# Download Android Studio

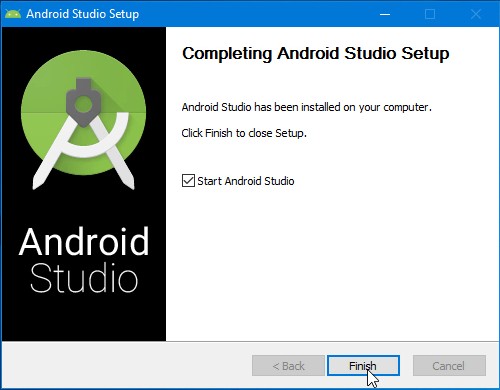
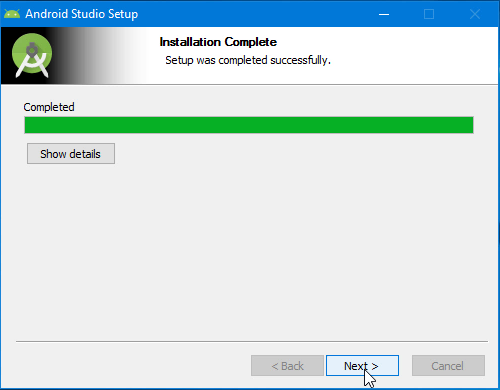
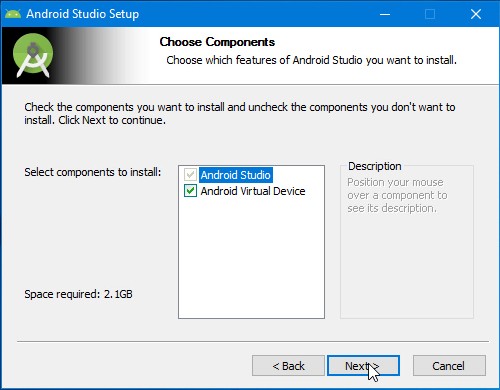
<https://developer.android.com/studio/index.html>

Click “Download Android Studio”, accept the terms and conditions, and proceed with the download.

The download might take a while, as it is about 700 MB. Run the installer after the download is finished.

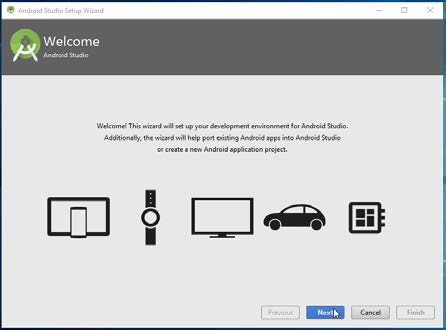
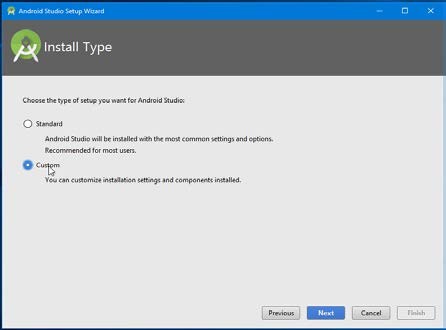
# Install Android Studio

When running the installer, we recommend using the default values, as shown below. Once installed, Android Studio will run, and launch a setup wizard to download additional components.



# Run Android Studio Setup Wizard

When you first start Android Studio, it will offer to import any previous settings. Since this is likely your first installation, there are no settings to import. Again, just accept the default option and continue.



Next, select the UI theme. Choose whichever one you prefer -- this is simply a matter of preference. On the next screen, select the option to install an Android Virtual Device. You will need an actual Android device for this course, but this virtual device can be useful.

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

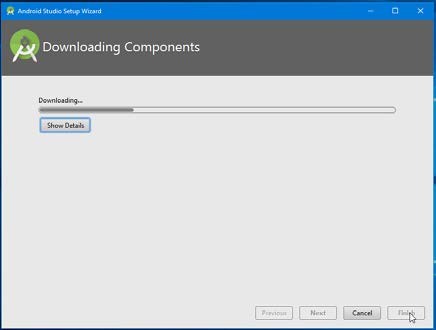
Accept the recommended RAM size for the emulator, and click “Next” to see a summary of the components to be downloaded. The additional components total about 1.8 GB in size.

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

When the download begins, the progress bar might appear to freeze, but do not worry -- the download is progressing. Once installation is complete, click “Finish”.

Graphical user interface, text, application, Word

Description automatically generated

# Begin Developing

The Create New Project appears. The first option is to select the type of activity. Choose Empty, which is the default and click NexGraphical user interface, text, application

Description automatically generatedt.

Chart

Description automatically generated

Configure your project is the next screen. It will ask for the name of the Application, Package Name, Project path, language, and API Level. Keep the defaults and click on Finish

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | Graphical user interface, application  Description automatically generated | Graphical user interface, text, application  Description automatically generated | |

Wait for it to finish. Once everything is downloaded and installed, the new project is created and you are taken to the **Android workspace**.

**Create New Virtual Device**

If you are starting the AVD Manager for the first time, you will see the following screen.  Else you will see the list of AVDs created.

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated | Graphical user interface, application  Description automatically generated |

In the left-hand panel displays a list of the Category of the device.  It includes TV, Phone, Wear & Tablet. Select the category.

The middle pane displays the list of devices available. Select one based on the requirement of your app. After this click on the Next button.

Note that phones with larger resolution Choose the pixels resolution according to your requirements as it will take huge RAM in large pixels resolution device. If your computer has low RAM, then prefer to choose less resolution device. Click Next to continue

|  |  |
| --- | --- |
| Graphical user interface, table  Description automatically generated | Graphical user interface, application  Description automatically generated |

Choose the system image based on the API level targeted by your App. The app won’t run if you choose lower API than the one target by the App,Select the image and click on Next to continue.

Here you can name your AVD, change startup orientation and few other hardware properties. **Click on Show Advanced Settings** to show more settings.

Click on Finish to create the AVD.

Graphical user interface, application

Description automatically generated

Under the action column, click on the Start Android Virtual Device icon to run the AVD. The Android Emulator uses the AVD to mimic the device.  You can then use the control panel to manage the device. The Extend control button at the bottom gives you more options

**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);