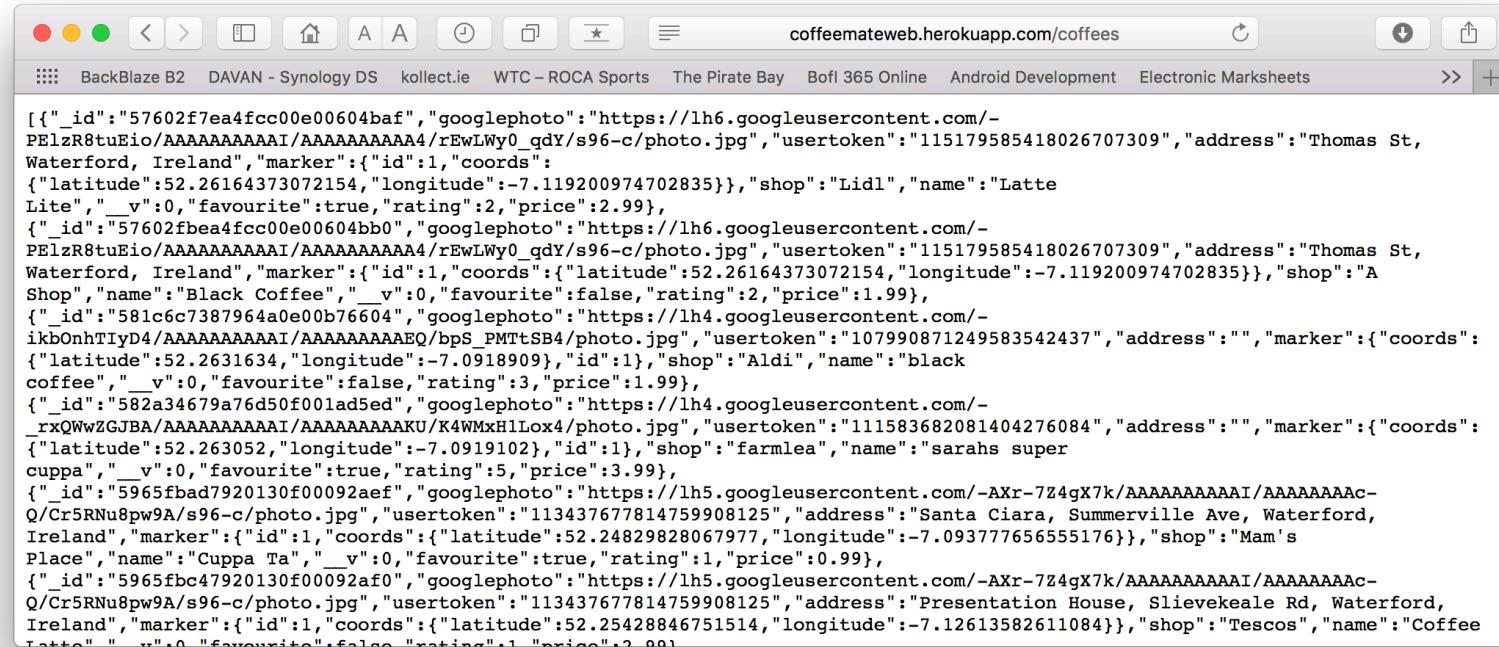
- In the previous examples, we assumed that there was an exact mapping of instance variable names between the Donation class and the server. This will often not be the case, especially if your server uses a different spelling convention than your Android app.
- For example, if you use a Rails server, you will probably be returning data using `snake_case`, while your Java probably uses `camelCase`. If we add a `dateCreated` to the Donation 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

