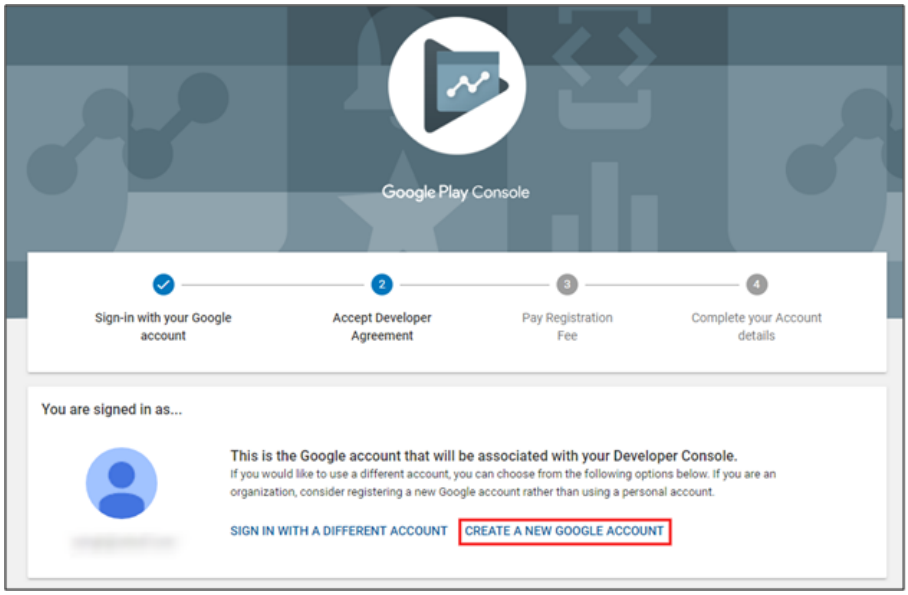
# UNIT -4

## ❖ Publish Android App on the Play Store

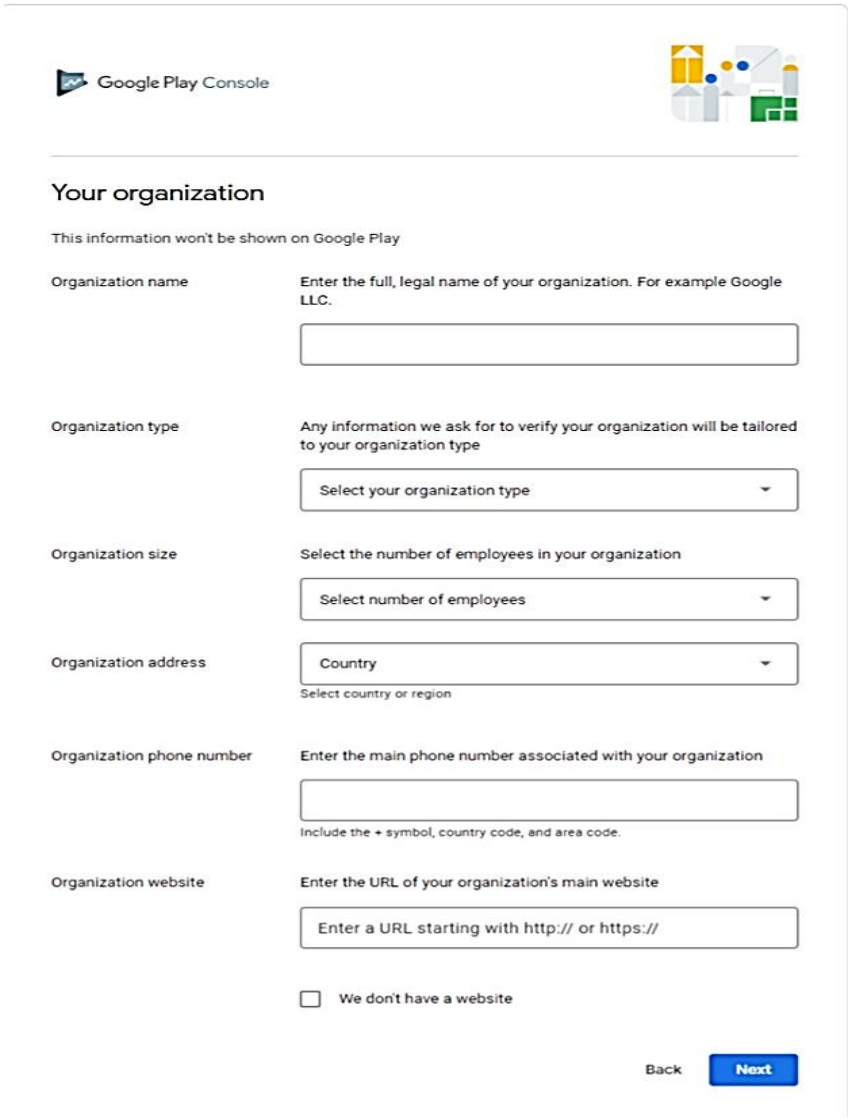
### Step 1 – Google Play Console Developer Account.

One of the primary steps of publishing an Android application on the Play Store is creating a Google developer account on the Google Play Console. Google Play Console can be understood as a backend operating system for all the apps published on the Play Store. Developers who intend to publish an app on the Play Store must create a developer account on the console and pay a one-time fee of $25, payable using a credit card or through online banking methods. Also, remember that post submission, the developer account can take up to 48-hours to activate.




.

