Mobile Application Development

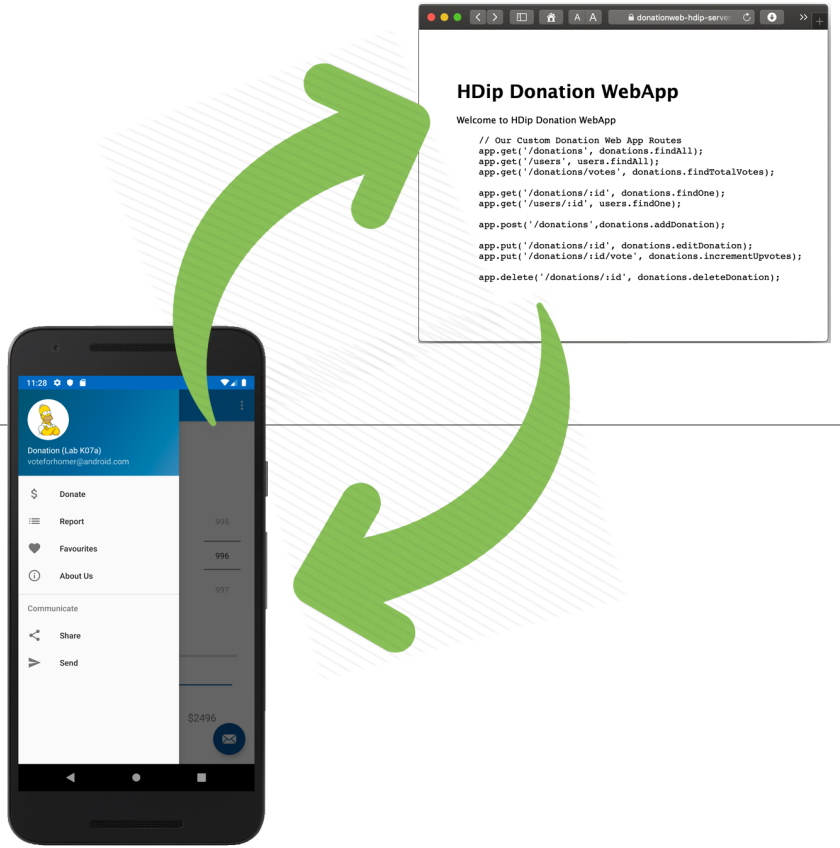
Produced by

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

Donation-V3 Walkthrough



Aside - Donation Service

- http://donationweb-hdip-server.herokuapp.com api endpoints

```
{ method: 'GET', path: '/donations', config: Donations.findAll },
{ method: 'GET', path: '/donations/{id}', config: Donations.findOne },
{ method: 'POST', path: '/donations', config: Donations.addDonation },
{ method: 'PUT', path: '/donations/{id}', config: Donations.editDonation },
{ method: 'DELETE', path: /donations/{id}', config: Donations.deleteDonation }
```


```
{
    "message": "Donation Successfully Added!",
    "data": {
        "upvotes": 0,
        "_id": "5daec8d0a904af0017dc5e00",
        "paymenttype": "PayPal",
        "amount": 1101,
        "message": "Another Test",
        "__v": 0
    }
}
```


- Use **DonationService** for
  - Adding / Updating / Deleting a Donation
  - Listing All Donations
  - Finding a single Donation

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

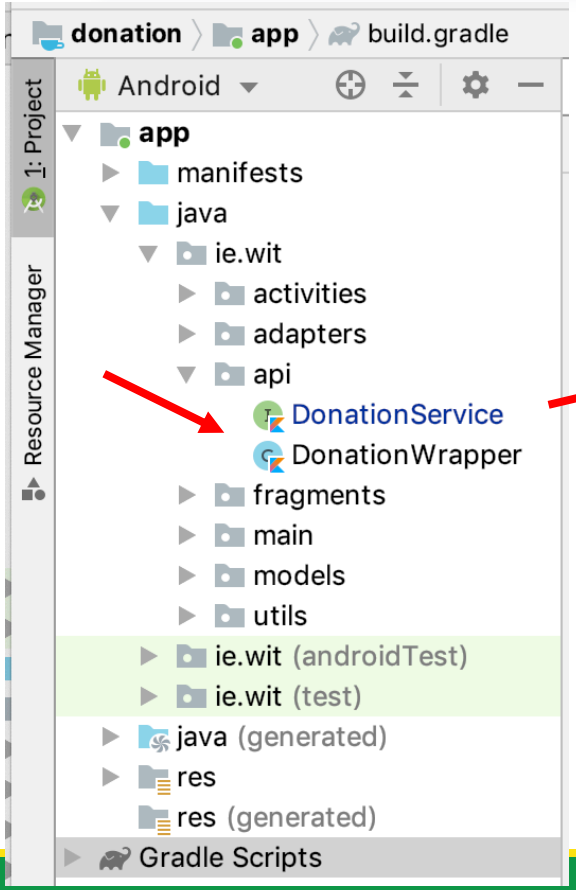
- Add the required dependencies to your build.gradle

```
implementation 'com.squareup.retrofit2:retrofit:2.6.0'
implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
implementation 'com.google.code.gson:gson:2.8.6'
```

