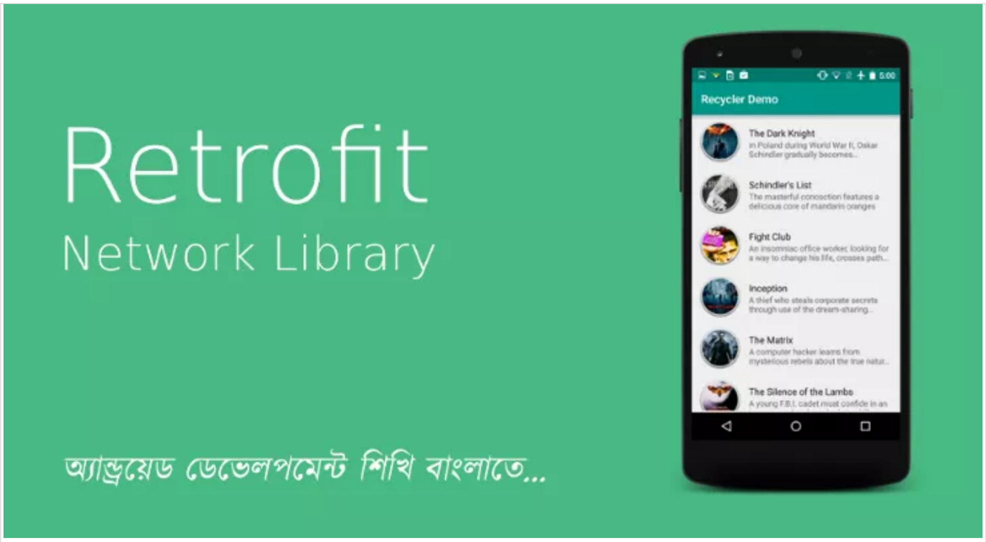
(https://www.facebook.com/hasan.cse91) (https://github.com/hasancse91) (https://medium.com/@hasan.cse91)
(https://bd.linkedin.com/in/abdullah-al-hasan-376030b1) (https://plus.google.com/u/0/+AbdullahAlHasanCSE91)

হাসানের রাফখাতা (https://hellohasan.com/)

পোস্টটি পড়া হয়েছে 4,765 বার



Android এ Retrofit ব্যবহার করে GET ও POST রিকোয়েস্ট

December 3, 2016 (https://hellohasan.com/2016/12/03/android-retrofit-get-post-method/) / Hasan Abdullah (https://hellohasan.com/author/hasan-cse91/) / অ্যান্ড্রয়েড অ্যাপ ডেভেলপমেন্ট (https://hellohasan.com/category/android-tutorial/), Android Library (https://hellohasan.com/category/android-tutorial/popular-android-library/)

Retrofit হচ্ছে Square এর ডেভেলপ করা একটা REST client বা network library. জাভা বা অ্যান্ড্রয়েডে এটা ব্যবহার করে খুব সহজেই নেটওয়ার্কের সাথে অর্থাৎ কোন ওয়েব সার্ভারের সাথে HTTP protocol এর মাধ্যমে কানেক্টেড হওয়া যায়। JSON ফরমেটে ডেটা আদান-প্রদানের জন্য এই লাইব্রেরিটা অতুলনীয়।

আমার ডেভেলপমেন্টের শুরুর দিকে নেটওয়ার্ক কল দেয়ার কাজ করেছিলাম AsyncTask এর মাধ্যমে। এরপর পেলাম গুগলের ডেভেলপ করা Volley library'র খোঁজ। গত প্রায় বছরখানেক ধরে সবগুলো প্রোজেক্টে ভলি লাইব্রেরি ইউজ করেছি। এখন পেলাম আরো সহজে ব্যবহার উপযোগি রেট্রোফিট লাইব্রেরি। ভলিতে কাজ করার সময় JSON ডেটা পাঠানো বা রিসিভ করার সময় manually parse করা লাগতো। রেট্রোফিটের বড় একটা সুবিধা হচ্ছে এটা JSON parse করার জন্য জনপ্রিয় আর বহুল