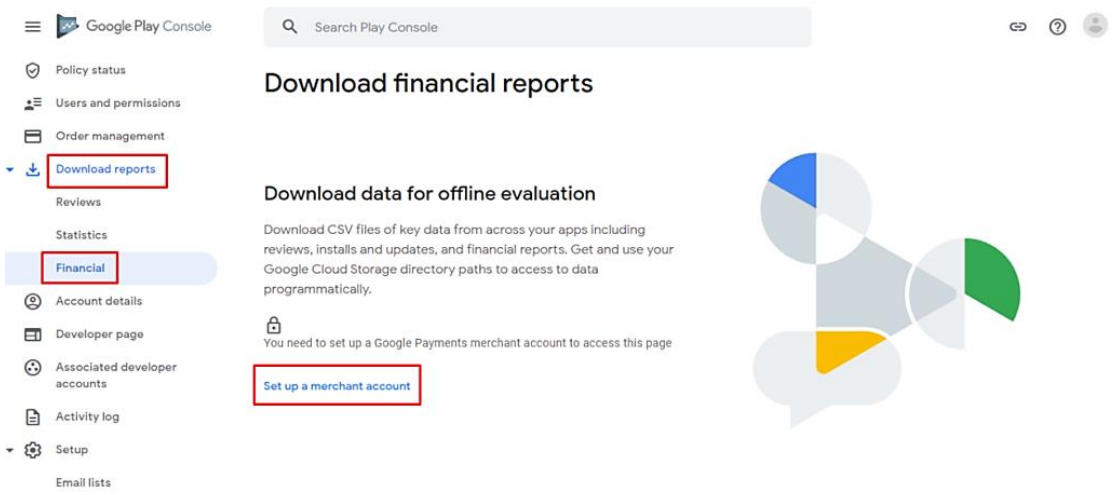
Finally, ensure you fill out all the credentials asked while creating the account.




## Step 2 – Set-up up a Google Merchant Account.

If your app involves in-app purchases, the next essential step is to link your developer account with your merchant account. If you already have an existing merchant account, you can navigate to Download reports and select Financial. However, to access this page, you must create a Google Merchant Account.

And to create one, click on setup up a merchant account.

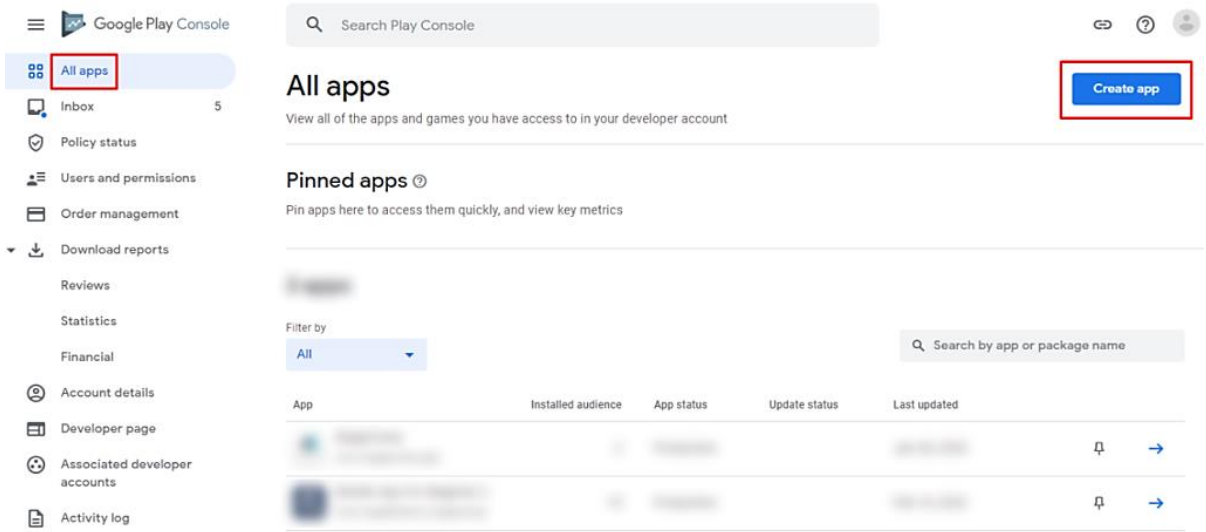



Once you create your new merchant account, it will automatically get linked to your Google Play Console account and enable you to monitor, manage, and analyze the app sales and generate reports.

