**Dagger Documentation**

1. **Module** : This is used on the class that does the work of constructing objects that’ll be eventually provided as dependencies
2. **Provides** : This is used on the methods inside the Module class that’ll return the object.
3. **Inject** : This is used upon a constructor, field or a method and indicates that dependency has been requested.
4. **Component** : The Module class doesn’t provide the dependency directly to the class that’s requesting it. For this, a Component interface is used that acts as a bridge between @Module and @Inject.
5. **Singleton** : This indicates that only a single instance of the dependency object would be created.
6. **Add dependency in build.gradle**

implementation 'com.google.dagger:dagger:2.28.3'  
annotationProcessor 'com.google.dagger:dagger-compiler:2.13'

### **Creating Modules**

1. **AppModule**

This module will provide the Context. You already know that we need Context everywhere, and in Retrofit as well we need the context object. And as the DI rule says we need an outsider to supply the objects, so here we will create this module that will give us the Context.

@Module  
 public class AppModule {  
 private Application mApplication;  
  
 public AppModule(Application mApplication) {  
 this.mApplication = mApplication;  
 }  
  
 @Provides  
 @Singleton  
 Application provideApplication() {  
 return mApplication;  
 }  
 }

1. **NetWork Module**

We need many objects in this Module. You might already know that for Retrofit fit we need a bunch of things.  
We need **Cache, Gson, OkHttpClient** and the Retrofit itself. So we will define the providers for these objects here in this module.

@Module  
 public class NetworkModule {  
 String baseUrl;  
  
 public NetworkModule(String baseUrl) {  
 this.baseUrl = baseUrl;  
 }  
  
 @Provides  
 @Singleton  
 Gson provideGson() {  
 GsonBuilder gsonBuilder = new GsonBuilder();  
 gsonBuilder.setFieldNamingPolicy

(FieldNamingPolicy.*LOWER\_CASE\_WITH\_UNDERSCORES*);  
 return gsonBuilder.create();  
 }  
  
 @Provides  
 @Singleton  
 Retrofit provideRetrofit(Gson gson) {  
 return new Retrofit.Builder()  
 .addConverterFactory(GsonConverterFactory.*create*(gson))  
 .baseUrl(baseUrl)  
 .build();  
 }  
  
 }

### **3.Building Component**

@Singleton  
 @Component(modules = {AppModule.class, NetworkModule.class})  
 public interface AppComponent {  
 void inject(MainActivity mainActivity);  
 }

So you can see we will inject in the**MainActivity.** We also define

all the modules using the @Component annotation as you can see in the code.

Now create a class named **BaseApplication**. In this class we will build the ApiComponent.

private AppComponent appComponent;  
  
 @Override  
 public void onCreate() {  
 super.onCreate();  
  
 appComponent = DaggerAppComponent.*builder*()  
 .appModule(new AppModule(this))  
 .networkModule(new NetworkModule(“URL”))  
 .build();  
 }  
  
public AppComponent getNetworkComponent() {  
 return appComponent;  
}

we have our API component, but we need to instantiate this class when our application launches. And for this, we need to define it inside our App Manifest file. So open your **AndroidManifest.xml** and modify it as shown below

<application  
 android:name=".BaseApplication"

.

.

.

</application>

Now finally, we can inject the dependency.

* Come inside **MainActivity.java** and modify it as below

((BaseApplication) getApplication()).getNetworkComponent().inject(this);