ব্যবহৃত Gson library ব্যবহার করে। তাই সার্ভারে ডেটা পাঠানোর সময় সেটাকে আপনার প্রয়োজন অনুসারে JSON বানিয়ে পাঠানোর দরকার হবে না। রিসিভ করার সময়েও আপনাকে নিজে হাতে ধরে ধরে পার্স করা লাগবে না। আপনি পাঠানোর সময় আপনার কোন একটা ক্লাসের অবজেক্টকে পাঠিয়ে দিবেন। Retrofit-ই Gson এর সহযোগিতায় সেটাকে JSON বানিয়ে সার্ভারে পাঠাবে। আবার রিসিভ করার পরে আপনি একটা সিংগেল মেথড কল করেই রেস্পন্সে পাওয়া ডেটাকে আপনার কাংক্ষিত ক্লাসের অবজেক্টে পার্স করতে পারবেন। আর ব্যবহার করাও তুলনামূলক সহজ। চলে যাচ্ছি সরাসরি প্রোজেক্টে।

Problem Definition

একটা Activity থাকবে। লগিন আইডি আর পাসওয়ার্ড ইনপুট দেয়ার ফিল্ড থাকবে। টেস্ট করার সুবিধার জন্য লগিন আইডি **hasan** আর পাসওয়ার্ড **123** ফিক্সড করে দেয়া হয়েছে। লগিন আইডি আর পাসওয়ার্ড ইনপুট দিয়ে লগিন বাটনে ক্লিক করলে সার্ভারে চলে যাবে এই আইডি-পাস। সেখানে চেক হবে আইডি-পাস ঠিক আছে কিনা। ঠিক থাকলে রেসপন্সে পাঠাবে 'Your are a valid user' আর ভুল হলে পাঠাবে 'User ID or Password is wrong'. যেহেতু এই নেটওয়ার্ক কলে পাসওয়ার্ড পাঠানো হচ্ছে তাই সিকিউরিটির একটা ব্যাপার আছে। তাই এই ডেটাগুলো পাঠানো হবে POST method এ।

একই একটিভিটিতে আরেকটা টেক্সট ইনপুটের ফিল্ড থাকবে। সেখানে ইউজার আইডি হিসেবে **hasan** দিয়ে সাবমিট করলে সার্ভার আপনাকে একটা joke পাঠাবে। Joke টা শুধু hasan আইডি এর জন্যেই পাঠানো হবে। অন্য কোন আইডি দিলে সার্ভার রেসপন্সে পাঠাবে 'You are not a valid user'. আর এই নেটওয়ার্ক রিকোয়েস্টটা হবে GET method এ।

উপরের দুইটা কাজ করার মাধ্যমে আমরা GET-POST দুইটা মেথডই শিখব।

Open a project

পছন্দ মত নাম দিয়ে একটা প্রোজেক্ট ওপেন করেন অ্যান্ড্রয়েড স্টুডিও ব্যবহার করে।

Add Retrofit to Gradle file

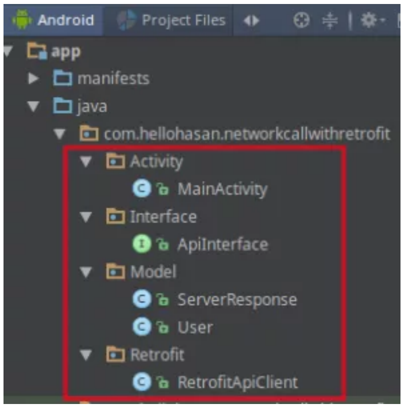
Retrofit আর Gson library ব্যবহার করার জন্য App level gradle ফাইলের dependencies এ অ্যাড করেন নিচের দুটি লাইনঃ

```
compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.squareup.retrofit2:converter-gson:2.0.2'
```

Sync করেন, প্রথম কাজ শেষ।

Create some packages and classes

MainActivity ছাড়াও আরো চারটা ক্লাস/ফাইল বানাতে হবে। ম্যানেজ করার সুবিধার্থে ৪ টা প্যাকেজ বানিয়ে নিয়ে পারেন ছবির মত করে। Package-গুলো হচ্ছেঃ Activity, Interface, Model ও Retrofit.