### Step 3 – Create Application

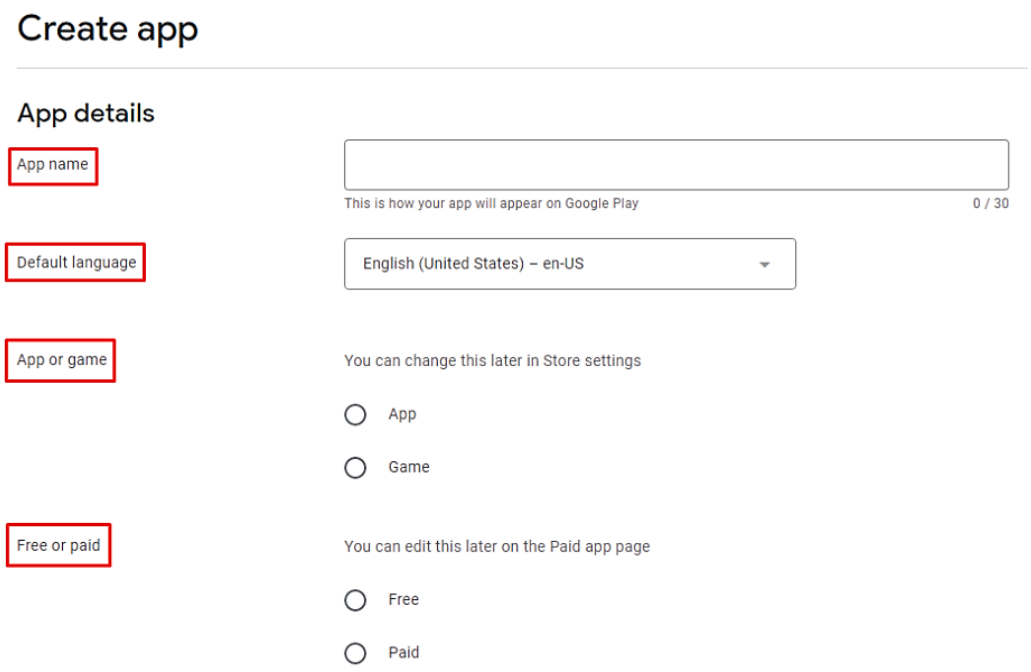
Once the merchant account is linked to your Google Play Console, the next step is to create an application. And for creating an application, there are a few essential steps that you need to follow:

- Click on – Menu > All applications
- Select the 'Create Application' option.




Next, the play console will ask you to enter some basic app details. For instance,

- App Name – You must enter a 30-character long name in this field which will be displayed on the Google Play Console. However, this app name can be changed afterward.
- Default language – Another essential field is the setup of the app language. You can navigate to the drop-down menu and set a default language for your app.
- App or game – The next step is to define whether you upload an app or a game, but this can again be revised afterward from the store settings.
- Free or paid – Define whether your app will be available free of cost or will require the user to pay for it. The free or paid section can be updated from the Paid app page later, but only until you publish your app. Once the app is live, you cannot transform your app from free to paid.

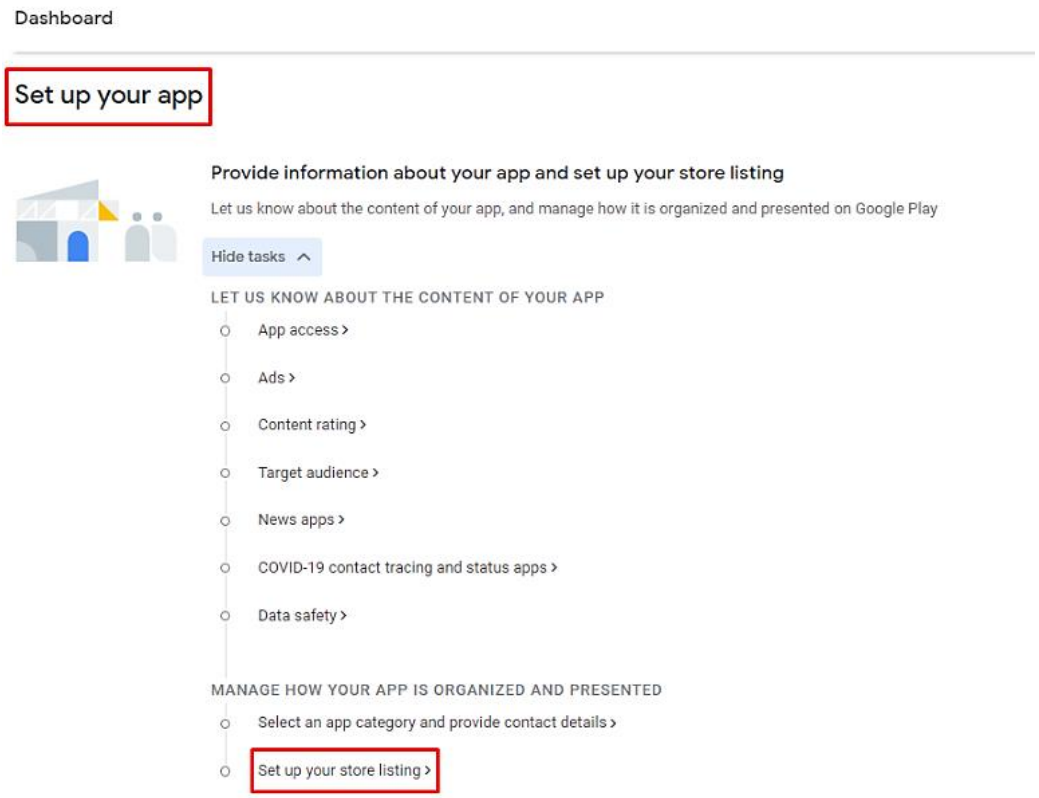



Once all of the above information is filled and verified, the Google Play Console will enquire for affirmations from you. Ensure that your app matches the Google policies of the Developer Program and Accepts US export laws. As soon as you agree to the terms and conditions, click **Create App**.

