Android REST demo

Current implementation

Current implementation is done using

- Background tasks
- AsyncTasks

Most extended method nowadays

Creates a lot of problems

AsyncTask vs RxJava approach

```java
public class NetworkRequestTask extends AsyncTask<Void, Void, User> {
    private final int userId;
    public NetworkRequestTask(int userId) {
        this.userId = userId;
    }
    @Override protected User doInBackground(Void... params) {
        return networkService.getUser(userId);
    }
    @Override protected void onPostExecute(User user) {
        nameTextView.setText(user.getName());
        // ...set other views
    }
}

private void onButtonClicked(Button button) {
    new NetworkRequestTask(123).execute()
}
```

```java
private Subscription subscription;

private void onButtonClicked(Button button) {
    subscription = networkService.getObservableUser(123)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Action1<User>() {
                @Override public void call(User user) {
                    nameTextView.setText(user.getName());
                    // ... set other views
                }
            });
}

@Override protected void onDestroy() {
    if (subscription != null && !subscription.isUnsubscribed()) {
        subscription.unsubscribe();
    }
    super.onDestroy();
}
```

AsyncTask vs RxJava approach

```
public class NetworkRequestTask extends AsyncTask<Void, Void, User> {
    private final int userId;
    public NetworkRequestTask(int userId) {
        this.userId = userId;
    }
    @Override protected User doInBackground(Void... params) {
        return networkService.getUser(userId);
    }
    @Overr
        nameTextView.setText(user.getName());
        // ...set other views
    }
}

private void onButtonCl
    new NetworkRequestTask(123).execute()
}
```

What if the UI is not there anymore

Inner class: leak risk

```
private Subscription subscription;

private void onButtonClicked(Button button) {

    subscription = networkService.getObservableUser(123)
            .subscribeOn(Schedulers.io())
                    nameTextView.setText(user.getName());
                    // ... set other views
                }
            });
}

@Override protected void onDestroy() {
    if (subscription != null && !subscription.isUnsubscribed()) {
        subscription.unsubscribe();
    }
    super.onDestroy();
}
```

Because of the unsubscription below, wont leak

Callback hell

```
new ServerCall(this, Script.CAMERA, Method.GET_CAMERAS, keyvalues, null, TIMEOUT_REQUEST)
        .setOnCompleteListener((OnCompleteListener) (response, parsedResponse) → {
                camerasData = parsedResponse.getXmlObject();

                new ServerCall(CamerasService.this, Script.ZONES, Method.GET_ZONES, keyvalues, null, TIMEOUT_REQUEST)
                        .setOnCompleteListener((OnCompleteListener) (response, parsedResponse) → {
                                if (parsedResponse.getXmlObject() != null)
                                    zonesData = parsedResponse.getXmlObject();
                                else
                                    zonesData = null;

                                new ServerCall(CamerasService.this, Script.CAMERA, Method.GET_GROUP_CAMERAS, keyvalues, null, TIMEOUT_REQUEST)
                                        .setOnCompleteListener((OnCompleteListener) (response, parsedResponse) → {
                                                camerasGroupData = parsedResponse.getXmlObject();

                                                new ServerCall(CamerasService.this, Script.CAMERA, Method.GET_CAMERA_STATUS_TREE, keyvalues, null, TIMEOUT_REQUEST)
                                                        .setOnCompleteListener((OnCompleteListener) (response, parsedResponse) → {
                                                                camerasStateData = parsedResponse.getXmlObject();
                                                                mApp.setCamerasStates(new CameraItemState(camerasStateData));
                                                                updateValuesIntoTheApp();
                                                        })
                                                        .setOnErrorListener((parsedResponse) → {
                                                                sendErrorIntent();
                                                                updateValuesIntoTheApp();
                                                                return true;
                                                        })
                                                        .execute();
                                        })
                                        .setOnErrorListener((parsedResponse) → {
                                                sendErrorIntent();
                                                return true;
                                        })
                                        .execute();
                        })
                        .setOnErrorListener((parsedResponse) → {
                                sendErrorIntent();
                                return true;
                        })
                        .execute();
        })
        .setOnErrorListener((parsedResponse) → {
                sendErrorIntent();
                return true;
        })
```

Real example from the current CM app

Chaining calls

```
public Observable<CameraTree> getCameraTree() {
    Observable<CameraTree> combined = Observable.zip(
            mZonesRepository.getZones(),
            mCamerasRepository.getCameras(),
            new Func2<List<Zone>, List<Camera>, CameraTree>() {
        @Override
        public CameraTree call(List<Zone> zones, List<Camera> cameras) {
            return new CameraTree(zones, cameras);
        }
    });
    return combined;
}
```

Filtering

```
Subscription subscription = mCamerasRepository
        .getCameras()
        .flatMap(new Func1<List<Camera>, Observable<Camera>>() {
            @Override
            public Observable<Camera> call(List<Camera> cameras) {
                return Observable.from(cameras);
            }
        })
        .filter(new Func1<Camera, Boolean>() {
            @Override
            public Boolean call(Camera camera) {
                switch (mCurrentFiltering) {
                    case RECORDING:
                        return camera.isRecording();
                    case ONLINE:
                        return camera.isOnline();
                    case ALL_CAMERAS:
                    default:
                        return true;
                }
            }
        })
        .toList()
        .subscribeOn(mSchedulerProvider.computation())
        .observeOn(mSchedulerProvider.ui())
        .subscribe(new Observer<List<Camera>>() {
            @Override
            public void onCompleted() {
                mCamerasView.setLoadingIndicator(false);
            }

            @Override
            public void onError(Throwable e) {
                mCamerasView.showLoadingCamerasError();
            }

            @Override
            public void onNext(List<Camera> cameras) {
                processCameras(cameras);
            }
        });
```

Retrofit + RxJava

```
public interface CameraService {

    @GET("cameras")
    public Observable<List<Camera>> getCameras();

    @GET("cameras/{cameraId}")
    public Observable<Camera> getCamera(@Path("cameraId") Long cameraId);

    @GET("cameras/{cameraId}/streams")
    public Observable<CameraStream> getCameraStream(@Path("cameraId") Long cameraId);

    @GET("cameras/{cameraId}/capabilities")
    public Observable<CameraCapabilities> getCameraCapabilities(@Path("cameraId") Long cameraId);

}
```