Activity প্যাকেজে MainActivity-কে drag and drop করুন। এরপর বাকি প্যাকেজগুলোতে যথাক্রমে ApiInterface, ServerResponse, User ও RetrofitApiClient নামের ক্লাসগুলো create করেন।

এখন একটা একটা করে ক্লাস দেখব আর বর্ণনা করব।

Model classes

মডেল ক্লাসের মধ্যে আমাদের ডেটাগুলো কিভাবে থাকবে না থাকবে সেটা বলা হয়েছে। Problem Definition থেকে এতক্ষণে হয়ত বুঝে গেছেন আমরা ইউজারের আইডি আর পাসওয়ার্ড ইনপুট নিচ্ছি। এটা সার্ভারে পাঠাচ্ছি। তাহলে User একটা ক্লাস হতে পারে। যার মধ্যে id, password এই ডেটাগুলো থাকবে।

```java
import com.google.gson.annotations.SerializedName;

public class User {

        @SerializedName("user_id")
        private String userId;
        @SerializedName("password")
        private String password;

        public User(){}

        public void setUserId(String userId) {
            this.userId = userId;
        }

        public void setPassword(String password) {
            this.password = password;
        }

}
```

সোজা-সাপটা একটা ক্লাস। দুইটা ডেটা আর দুইটা মেথড আছে। একটু কারিগরি ফলানো হয়েছে ডেটাগুলো declare করার আগের লাইনে। এইটুকু কারিগরিই আমাদের এই ক্লাসের অবজেক্টকে দরকার মত JSON বানিয়ে দিবে। যখন এই User ক্লাসের কোন অবজেক্টকে সার্ভারে পাঠাবো তখন Gson library খুঁজবে যে এই ক্লাসের ডেটাগুলোর নাম কী? সে এসে এই ক্লাসের ডেটা বা ভেরিয়েবলগুলোর নাম দিয়েই JSON key বানাবে। কিন্তু আমরা “userId” এরকম JSON key চাই না। আমাদের দরকার “user_id” এই ফরমেট। এটার জন্যেই ভেরিয়েবলের শুরুতে লিখে দেয়া হয়েছে @SerializedName(“user_id”). কারণ Gson library শুরুতে খুঁজে দেখে @SerializedName আছে কিনা। যদি থাকে তাহলে তার পরের ভেরিয়েবলের ডেটাকে @SerializedName এর ভিতরের string এর Key হিসেবে সার্ভারে পাঠায়।

যদি @SerializedName ব্যবহার না করতাম তাহলে এই ক্লাসের অবজেক্টকে সার্ভারে পাঠানো হত এই ফরমেটেঃ

```json
{
    "userId":"hasan",
    "password":"123"
}
```

কিন্তু @SerializedName এর ভিতরে প্যারামিটার দিয়ে সেট করে দেয়াতে আমাদের লিখা ক্লাসের অবজেক্টকে পাঠানো হবে এই ফরমেটেঃ

```
{
    "user_id":"hasan",
    "password":"123"
}
```

লাইফ অনেক সহজ হয়ে গেল তাই না? 🙂

এ তো গেল যেই ডেটা সার্ভারে পাঠাবো তার Model. সার্ভার থেকে তো কিছু ডেটা আসবে। সেগুলোকে রিসিভ করে ফরমেট করার জন্য আরেকটা ক্লাস দরকার। সেই ক্লাসের নাম দিয়েছি ServerResponse ক্লাস।

```
import com.google.gson.annotations.SerializedName;

public class ServerResponse {

        @SerializedName("status")
        boolean statusString; //variable name is statusString but it'll map with "stat
        @SerializedName("message")
        String messageString; //variable name is messageString but it'll map with "mes

        public boolean isSuccess(){
            return statusString;
        }

        public String getMessage() {
        return messageString;
    }
}
```