### Step 4. App Store Listing Details

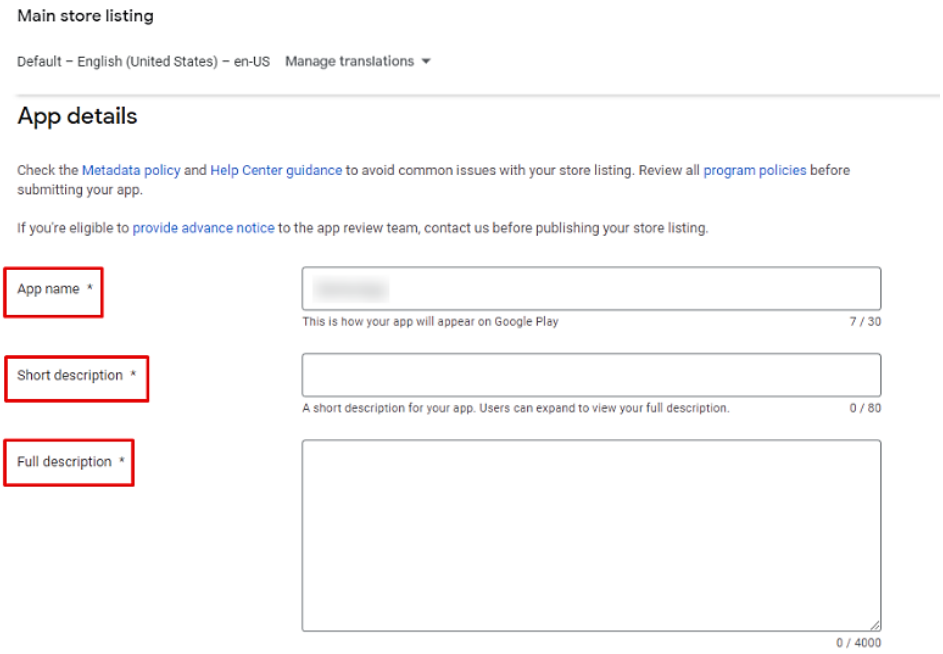
The next step to uploading the app on the Google Play console is filling in the essential information regarding the application listing. Once you click on the 'Create

App,' the Play Console will automatically take you to a consolidated dashboard wherein you'll have to enter the necessary details to set up your app.




The Google Play Console will enquire about the following:

- App name – As you have already entered the app name in the previous step, you need not enter the same name again; however, if you wish to revise the title, this is where you can change the name.
- Short description – This field requires you to enter an 80-character-long description that best describes your app.
- Full description – The following field to the short description enables you to explain your app in detail. You can expand the word limit to 4000 characters and leverage your targeted keywords to lead Google to share your app with the relevant audience.




Once all this information is added to the Google console, the next step is adding the app graphics, category of the app, and the privacy policy. Remember, we asked you to keep high-quality images ready before beginning the app publishing process; this is precisely where all those images will be leveraged.

Further, here are the details you would require:

| **Particulars** | **Details** |
|---|---|
| Screenshots | -2 to 8 in number, JPG, or PNG. The ratio shouldn't exceed 2:1 |
| Icon | – 512 X 512– PNG– Maximum file size: 1024KB |
| Localization | -If your app comes in several languages, you need to mention them and add additional translations of your app's information to appeal to the users coming and checking out your app. |
| Application type and categorization | – Navigate to the Drop-down menu and select application type – game or app.- Pick a category suitable for your app- Rate your content after uploading your APK. |
| Contact details | -Provide the necessary contact forms so users can contact you. |
| Privacy Policy | -To escape the breach of app privacy, Google mandates adding a privacy policy while publishing your app. -If you need a break, click Save Draft and |

| | |
|---|---|
| | complete it later. |

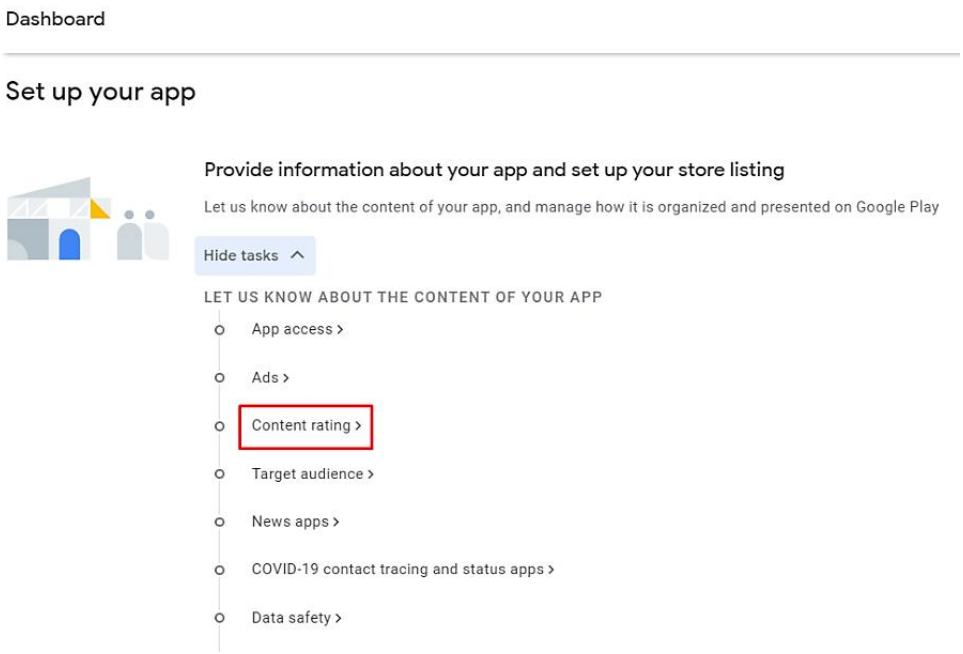
Once you are done uploading details,  Hit the Save button.