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

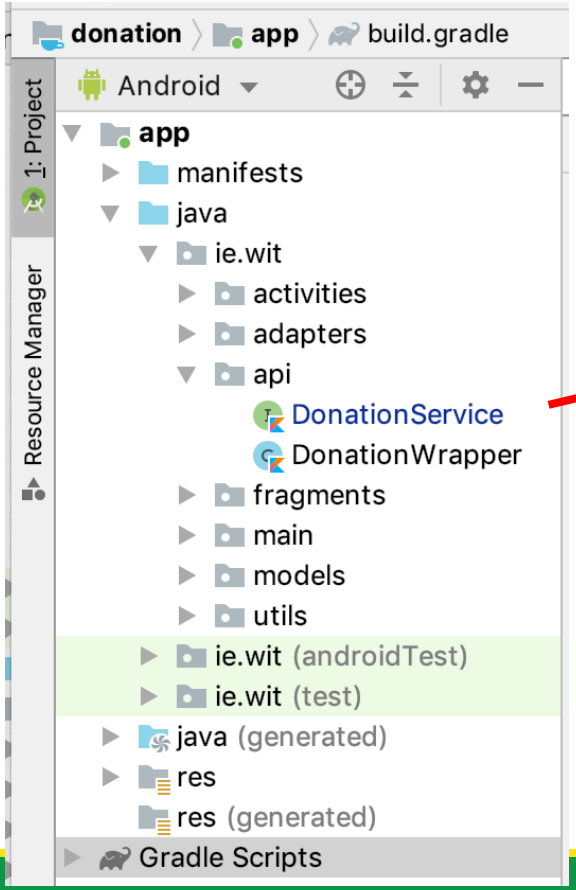
```
<uses-permission android:name="android.permission.INTERNET"/>
```

2. Create interface (and Wrapper) for API




```kotlin
interface DonationService {
    @GET("/donations")
    fun getall(): Call<List<DonationModel>>

    @GET("/donations/{id}")
    fun get(@Path("id") id: String): Call<DonationModel>

    @DELETE("/donations/{id}")
    fun delete(@Path("id") id: String): Call<DonationWrapper>

    @POST("/donations")
    fun post(@Body donation: DonationModel): Call<DonationWrapper>

    @PUT("/donations/{id}")
    fun put(@Path("id") id: String,
            @Body donation: DonationModel
    ): Call<DonationWrapper>
```

2. Create interface (and Wrapper) for API



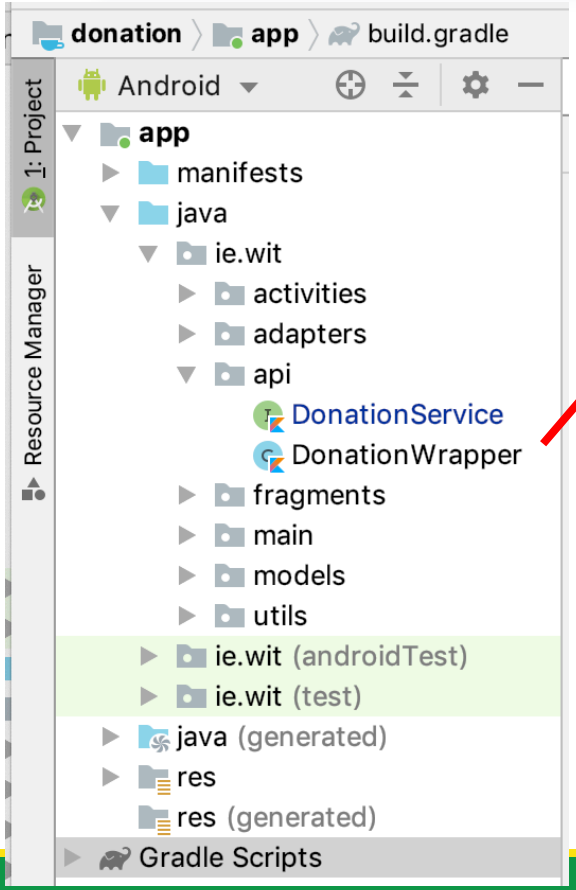

```
companion object {

    val serviceURL = "https://donationweb-hdip-server.herokuapp.com"

    fun create() : DonationService {

        val gson = GsonBuilder().create()

        val okHttpClient = OkHttpClient.Builder()
            .connectTimeout(30, TimeUnit.SECONDS)
            .writeTimeout(30, TimeUnit.SECONDS)
            .readTimeout(30, TimeUnit.SECONDS)
            .build()

        val retrofit = Retrofit.Builder()
            .baseUrl(serviceURL)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .client(okHttpClient)
            .build()
        return retrofit.create(DonationService::class.java)
    }
}
```