আমরা সার্ভারে দুইটা কল করছি। একটা করছি লগিন করার জন্য অর্থাৎ POST method শেখার জন্য। আরেকটা হচ্ছে সার্ভারে আইডি পাঠিয়ে একটা কৌতুক পড়তে চাচ্ছি। এটা GET method শেখার জন্য। টিউটোরিয়ালের সাইজ ছোট করার জন্য বুদ্ধি করে দুইটা রিকোয়েস্টের রেসপন্সের ডেটা একই রেখেছি। লগিনের সময় সার্ভার আমাকে দুইটা ডেটা পাঠাবে। একটা হচ্ছে **status** (login successful কিনা সেটার true-false স্ট্যাটাস) আরেকটা ডেটা হচ্ছে **message** (status true হলে সার্ভার পাঠাবে You are a valid user আর false হলে পাঠাবে ID/Password wrong). GET request এর কৌতুক রিসিভ করার জন্যেও একই JSON KEY ব্যবহার করা হচ্ছে। আইডি সঠিক হলে সার্ভার থেকে **status** এ true, সাথে **message** এ কৌতুক পাঠানো হবে। অন্যথায় **status** এ false আর **message** এ বলা হবে Invalid User.

এত কথা বলার কারণ হচ্ছে, আপনি যতগুলো সার্ভার কল করবেন তার রেসপন্সের ডেটাগুলো ভিন্ন ভিন্ন হতে পারে। সবগুলো ভিন্ন ভিন্ন JSON object এর জন্য আলাদা আলাদা response ক্লাস বানাতে হবে। এত্তগুলা ক্লাস বানাতে কষ্ট লাগবে? বানায়ে দেখেন, পরের কাজ কত্ত সোজা হয়ে যায়!

এরপরেও যারা ভয় পাচ্ছেন তাদের জন্য JetBrain এর POJO Generator প্লাগিন আছে যেটা Android Studio এর সাথে এড করে ব্যবহার করতে পারবেন। এতে একটা JSON object ইনপুট দিলে আউটপুট হিসেবে Java model class পাওয়া যায়। এছাড়াও online pojo class generator রয়েছে। গুগল করলেই সব পেয়ে যাবেন।

RetrofitApiClient class

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitApiClient {

        private static final String BASE_URL = "http://192.168.0.101"; //address of y
        private static Retrofit retrofit = null;

        private static Gson gson = new GsonBuilder()
                .setLenient()
                .create();

 private RetrofitApiClient() {} // So that nobody can create an object with construct

        public static synchronized Retrofit getClient() {
            if (retrofit==null) {
                retrofit = new Retrofit.Builder()
                        .baseUrl(BASE_URL)
                        .addConverterFactory(GsonConverterFactory.create(gson))
                        .build();
            }
            return retrofit;
        }