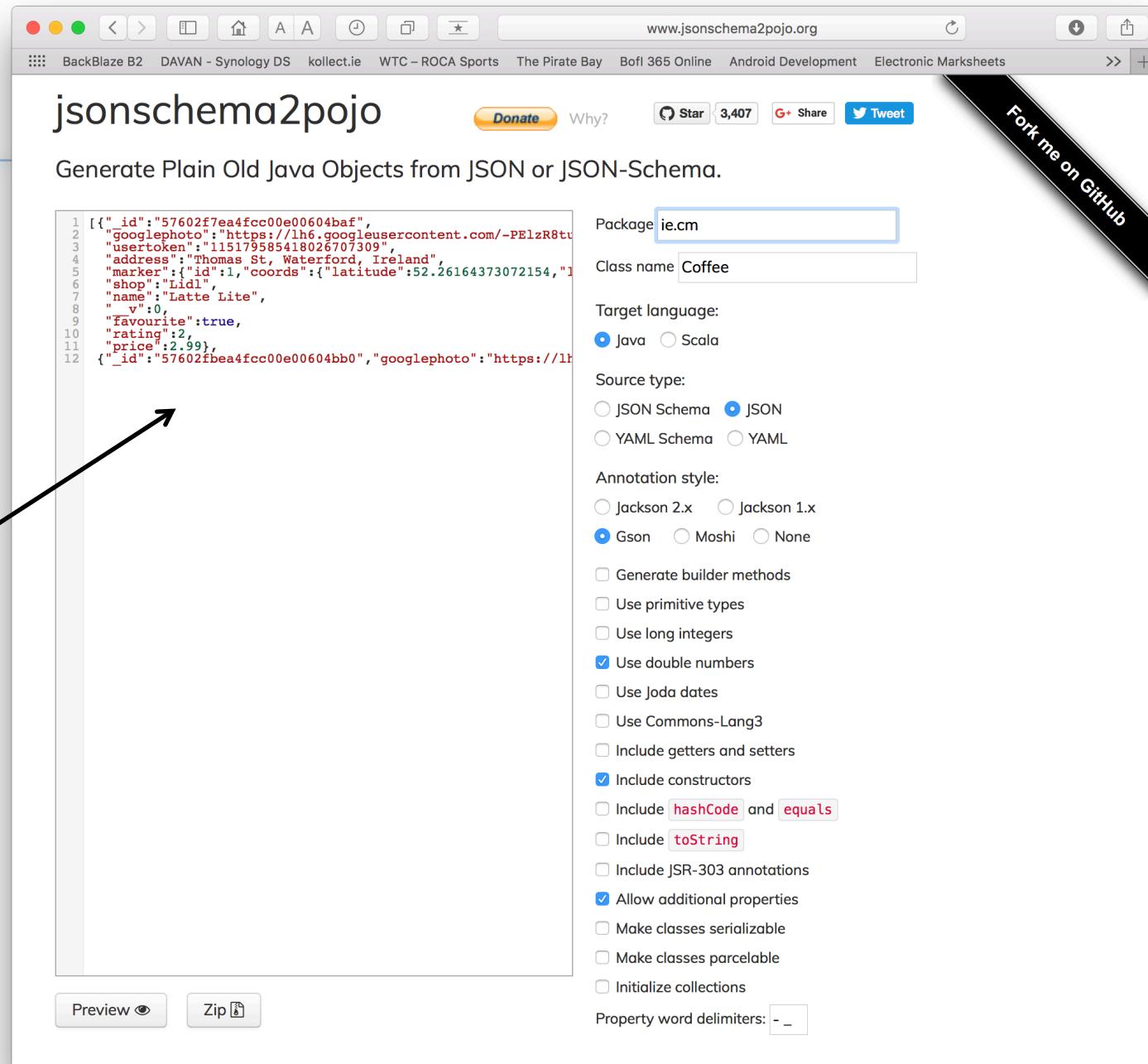
- You can also create your models automatically from your JSON response data by leveraging a very useful tool:
jsonschema2pojo - <http://www.jsonschema2pojo.org>
- Grab your JSON string, visit the above link and paste it in, like SO



```
[{"_id": "57602f7ea4fcc00e00604baf", "googlephoto": "https://lh6.googleusercontent.com/-PELzRtuEio/AAAAAAAIAI/AAAAAAAAAA/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/AAAAAAAIAI/AAAAAAAAAA/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/-ikbDnhTIyD4/AAAAAAAIAI/AAAAAAAAEQ/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/AAAAAAAIAI/AAAAAAAIAKU/K4WMxH1Lox4/photo.jpg", "usertoken": "111583682081404276084", "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/AAAAAAAIAI/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/AAAAAAAIAI/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 Totto", "favourite": false, "rating": 1, "price": 0.99}
```