Gson for converting our JSON

OkHttpClient for timeouts (optional)

Retrofit.Builder to create an instance of our interface

2. Create interface (and Wrapper) for API




```kotlin
class DonationWrapper {
    var message: String? = null
    var data: DonationModel? = null
}
```

```
{
    "message": "Donation Successfully Added!",
    "data": {
        "upvotes": 0,
        "_id": "5daec8d0a904af0017dc5e00",
        "paymenttype": "PayPal",
        "amount": 1101,
        "message": "Another Test",
        "__v": 0
    }
}
```

3. Create Service Client - DonationApp

```
class DonationApp : Application(), AnkoLogger {

    lateinit var donationService: DonationService
    var donations = ArrayList<DonationModel>()

    override fun onCreate() {
        super.onCreate()
        info("Donation App started")
        donationService = DonationService.create()
        info("Donation Service Created")
    }
}
```

Our **DonationService** instance

4. Calling the API - ReportFragment

- Implement the necessary interface

Note the **Callback** interface

```
class ReportFragment : Fragment(), AnkoLogger,
                        Callback<List<DonationModel>> {
```

- and Callback objects

Called inside **onResume()**

```
fun getAllDonations() {
    showLoader(loader, "Downloading the Donations List")
    var callGetAll = app.donationService.getall()
    callGetAll.enqueue(this)
}
```

enqueue() allows for asynchronous callback to our service

4. ReportFragment – `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 fun onResponse(call: Call<List<DonationModel>>, response: Response<List<DonationModel>>) {
    serviceAvailableMessage(activity!!)
    info("Retrofit JSON = ${response.body()}")
    app.donations = response.body() as ArrayList<DonationModel>
    root.recyclerView.adapter = DonationAdapter(app.donations)
    root.recyclerView.adapter?.notifyDataSetChanged()
    checkSwipeRefresh()
    hideLoader(loader)
}
```

4. ReportFragment – `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 fun onFailure(call: Call<List<DonationModel>>, t: Throwable) {
    info("Retrofit Error : $t.message")
    serviceUnavailableMessage(activity!!)
    checkSwipeRefresh()
    hideLoader(loader)
}
```

Anonymous Callbacks

Anonymous Callbacks allows for multiple calls in same class

```
fun deleteDonation(id: String) {
    showLoader(loader, "Deleting Donation $id")
    var callDelete = app.donationService.delete(id)
    callDelete.enqueue(object : Callback<DonationWrapper> {
        override fun onFailure(call: Call<DonationWrapper>, t: Throwable) {
            info("Retrofit Error : $t.message")
            serviceUnavailableMessage(activity!!)
            hideLoader(loader)
        }

        override fun onResponse(call: Call<DonationWrapper>,
                    response: Response<DonationWrapper>)
                        { hideLoader(loader) }
    })
}
```