}
```

শুরুর BASE_URL এ আমার পিসির localhost এর IP বসানো আছে। ইমুলেটর বা রিয়েল ডিভাইস (যদি WiFi এর সাথে কানেক্টেড থাকে তাহলে রান করে) যেন চেক করা যায় তাই এই আইপি দেয়া। আপনার যদি রিয়েল সার্ভার থাকে তাহলে সেটার লিংক এখানে বসবে।

পুরো প্রোজেক্টে Retrofit এর একটা মাত্র instance-ই তৈরি হবে। সেটা নিয়ন্ত্রিত হচ্ছে getClient() মেথডের মাধ্যমে। নেটওয়ার্ক কলের জন্য RetrofitApiClient ক্লাসের একটা মাত্র অবজেক্টই সবাই ব্যবহার করবে। খেয়াল করে দেখেন, এই ক্লাসের constructor একটা private method. একটা প্রাইভেট মেথড দিয়ে কিভাবে এই ক্লাসের অবজেক্ট বানাবো? কোন ক্লাস থেকে এই ক্লাসের অবজেক্ট বানাতে গেলে কনস্ট্রাকটর ব্যবহার করা যাবে না। বরং ব্যবহার করতে হবে getClient() এই স্ট্যাটিক মেথডটি। এই মেথডের ভিতরে চেক করা হচ্ছে এই ক্লাসেরই একটা প্রাইভেট ডেটা retrofit অবজেক্টটা null কিনা। retrofit একটা প্রাইভেট ডেটা, এটার কোন সেটারও নাই। তাই অন্যান্য ক্লাস থেকে ইচ্ছা করলেও এই ডেটাকে ইনিশিয়ালাইজ করা সম্ভব না। প্রথমবার getClient() কল হবার সময় retrofit==null সত্য হবে। সত্য হলে এর ভিতরে retrofit কে initialize করা হয়েছে। এরপর return করে দেয়া হয়েছে। পরের বার আবার যদি getClient() কল করা হয় তখন এই প্রথম বার তৈরি হওয়া instance এর রেফারেন্সটাই রিটার্ন করে দেয়া হবে। এই যে কোন একটা ক্লাসের একটা অবজেক্টই শুধু তৈরি করা যাবে এই বাধ্যবাধ্যকতার একটা গাল ভরা নাম আছে। গুরুজনেরা এটাকে Singleton Design Pattern বলে থাকেন। এই ডিজাইন প্যাটার্ন ব্যবহার করে অনেক বড় বড় হাতি-ঘোড়া মারা যায়। শুধু Singleton Design Pattern এর উপর বিস্তারিত একটা ব্লগ পোস্ট করার ইচ্ছা আছে। আপাতত এইটুকু আইডিয়া থাকলেই হবে।

retrofit কে initialization এর সময় addConverterFactory() ব্যবহার করা হয়েছে যার প্যারামিটার হিসেবে দেয়া আছে ক্লাসের ভিতরে বানানো Gson ক্লাসের একটা প্রাইভেট object. এটা দিয়ে বুঝানো হয়েছে যে retrofit অবজেক্টের মাধ্যমে যত ডেটা পাঠানো হবে সেগুলো যাওয়ার আগে আমাদের মডেল ক্লাস অনুযায়ী সাইজ হয়ে যাবে। আর এই কনভার্ট বা সাইজ করার মহান দায়িত্ব পালন করতে সাহয্য করবে gson অবজেক্টটি।

ApiInterface class

এটা সাধারণ জাভা ক্লাস না। এটা ইন্টারফেস। আপনি ক্লাস বানানোর সময় এটা অটোমেটিক্যালি পাবলিক ক্লাস হিসেবে তৈরি হয়েছিল। নিচের কোডটা কপি পেস্ট করুন আপনার এই ফাইলেঃ

```java
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Query;

public interface ApiInterface {

    @POST("/retrofit_get_post/server_side_code.php")
    Call<ServerResponse> getUserValidity(@Body User userLoginCredential);

    @GET("/retrofit_get_post/server_side_code.php")
    Call<ServerResponse> getJoke(@Query("user_id") String userId);