**Also, Read- Mobile App Security: A Comprehensive Guide to Secure Your Apps**

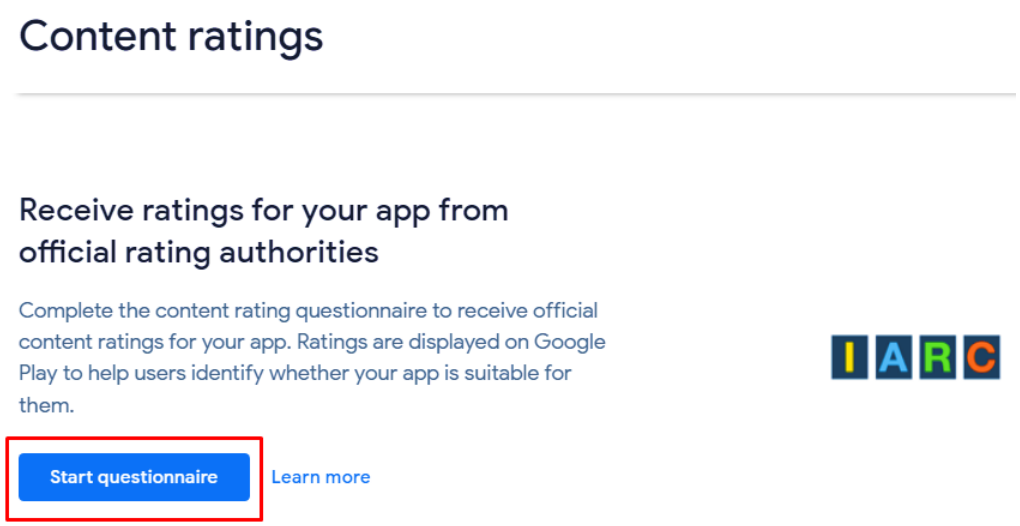
## Step 5 – Content Rating

The next most crucial step is the content rating questionnaire. Without rating your app's content, Google will consider it an Underrated app and will certainly remove it from Google Play Store. And since you might not want it, let's learn the steps of adding a content rating.

To add the content rating, you'll have to navigate to the main dashboard, set up your app, and select the Content rating option.

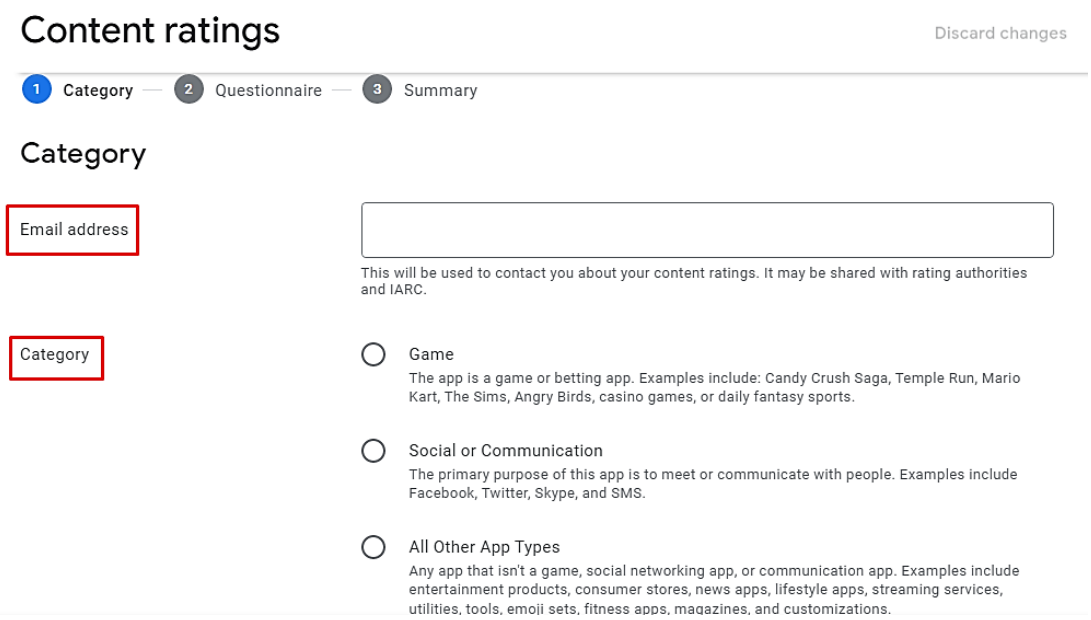



The Next dashboard will pop up, and you'll be able to navigate the "Start Questionnaire" button; you have to click the tab and get started.