Bridging the Gap Between Your Code & Your API

Your JSON goes here





Bridging the Gap Between Your Code & Your API

Your annotated class

The screenshot shows a web browser window with the URL www.jsonschema2pojo.org. The page title is "jsonschema2pojo" and the subtitle is "Generate Plain Old Java Objects from JSON or JSON-Schema". A modal dialog box is open, titled "Preview", containing the generated Java code for a "Coffee" class. The code uses annotations like @SerializedName and @Expose from the com.google.gson.annotations package. The modal has a "Copy to Clipboard" button and a close button. At the bottom of the page, there are "Preview" and "Zip" buttons, and a checkbox for "Initialize collections". A watermark "Fork me on GitHub" is visible on the right side of the page.

```
1 { "_id": "57602f7ea4",  
2   "googlephoto": "http://.../  
3   "usertoken": "115...",  
4   "address": "Thomas ...  
5   "marker": { "id": 1,  
6     "shop": "Lidl",  
7     "name": "Lidl - Lidl ...  
8     "v": 0,  
9     "favourite": true,  
10    "rating": 2,  
11    "price": 2.99},  
12  {"id": "57602f1bea4..."  
-----  
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 price  
     * @param marker  
     */  
}
```



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();
```



Full List of Retrofit Annotations

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



A bit about JSON



Question?

- ❑ Given a particular set of data, how do you store it permanently?
 - What do you store on disk?
 - What format?
 - Can you easily transmit over the web?
 - Will it be readable by other languages?
 - Can humans read the data?
- ❑ Examples:
 - A Square
 - A Dictionary
 - A Donation...



Storage Using Plain Text?

❑ Advantages

- Human readable (good for debugging / manual editing)
- Portable to different platforms
- Easy to transmit using web

❑ Disadvantages

- Takes more memory than necessary

❑ Alternative? - use a standardized system -- JSON

- Makes the information more portable



JavaScript Object Notation – What is it?

- JSON is lightweight text-data interchange format
- JSON is language independent*
 - *JSON uses JavaScript syntax for describing data objects
 - JSON parsers and JSON libraries exists for many different programming languages.
- JSON is "self-describing" and easy to understand
- **JSON - Evaluates to JavaScript Objects**
 - The JSON text format is syntactically identical to the code for creating JavaScript objects.
 - Because of this similarity, instead of using a parser, a JavaScript program can use the built-in `eval()` function and execute JSON data to produce native JavaScript objects.



JSON example

curly brackets

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "age": 25,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": 10021  
    },  
    "phoneNumbers": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "fax",  
            "number": "646 555-4567"  
        }  
    ]  
}
```

square brackets



A Donation in JSON

```
{  
  "_id": "5a23238c3032e512006c6de1",  
  "longitude": -7.08867073059082,  
  "latitude": 52.25166129217113,  
  "amount": 1001,  
  "paymenttype": "PayPal",  
  "__v": 0,  
  "upvotes": 1  
}
```



JSON Values

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes)
- A boolean (true or false)
- An *object* (in curly brackets)
- An *array* (in square brackets)
- null



JSON Values

❑ Object

- Unordered set of name-value pairs
- names must be strings
- { name1 : value1, name2 : value2, ..., nameN : valueN }

❑ Array

- Ordered list of values
- [value1, value2, ... valueN]



Google's Gson

<https://sites.google.com/site/gson/gson-user-guide>

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson is an open-source project hosted at [http://code.google.com/p/google-gson.](http://code.google.com/p/google-gson)

Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.



Android Networking (Using Retrofit) in Donation.4.1



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.



