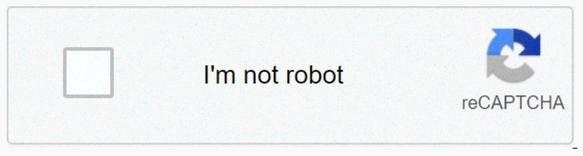

Continue

Android studio project sample

Android Studio makes it easy to create Android apps for a variety of sizes, such as phones, tablets, TVs, and Wear devices. This page shows you how to run a new Android app project or import an existing project. If you don't have a project open, Android Studio shows you a welcome screen where you can create a new project by clicking Start a new Android Studio project. If you have a project open, you can start creating a new project by selecting File > New > New Project from the main menu. You should then see the Create a New Project wizard, which lets you select the type of project that you want to create and fills in the code and resources to get started. This page walks you through creating a new project by using the Create New Project wizard. Select project On the Select Project screen that appears, you can select the type of project you want to create from the device size categories that appear as tabs at the top of the wizard. For example, Figure 1 shows a project with basic Android activity for the selected phone and tablet. Figure 1. On the first screen of the wizard, select the type of project you want to create. By choosing the type of project you want to create, Android Studio can include sample code and resources to help you get started. Once you've made your selection, click Next. Set up a project The next step is to configure some settings and create a new project, as described below and shown in Figure 2. If you're creating a native C++ project, you can learn more about the options that you need to configure by reading Create a new C/C++ enabled project. Figure 2. Set up a new project with several settings. Specify the name of the project. Specify the name of the package. By default, this package name also becomes an application ID that you can change later. Specify the save location where you want to store the project locally. Select the language that you want Android Studio to use when you create sample code for the new project. Note that you are not limited to using only this project creation language. Select the minimum API level that you want your application to support. When you select a lower API level, your app can rely on fewer modern Android APIs. However, a larger percentage of Android devices are able to run the app. The opposite is true when selecting a higher-level API. If you want to see more data to help you decide, click Help Me Choose. If you want your project to use AndroidX libraries by default, which are improved replacements for Android support libraries, check the box next to Use AndroidX artifacts. To learn more, read the AndroidX overview. When you are ready to create a project, Finish button Android Studio creates a new project with some basic code and resources to get started. If you later decide to add support for a different device format, you can add the module to the project later. And if you want to share code and resources between modules, you can do so, in your Android library. For more information about the Android project structure and module types, read the project overview. If you're new to Android, start by getting started on Android. Import an existing project To import an existing local project into Android Studio, proceed as follows: Click File > New Project > Import. In the window that appears, browse to the root of the project that you want to import. Click OK. Android Studio then opens the project in a new IDE window and indexes its contents. If the project is imported from version control, use the File > New > Project menu from version control. For more information about importing projects from version control, read IntelliJ VCS-specific procedures. If you're importing an existing Eclipse ADT project into Android Studio, how you add a project depends on its structure. To learn more about importing projects from Eclipse, see Migrate from Eclipse. Google is committed to developing racial equality for black communities. See how. Welcome to android developer code samples. Here you can browse the sample code and learn how to create different components for your application. Use the categories on the left to view the available examples. Each example is a fully functioning Android app. You can view resources, source files, and see the overall structure of the project. You can copy and paste the code you need, and if you want to share a link to a specific row, you can double-click it to get the URL. Import samples from GitHub Android Studio provides easy access to import Android code samples from GitHub and is the recommended method for downloading Android code samples. To import the sample code into Android Studio: On the Android Studio menu, choose File > Import Example to open the Import Sample wizard. Select the example that you want to import, and then click Next. Specify the name of the application and the location of the project if it differs from the displayed settings. Click Finish. The sample project opens in a new Android Studio project. Note: When you start Android Studio, you can also select Import sample Android code in the Welcome to Android Studio wizard to import the sample project from gitHub as a new project. For more information about importing samples, see Easy access to Android code samples on GitHub. Get samples Although importing samples from Android Studio is the recommended method, you can also use the categories on the left to view the available samples and learn how to create different components for your app. If download the complete project, simply click any source file in the project and click the Download link in the upper right corner of the source page. To import a downloaded project: Unpack the downloaded project package. In Android Studio, select File > Import Project and select the root folder for the unzipped project. Android Studio may ask you to select the type of project you are importing. In this case, select Import Import from the external model and select Gradle. Note: When you start Android Studio, you can also select Import Non-Android Studio project in the Welcome to Android Studio wizard to import the sample project that you downloaded. Note: At this time, downloadable projects are intended for use with Gradle and Android Studio. Let's start actual programming with the Android Framework. Before you start writing your first example using the Android SDK, you should make sure that you have set up your Android development environment correctly, as explained in the Android - Configuration Environment tutorial. I also assume that you have some working knowledge from android studio. So let's move on to writing a simple Android app that will print Hello World!. Create an Android app The first step is to create a simple Android app using Android studio. When you click on the Android studio icon, it will show the screen as shown below You can start creating apps by calling start a new android studio project. in the new installation frame, ask the name of the application, the package information and the location of the project.