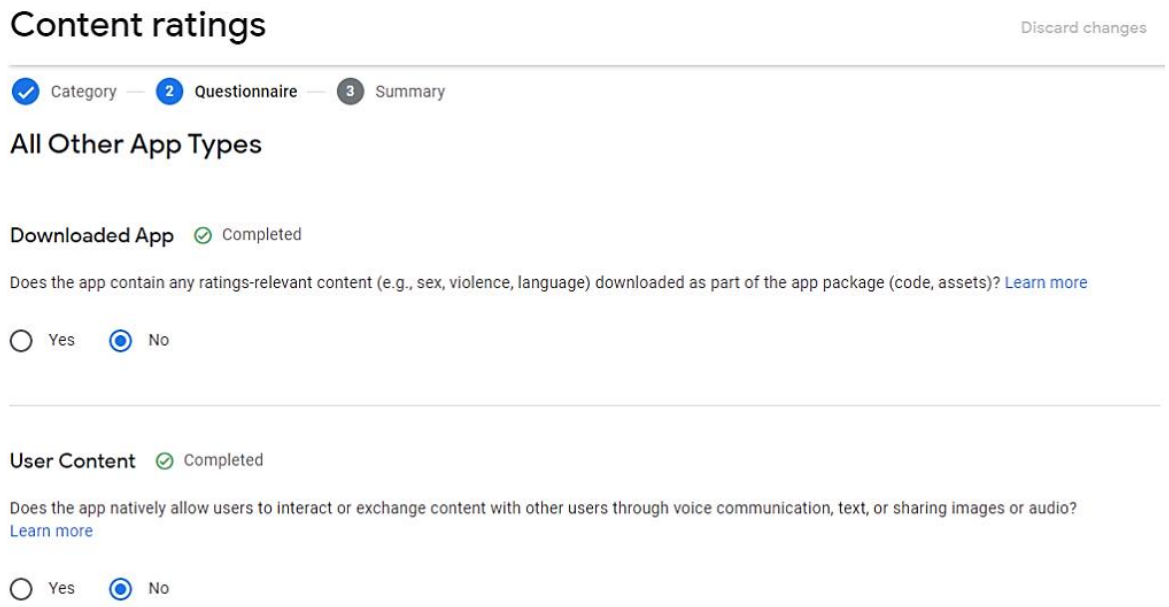



You'll have to enter basic information about your app in the content rating section. This section is divided into three sub-sections – Categories, Questionnaire, and Summary.

In the Category section, you have to enter the email address that the users can leverage to contact you and the category of the app you are publishing on the Play Store.




Once you are done filing the above fields, click the Next button, and you'll be redirected to the questionnaire section. The questionnaire section lets Google explore more about your app to understand your target audience better.

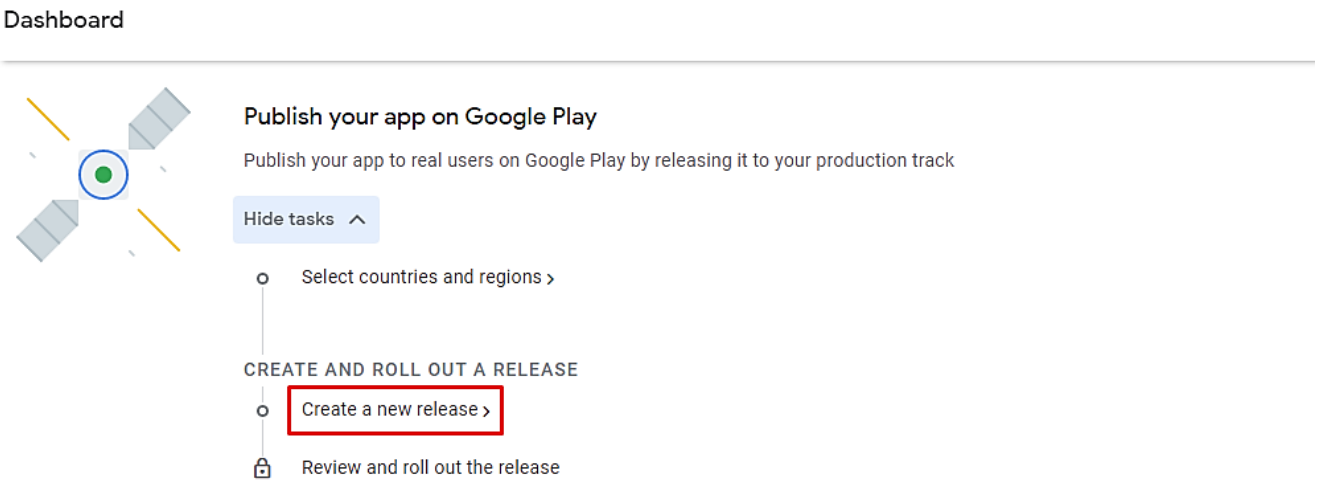



Once all the details are filled in, you can look at the content rating summary and hit 'Submit' to apply the changes.

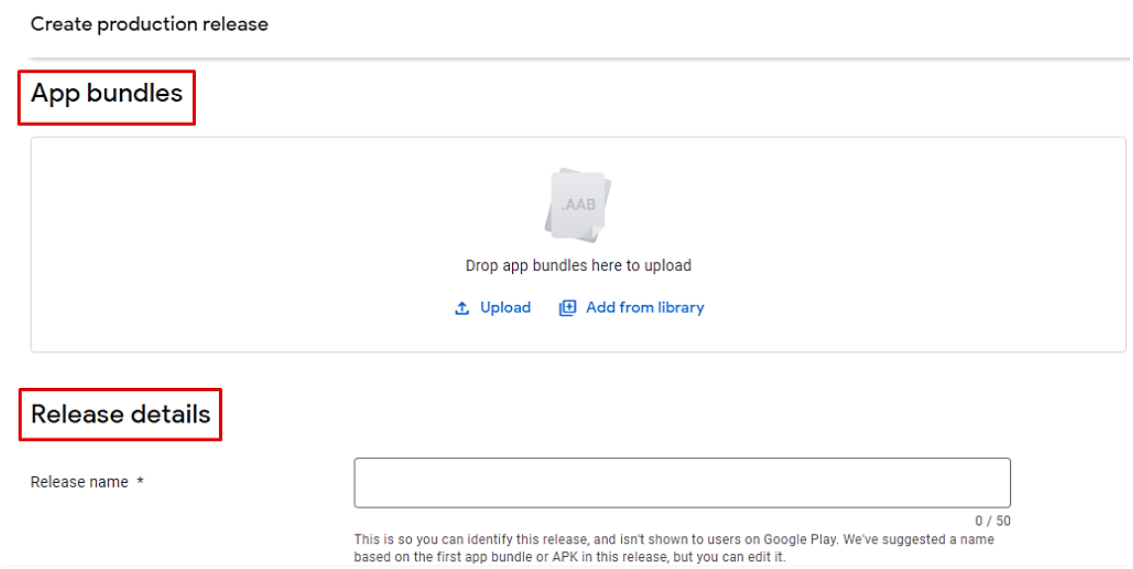
### Step 6 – Create & Upload Android App to Google Play

Uploading the APK to the Google Play Console is the foremost step of the app publishing process, where the app is finally uploaded and submitted for Google to review and go live.