1. Project Dependencies & Permissions *

- ❑ Add the required dependencies to your build.gradle

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:appcompat-v7:25.4.0'  
    compile 'com.android.support:design:25.4.0'  
    compile 'com.squareup.retrofit2:retrofit:2.1.0'  
    compile 'com.google.code.gson:gson:2.8.2'  
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'  
    compile 'com.squareup.okhttp3:okhttp:3.7.0'  
}
```

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

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



Aside - Donation Android Client

❑ donation-web api endpoints

```
{ method: 'GET', path: '/donations', config: DonationsApi.findAllDonations },
{ method: 'GET', path: '/donations/{id}', config: DonationsApi.findOneDonation },
{ method: 'POST', path: '/donations', config: DonationsApi.makeDonation },
{ method: 'PUT', path: '/donations/{id}/votes', config: DonationsApi.incrementUpvotes },
{ method: 'DELETE', path: '/donations/{id}', config: DonationsApi.deleteOneDonation }
```

❑ Use donation-service for

- Donate
- List All Donations
- Find one Donation
- Delete one Donation
- Upvote a Donation



2. Create interface for API

The screenshot shows the Android Studio project structure. The main project is 'Donation.4.1'. Inside the 'app' module, there are 'manifests', 'java', 'res', and 'Gradle Scripts' directories. The 'java' directory contains a package 'ie.app' which includes 'activities' (with files Base, Donate, Login, Register, Report, Splash) and 'main' (with files DonationApp and DonationService). The 'models' directory contains a file Donation. The 'ie.app' directory is highlighted in green, indicating it is the current test target. The 'res' directory contains 'drawable', 'layout', 'menu', 'mipmap', and 'values' sub-directories.

```
public interface DonationService
{
    @GET("/donations")
    Call<List<Donation>> getAllDonations();

    @GET("/donations/{id}")
    Call<List<Donation>> getDonation(@Path("id") String id);

    @DELETE("/donations/{id}")
    Call<Donation> deleteDonation(@Path("id") String id);

    @POST("/donations")
    Call<Donation> createDonation(@Body Donation donation);

    @PUT("/donations/{id}/votes")
    Call<Donation> upVoteDonation(@Path("id") String id,
                                    @Body Donation donation);
}
```



3. Build Client using Retrofit

```
public class DonationApp extends Application
                    implements Callback<List<Donation>>
{
    public final int          target      = 10000;
    public int                totalDonated = 0;
    public List <Donation> donations     = new ArrayList<Donation>();
    //public DBManager dbManager;

    public DonationService donationService;
    public boolean            donationServiceAvailable = false;
    public String              service_url   = "http://donationweb-4-0.herokuapp.com/";
    //public String             service_url   = "http://10.0.2.2:4000";
                                            //Standard Emulator IP Address
}
```



3. Build Client using Retrofit

```
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(service_url)
    .addConverterFactory(GsonConverterFactory.create(gson))
    .client(okHttpClient)
    .build();

donationService = retrofit.create(DonationService.class);

Log.v(tag: "Donate", msg: "Donation Service Created");

Call<List<Donation>> call1 = (Call<List<Donation>>) donationService.getAllDonations();
call1.enqueue(callback: this);
```



3. Build Client using Retrofit

```
@Override  
public void onResponse(Call<List<Donation>> call, Response<List<Donation>> response) {  
    serviceAvailableMessage();  
    Log.v( tag: "retrofit", msg: "JSON = " + response.raw());  
    donations = response.body();  
    donationServiceAvailable = true;  
}  
  
@Override  
public void onFailure(Call<List<Donation>> call, Throwable t) {  
    donationServiceAvailable = false;  
    Log.v( tag: "retrofit",t.getMessage());  
    serviceUnavailableMessage();  
}
```



3. Build Client using Retrofit *

```
public class Report extends Base implements OnItemClickListener,  
    OnClickListener,  
    Callback<List<Donation>> {  
  
    ListView listView;  
    DonationAdapter adapter;  
    SwipeRefreshLayout mSwipeRefreshLayout;  
    Call<List<Donation>> call1;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_report);  
  
        mSwipeRefreshLayout = (SwipeRefreshLayout)  
            findViewById(R.id.report_swipe_refresh_layout);  
        listView = (ListView) findViewById(R.id.reportList);  
        listView.setOnItemClickListener(this);  
  
        call1 = app.donationService.getAllDonations();  
        call1.enqueue(callback: this);  
  