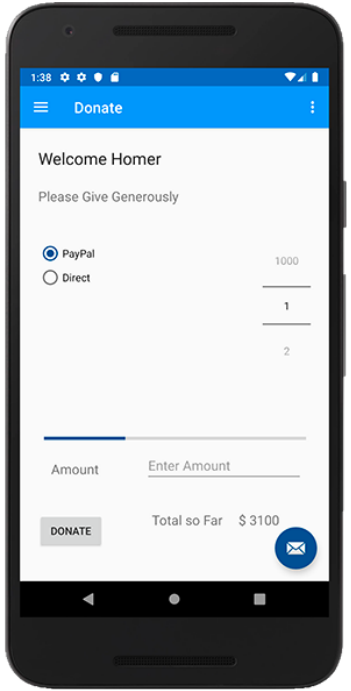
Helper Classes - SwipeToDeleteCallback

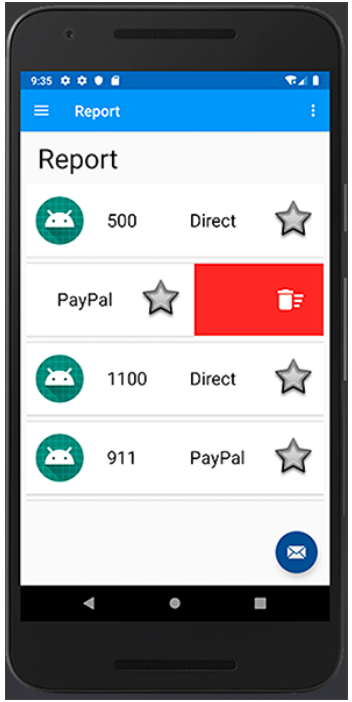
```kotlin
abstract class SwipeToDeleteCallback(context: Context) :
    ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT) {

    private val deleteIcon = ContextCompat.getDrawable(context, R.drawable.ic_swipe_delete)
    private val intrinsicWidth = deleteIcon?.intrinsicWidth
    private val intrinsicHeight = deleteIcon?.intrinsicHeight
    private val background = ColorDrawable()
    private val backgroundColor = Color.parseColor("#f44336")
    private val clearPaint = Paint().apply { xfermode = PorterDuffXfermode(PorterDuff.Mode.CLEAR) }

    override fun getMovementFlags(recyclerView: RecyclerView, viewHolder: RecyclerView.ViewHolder):

    override fun onMove(recyclerView: RecyclerView, viewHolder: RecyclerView.ViewHolder,
                        target: RecyclerView.ViewHolder): Boolean {...}

    override fun onChildDraw(
        c: Canvas, recyclerView: RecyclerView, viewHolder: RecyclerView.ViewHolder,
        dX: Float, dY: Float, actionState: Int, isCurrentlyActive: Boolean
    ) {...}

    private fun clearCanvas(c: Canvas?, left: Float, top: Float, right: Float, bottom: Float) {...}
}
```

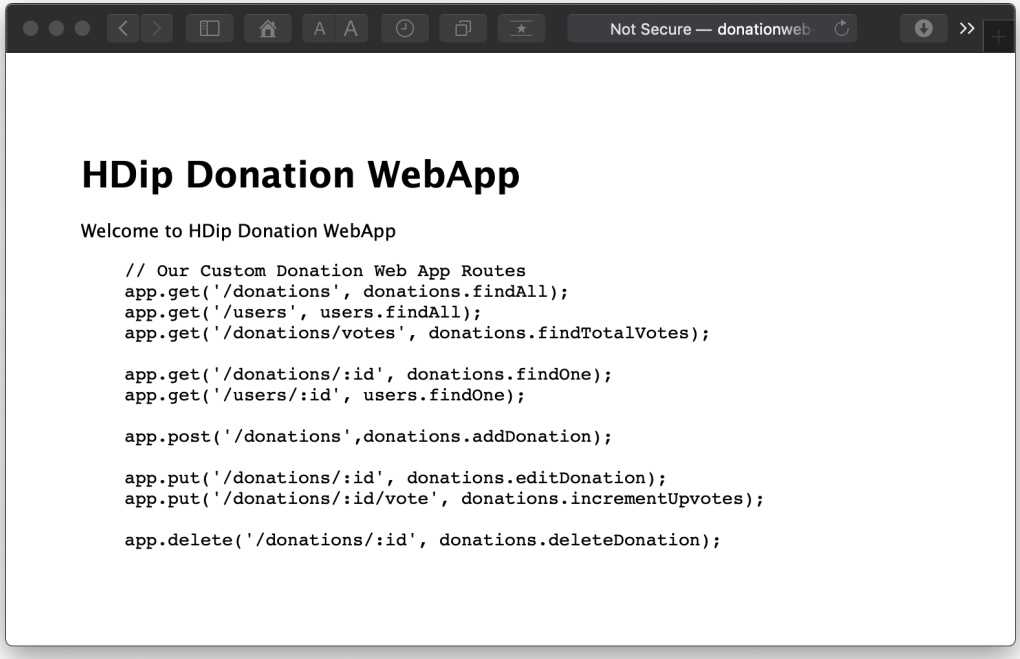
SwipeToDeleteCallback Usage

```
val swipeDeleteHandler = object : SwipeToDeleteCallback(activity!!) {
    override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
        val adapter = root.recyclerView.adapter as DonationAdapter
        adapter.removeAt(viewHolder.adapterPosition)
        deleteDonation(viewHolder.itemView.tag as String)
    }
}
val itemTouchDeleteHelper = ItemTouchHelper(swipeDeleteHandler)
itemTouchDeleteHelper.attachToRecyclerView(root.recyclerView)
```

Donation Service + Mobile App








Thanks.