}
```

একটা পোস্ট আরেকটা গেট মেথড উল্লেখ করা আছে। @POST annotation এর পরে উল্লেখ করা হয়েছে সার্ভারের রাউট। Base URL আমরা RetrofitApiClient ক্লাসে উল্লেখ করেছিলাম। সেই বেজ লিংক কিন্তু সাধারণত সবগুলো কলের ক্ষেত্রে একই থাকে। যেমন বেজ লিংক হিসেবে বলা হয়েছে লোকালহোস্টের আইপি বা localhost. localhost এ ঢুকে আমি একটা ফোল্ডার বানিয়েছি retrofit_get_post নামে। এর ভিতরে একটা ফাইল ক্রিয়েট করেছি server_side_code.php নাম দিয়ে। তার মানে আমি যদি কোন রিকোয়েস্ট এই ফাইলে পাঠাতে চাই তাহলে localhost এর URL এর পরে ফাইলের path-ও বলে দিতে হবে। লোকালহোস্টের URL RetrofitApiClient এ উল্লেখ করেছি। এখানে বাকি অংশটা উল্লেখ করলাম। POST method এর ক্ষেত্রে আমাদের রিকোয়েস্টটা যাবে মূলত **http://192.168.0.101/retrofit_get_post/server_side_code.php** অ্যাড্রেসে। POST, GET উভয় ক্ষেত্রেই একই রাউট দেখা যাচ্ছে। এর কারণ হচ্ছে আমি একটা PHP ফাইল দিয়েই উভয় রিকোয়েস্ট হ্যান্ডেল করেছি। আপনার পোস্ট, গেট এর ফাইল যদি আলাদা হয় তাহলে সেই অনুযায়ী রাউট বলে দিবেন।

POST request এর মাধ্যমে আমরা একজন ইউজারের ভ্যালিডিটি চেক করতে চাচ্ছি। তাই @POST annotation এর পরের লাইনে মেথডের নাম দিয়েছি getUserValidity(). এতে প্যারামিটার হিসেবে পাঠানো হয়েছে User ক্লাসের একটি অবজেক্টকে। সেটার শুরুতে উল্লেখ করতে হবে @Body annotation. এতো গেল পাঠানোর কথা। সার্ভার থেকে যেটা পাঠাবে সেটা রিসিভ করতে হবে না? সেটা রিসিভ করে কোন ক্লাস অনুযায়ী JSON parse করা লাগবে? ServerResponse ক্লাস অনুযায়ী তাই না? সেটাই বলা হয়েছে getUserValidity() এর শুরুতে return type হিসাবে।

দ্বিতীয় মেথডটা GET রিকোয়েস্টের জন্য। এতে প্যারামিটার হিসেবে কোন অবজেক্ট পাঠাচ্ছি না। শুধু একটা ইউজার আইডি পাঠালেই হচ্ছে তাই স্ট্রিং হিসেবে এটাকে পাঠিয়ে দিচ্ছি। @Query annotation এর ভিতরে “user_id” দিয়ে এরপর একটা স্ট্রিং এর অবজেক্ট userId উল্লেখ করা হয়েছে। এর মানে হচ্ছে userId স্ট্রিং অবজেক্টটা “user_id” এর সাথে map হবে। অর্থাৎ user_id হবে key আর userId হবে value. POST request এ URL এর সাথে ডেটা দৃশ্যমান থাকে না। কিন্তু GET request এ কিন্তু ডেটাগুলো URL এর সাথে যায়। আমার করা PHP কোড অনুযায়ী অ্যাপ থেকে গেট রিকোয়েস্টে হিট করতে হবে **http://192.168.0.101/retrofit_get_post/server_side_code.php?user_id=hasan** এই লিংকে। লক্ষ্য করেন, মূল লিংকের পরে **?user_id=hasan** যুক্ত হয়েছে। আমার সার্ভার সাইডের কোড user_id এর ভ্যালু hasan কিনা সেটা চেক করবে। কিন্তু প্রশ্ন হচ্ছে জাভা কোডের @GET annotation এর ভিতরে কিন্তু এই টাইপের কিছুই উল্লেখ নাই, তাহলে এই লিংকে হিট করবে কেমনে? উত্তর হচ্ছে (@Query(“user_id”) String userId), এটাই লিংকের শেষে “?” ও “user_id” যোগ করে সমান চিহ্ন দিয়ে এরপরে userId স্ট্রিং এর ভ্যালু বসায় দিবে।

@Body, @Query, @Path এরকম বেশ কয়েকটা annotation আছে। গুগল করে জেনে নিবেন প্লিজ।

উপরে আলোচনা করা হয়েছে এমন দুইটি API request নিয়ে যাদের base url আমরা RetrofitApiClient ক্লাসে বলে দিয়েছি। কিন্তু যদি কখনো ভিন্ন একটা URL এ হিট করার দরকার হয়? অর্থাৎ যেই URL টা আপনি রান টাইমে হিট করবেন, যার base url আগে assign করা url থেকে ভিন্ন তখন কী করবেন? সেটা করা খুব সহজ!

Retrofit Dynamic URL

```
...
@GET
Call<WeatherResponse> getWeatherUpdate(@Url String dynamicUrl);
...
```

এক্ষেত্রে @GET এর পরে route বলে দিতে হবে না। কারণ আপনি তো predefined base url এর কোনো রাউটে হিট করতে চাচ্ছেন না। তাই রাউটের অংশটুকু ফাঁকা থাকবে। আর আপনি যেই url এ হিট করতে চাচ্ছেন সেটা মেথডের প্যারামিটার হিসাবে পাঠাতে হবে। এটা যদি আবহাওয়ার আপডেট পাওয়ার কোনো end point হয় তাহলে আপনার dynamicUrl এর ভ্যালু হতে পারে **http://weather-update.com/api/dhaka/**. অর্থাৎ এই মেথডটি কল করার সময় আপনার সম্পূর্ণ URL-টাই পাঠিয়ে দিতে হবে।

Network call from MainActivity class

MainActivity তে Interface class এর একটা instance বানাতে হবে। ক্লাসের ডেটা মেম্বার হিসেবে উল্লেখ করা যায়ঃ