Once you have decided on the testing, you can go to the dashboard and select "**Create a new release.**"



Post selecting "**Create a new release**," you'll be redirected to a dashboard, wherein you will upload the app bundles and the release details.




Once you enter all the details, confirm that everything is correct, and take the last step of this guide on Google Play app upload and add the application to the platform. Then, navigate to the 'App Releases' tab and select 'Manage Production' followed by 'Edit Release'; click on 'Review' and then the 'Start rollout to production option.

Select the 'Confirm' option, and that's it!

You are done with the successful upload of your app to the Google Play Store account for free.

Once the Google Play app upload is done, you must patiently wait for your application to get reviewed and approved by Google. The Google app approval process can take a few hours or extend up to 7days, so ensure that you are well prepared for both successful publishing of the app or for doing any revisions if required.

## ❖ Android Shared Preferences

Shared Preferences allows activities and applications to keep preferences, in the form of key-value pairs similar to a Map that will persist even when the user closes the application. Android stores Shared Preferences settings as XML file in **shared_prefs** folder under DATA/data/{application package} directory. The DATA folder can be obtained by calling Environment.getDataDirectory(). **SharedPreferences** is application specific, i.e. the data is lost on performing one of the following options:

- on uninstalling the application
- on clearing the application data (through Settings)

As the name suggests, the primary purpose is to store user-specified configuration details, such as user specific settings, keeping the user logged into the application. To get access to the preferences, we have three APIs to choose from:

- **getPreferences()** : used from within your Activity, to access activity-specific preferences
- **getSharedPreferences()** : used from within your Activity (or other application Context), to access application-level preferences
- **getDefaultSharedPreferences()** : used on the PreferenceManager, to get the shared preferences that work in concert with Android's overall preference framework

In this topic we'll go with getSharedPreferences(). The method is defined as follows: getSharedPreferences (String PREFS_NAME, int mode) **PREFS_NAME** is the name of the file. **mode** is the operating mode. Following are the operating modes applicable:

- **MODE_PRIVATE**: the default mode, where the created file can only be accessed by the calling application
- **MODE_WORLD_READABLE**: Creating world-readable files is very dangerous, and likely to cause security holes in applications
- **MODE_WORLD_WRITEABLE**: Creating world-writable files is very dangerous, and likely to cause security holes in applications
- **MODE_MULTI_PROCESS**: This method will check for modification of preferences even if the Shared Preference instance has already been loaded
- **MODE_APPEND**: This will append the new preferences with the already existing preferences
- **MODE_ENABLE_WRITE_AHEAD_LOGGING**: Database open flag. When it is set, it would enable write ahead logging by default

The activity_main.xml layout consists of two EditText views which store and display name and email. The three buttons implement their respective onClicks in the MainActivity.

```
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >

    <Button
        android:id="@+id/btnSave"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:onClick="Save"
        android:text="Save" />

    <Button
        android:id="@+id/btnRetr"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:onClick="Get"
        android:text="Retrieve" />

    <Button
        android:id="@+id/btnClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/etEmail"
        android:layout_centerVertical="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:onClick="clear"
        android:text="Clear" />

    <EditText
        android:id="@+id/etEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Email"
        android:inputType="textEmailAddress"
        android:layout_below="@+id/etName"
```

```
        android:layout_marginTop="20dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <EditText
        android:id="@+id/etName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Name"
        android:inputType="text"
        android:layout_alignParentTop="true"
        android:layout_alignLeft="@+id/etEmail"
        android:layout_alignStart="@+id/etEmail" />

</RelativeLayout>
```

```
package com.example.sharedpreferences;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {
    SharedPreferences sharedpreferences;
    TextView name;
    TextView email;
    public static final String mypreference = "mypref";
    public static final String Name = "nameKey";
    public static final String Email = "emailKey";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        sharedpreferences = getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);
        if (sharedpreferences.contains(Name)) {
            name.setText(sharedpreferences.getString(Name, ""));
        }
        if (sharedpreferences.contains(Email)) {
            email.setText(sharedpreferences.getString(Email, ""));

        }

    }

    public void Save(View view) {
        String n = name.getText().toString();
        String e = email.getText().toString();
        SharedPreferences.Editor editor = sharedpreferences.edit();
```

```
        editor.putString(Name, n);
        editor.putString(Email, e);
        editor.commit();
    }

    public void clear(View view) {
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        name.setText("");
        email.setText("");

    }

    public void Get(View view) {
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        sharedpreferences = getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);

        if (sharedpreferences.contains(Name)) {
            name.setText(sharedpreferences.getString(Name, ""));
        }
        if (sharedpreferences.contains(Email)) {
            email.setText(sharedpreferences.getString(Email, ""));

        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

}
```

## ❖ Managing application resources in a hierarchy

Managing application resources in a hierarchy means organizing your app's assets like images, layouts, strings, and other data in a structured tree-like structure, where each resource is categorized and nested within parent categories, allowing for easier access, organization, and management, often with the ability to inherit properties from higher levels in the hierarchy; essentially mirroring a file system structure within your project, with different folders representing different resource types and their variations based on device configurations.