        mSwipeRefreshLayout.setOnRefreshListener(() -> {  
            call1 = app.donationService.getAllDonations();  
            call1.enqueue(callback: Report.this);  
        });  
    }  
}
```

Call declared here
for reuse



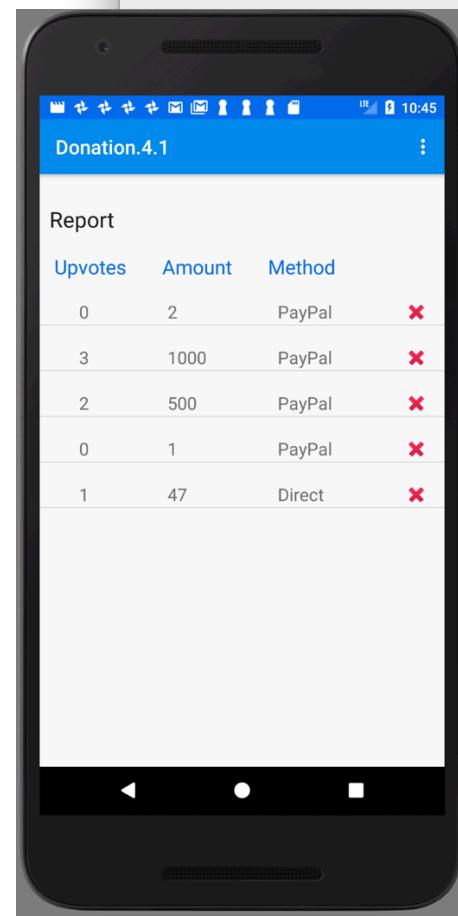
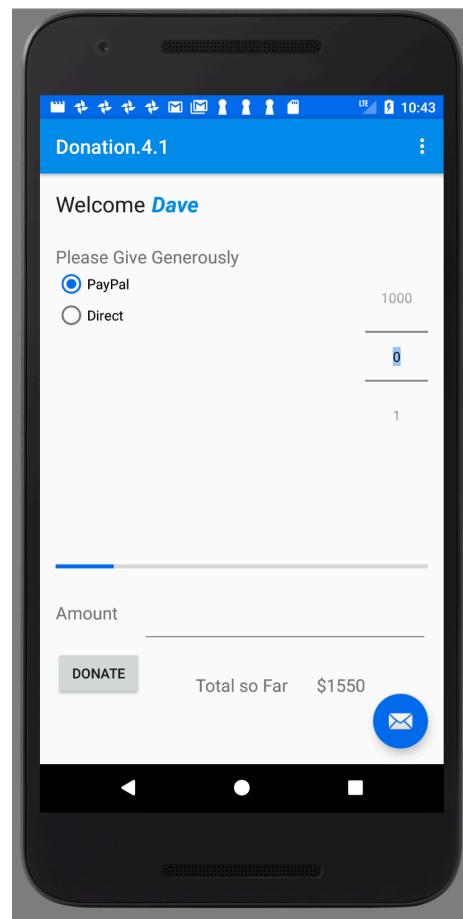
3. Build Client using Retrofit *

Anonymous Callback
allows for multiple calls
in same class

```
@Override
public void onItemClick(AdapterView<?> arg0, View row, int pos, long id) {
    String _id = row.getTag().toString();
    Call<List<Donation>> call2 = app.donationService.getDonation(_id);
    call2.enqueue(new Callback<List<Donation>>() {
        @Override
        public void onResponse(Call<List<Donation>> call,
                               Response<List<Donation>> response) {
            Toast toast = Toast.makeText(context: Report.this, text: "Donation : " +
                response.body() + " Selected.", Toast.LENGTH_LONG);
            toast.show();
        }

        @Override
        public void onFailure(Call<List<Donation>> call, Throwable t) {
            Toast toast = Toast.makeText(context: Report.this,
                text: "Error retrieving Donation", Toast.LENGTH_LONG);
            toast.show();
        }
    });
}
```

Donation Web + Mobile

A screenshot of a web browser window titled 'donationweb-4-0.herokuapp.com/#/donations'. The page is titled 'List All Donations' and displays a table of donations:

Upvotes	Method	Amount	Action
3	PayPal	€ 1000	X Remove E Edit
2	PayPal	€ 500	X Remove E Edit
1	Direct	€ 47	X Remove E Edit
0	PayPal	€ 2	X Remove E Edit
0	PayPal	€ 1	X Remove E Edit



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>