```
private ApiInterface apiInterface;
```

onCreate() এর ভিতরে initialize করতে পারেন এভাবেঃ

```
apiInterface = RetrofitApiClient.getClient().create(ApiInterface.class);
```

Login এর button click event থেকে এই মেথডটা কল করে পোস্ট রিকোয়েস্ট পাঠানো হচ্ছে ও রেসপন্স রিসিভ করা হচ্ছেঃ

```
private void checkUserValidity(User userCredential){

    Call<ServerResponse> call = apiInterface.getUserValidity(userCredential);

    call.enqueue(new Callback<ServerResponse>() {

        @Override
        public void onResponse(Call<ServerResponse> call, Response<ServerResponse> res

            ServerResponse validity = response.body();
            Toast.makeText(getApplicationContext(), validity.getMessage(), Toast.LENGT
        }

        @Override
        public void onFailure(Call call, Throwable t) {
            Log.e(TAG, t.toString());
        }
    });
}
```

সার্ভার থেকে রেসপন্স যদি ঠিকঠাক মত আসে তাহলে সেটা রিসিভ হবে onResponse() এর ভিতরে। কিন্তু যদি আপনার এখান থেকে কোন কারণে রিকোয়েস্ট সেন্ড না হয়, বা সার্ভারে কোন ঝামেলা থাকে। বা দুই দিকের data key না মিলে অথবা URL ভুল থাকে ইত্যাদি কারণে সার্ভারে রিকোয়েস্ট গিয়ে ‘সহি সালামতে’ ফেরত না আসলে সেটার ট্র্যাক পাবেন onFailure মেথডে। সার্ভার থেকে আসা রেসপন্সকে আমাদের ServeResponse ক্লাসের অবজেক্টে কনভার্ট করা হচ্ছে validity = resoponse.body() এই একটা মাত্র মেথড কল করার মাধ্যমে।

GET request এর জন্যেও প্রায় একই ভাবে কাজ করবে এই মেথডটিঃ

```java
private void getJokeFromServer(String userId) {

    Call<ServerResponse> call = apiInterface.getJoke(userId);

    call.enqueue(new Callback<ServerResponse>() {
        @Override
        public void onResponse(Call<ServerResponse> call, Response<ServerResponse> res
            ServerResponse validity = response.body();
            jokeTextView.setText(validity.getMessage());
        }

        @Override
        public void onFailure(Call<ServerResponse> call, Throwable t) {
            Log.e(TAG, t.toString());
        }
    });
}
```

Dynamic URL এ রিকোয়েস্ট পাঠানোর কোডঃ

Network call with a dynamic URL - Retrofit

```java
private void getWeather(String weatherUpdateUrl) {

    Call<WeathereResponse> call = apiInterface.getWeatherUpdate(weatherUpdateUrl);

    call.enqueue(new Callback<WeatherResponse>() {
        @Override
        public void onResponse(Call<WeatherResponse> call, Response<WeatherResponse> r
             WeatherResponse weatherResponse = response.body();
            // do something
        }

        @Override
        public void onFailure(Call<WeatherResponse> call, Throwable t) {
            Log.e(TAG, t.toString());
        }
    });
}
```

Server side PHP Code

আপনার যদি লাইভ সার্ভার না থাকে সেক্ষেত্রে Xampp/Wamp এর মাধ্যমে আপনার পিসিকেই সার্ভার হিসেবে ব্যবহার করে অ্যাপটা টেস্ট করতে পারবেন। Xampp/wamp ইন্সটলের পর নির্দিষ্ট ফোল্ডারে গিয়ে retrofit_get_post নামের একটা ফোল্ডার বানান। এর ভিতরে তৈরি করুন server_side_code.php নামের একটি php file. <**?php ?**> tag এর ভিতরে কপি-পেস্ট করুন নিচের কোডঃ