− After entering the name of the application, it will be called select the aspect ratios on which the application runs, here you need to specify the minimum SDK, in our tutorial I declared as API23: Android 6.0(Mashmallow) − The next level of installation should include a selection of activities on mobile devices, specifies the default layout of the application. At the last stage, you will have an open development tool for writing application code. Anatomy of android applications Before running the application, be aware of several directories and files in the Android project - Sr.No. Folder, File & Description 1 Java Contains .java source files for the project. By default, it contains mainactivity.java a source file with an activity class that runs when the application starts using the application icon. 2 os/drawable-hdpi is a directory for drawn objects that are designed for high-density screens. 3 res/layout Is a directory for files that define the user interface of an application. 4 res/values Is a directory for other different XML files that contain a collection of resources such as strings and color definitions. 5 AndroidManifest.xml is a manifest file that describes the basic characteristics of an application and defines each of its components. 6 Build.gradle This is an auto-generated file that contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode, and versionName The following section gives you a brief overview of important application files. The main activity file The main activity code is the Java MainActivity.java. This is the actual application file that will eventually be converted to the Dalvik executable file and run the application. Below is the default code generated by the application wizard for Hello application − com.example.helloworld package; Import Import import android.os.Bundle; MainActivity public class extends AppCompatActivity { @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); } } Here, the R.layout.activity_main refers to the activity_main.xml file in the res/layout folder. The OnCreate() method is one of many methods that are evaluated when an activity is loaded. Manifest File Regardless of the component that you are developing as part of your application, you must declare all its components in the manifest.xml which is located at the root of the application project directory. This file acts as an interface between the Android operating system and the application, so if you do not declare a component in this file, then it will not be included by the operating system. For example, the default manifest file will look like the following file − <?xml version=1.0 encoding=utf-8?>Here ... Tags included components associated with the application. <application></application> <manifest xmlns:android= package=com.example.tutorialspoint7.myapplication> <application android:allowbackup=true android:icon=@mipmap/ic_launcher android:label=@string/app_name android:supportsrtl=true android:theme=@style/AppTheme> <activity android:name=.MainActivity> <intent-filter> <action android:name=android.intent.action.MAIN></action> <category android:name=android.intent.category.LAUNCHER></category> </intent-filter> </activity> </application> </manifest> Android attribute:icon point to the application icon available under os/drawable-hdpi. The application uses an image named ic_launcher.png in folders to draw <activity>Tag to specify the activity, and the android:name attribute specifies the fully qualified name of the activity subclass class, and the android:label attributes specify the string to be used as the activity label. You can use tags to specify multiple <activity>activities. The intent filter action is called android.intent.action.MAIN to indicate that this activity serves as an entry point for the application. The intent filter category is called android.intent.category.LAUNCHER to indicate that the application can be started from the device startup icon. @string refers to the string file .xml explained below. Therefore, the @string/app_name refers to the app_name string defined in the string .xml that is HelloWorld. Similarly, other strings are populated in the application. Below is a list of tags that will be used in the manifest file to specify the different components of the Android application − <activity>items for activity items <service>for service items for <receiver>receiver items <provider>broadcast for content providers Strings Strings Strings .xml the file is located in the res/values folder and contains all the text przez aplikację. Na przykład nazwy przycisków, etykiet, tekstu domyślnego i podobnych typów</provider> </receiver> </service> </activity> </activity> </activity> </activity> go to this file. This file is responsible for their text content. For example, the default string file will look like the following file − <resources> <string name=app_name>HelloWorld</string> <string name=hello_world>Hello world!</string> <string name=menu_settings>Settings</string> <string name=title_activity_main>MainActivity</string> </resources> Layout file activity_main.xml is a layout file available in the res/layout directory referenced by the application when creating the interface. This file will modify very often to change the layout of the application. For Hello World! this file will have the following content related to the default layout − <RelativeLayout xmlns:android= xmlns:tools= tools:layout_width=match_parent android:layout_height=match_parent> <TextView android:layout_width=wrap_content android:layout_height=wrap_content android:layout_centerhorizontal=true android:layout_centervertical=true android:padding=@dimen/padding_medium android:text=@string/hello_world tools:context=. MainActivity></TextView> </RelativeLayout> This is an example of a simple RelativeLayout that we will study in a separate chapter. TextView is an android control used to build a GUI and has various attributes such as android: layout_width, android: layout_height, etc., which are used to set its width and height etc. @string refers to a string .xml located in the res/values folder. Therefore, the @string/hello_world refers to the hello string defined in the strings.xml file that is Hello World!. Let's launch the app Let's try to launch our Hello World! application we have just created. I assume you created AVD while executing the configuration environment. To launch an app from an Android studio, open one of the project activity files and click the Run icon on the toolbar. Android studio installs the application on AVD and runs it, and if everything is fine with the configuration and application, it will appear after the Emulator window − Congratulations!!! You developed your first Android app, and now just follow the rest of the step-by-step tutorial to become a great Android developer. Happy Birthday. Best.