Key points about managing application resources hierarchically:

- **Centralized location:**

All app resources are typically stored within a dedicated directory within your project, like the "res" folder in Android development.

- **Resource types:**

  Different types of resources are separated into distinct subfolders within the main resource directory, such as "drawable" for images, "layout" for screen layouts, "values" for strings and dimensions.

- **Qualifiers:**

  You can further categorize resources by adding qualifiers to the filenames, allowing you to provide different versions of a resource based on device factors like screen density, orientation, or language.

- **Inheritance and overriding:**

  In some systems, settings or access controls defined at a higher level in the hierarchy can be inherited by child resources, allowing for consistent application behavior across different parts of your app while still enabling customization at lower levels.
  Example of a resource hierarchy in Android development:


- **"res" directory:**
  - **"drawable":** Contains image files for different screen densities (e.g., "logo.png", "logo_hdpi.png")
  - **"layout":** Holds layout files for different screen sizes (e.g., "activity_main.xml", "activity_main_tablet.xml")
  - **"values":** Contains default values for strings, dimensions, and styles (e.g., "strings.xml", "dimens.xml")
  - **"mipmap":** Used for app icons that are optimized for different launcher densities

**Benefits of hierarchical resource management:**

- **Organization:**

  Keeps your app resources well-structured and easy to find

- **Maintainability:**

Simplifies updating resources across different parts of your app by making changes at higher levels in the hierarchy

- **Device compatibility:**

  Allows you to provide tailored resources for different device configurations using qualifiers

- **Access control:**

  In some systems, you can set permissions at different levels in the hierarchy to manage who can access specific resources

## ❖ Working with Different Types of Resources in Android

In Android, resources are external elements such as strings, images, colors, and layouts that help manage UI components efficiently. They allow for better maintainability, localization, and adaptability across different screen sizes and configurations.

### Types of Resources in Android

### 1.Drawable Resources

- Used for images, vector graphics, and XML-based shapes.
- Stored in res/drawable/.
- **Examples:** PNG, JPG, SVG, and XML-based drawables.

#### 🔻 Example of a Vector Drawable (res/drawable/ic_launcher.xml)

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
   android:width="24dp"
   android:height="24dp"
   android:viewportWidth="24"
   android:viewportHeight="24">
   <path
      android:fillColor="#FF0000"
      android:pathData="M12,2L15,8H9L12,2Z" />
</vector>
```

### 2 Layout Resources

- Define the structure of the user interface.
- Stored in res/layout/.

### 🔻 Example (res/layout/activity_main.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"/>
</LinearLayout>
```

## 3 String Resources

- Used for defining text values (useful for localization).
- Stored in res/values/strings.xml.

### 🔻 Example (res/values/strings.xml)

```
<resources>
    <string name="app_name">MyApplication</string>
    <string name="hello_world">Hello, World!</string>
</resources>
```

### 🔻 Accessing String Resources

- In XML: android:text="@string/hello_world"
- In Java/Kotlin:

## 4 Color Resources

- Used to define color values.
- Stored in res/values/colors.xml.

### 🔻 Example (res/values/colors.xml)

```
<resources>
    <color name="primaryColor">#6200EE</color>
    <color name="accentColor">#03DAC5</color>
</resources>
```

### 🔻 Using Color in XML

```
<TextView
   android:textColor="@color/primaryColor"/>
```

## 5 Dimension Resources

- Used for defining sizes, margins, and paddings.
- Stored in res/values/dimens.xml.

### 🔹 Example (res/values/dimens.xml)

```
<resources>
   <dimen name="text_size">16sp</dimen>
   <dimen name="margin">8dp</dimen>
</resources>
```

### 🔹 Using in XML

```
<TextView
   android:textSize="@dimen/text_size"
   android:layout_margin="@dimen/margin"/>
```

## 6 Style and Theme Resources

- Define UI consistency across the app.
- Stored in res/values/styles.xml and res/values/themes.xml.

### 🔹 Example (res/values/styles.xml)

```
<style name="CustomButton" parent="Widget.MaterialComponents.Button">
   <item name="android:background">@color/primaryColor</item>
   <item name="android:textColor">@color/white</item>
</style>
```

### 🔹 Applying Style to a Button in XML

```
<Button
   style="@style/CustomButton"
   android:text="Click Me"/>
```

## 7 Menu Resources

- Defines app menus (e.g., options menu, context menu).
- Stored in res/menu/.

### 🔹 Example (res/menu/main_menu.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/action_settings"
      android:title="@string/settings"
      android:icon="@drawable/ic_settings"
      android:showAsAction="always"/>
</menu>
```

## 8 Animation Resources

- Define animations using XML.
- Stored in res/anim/.

### 🔹 Example (res/anim/fade_in.xml)

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:duration="1000"
   android:fromAlpha="0.0"
   android:toAlpha="1.0"/>
```

## 9 Font Resources

- Used to define custom fonts.
- Stored in res/font/.

### 🔹 Example (res/font/roboto_medium.xml)

```
<font-family xmlns:android="http://schemas.android.com/apk/res/android">
   <font android:font="@font/roboto_medium" android:weight="500"/>
</font>
```

### 🔹 Using in XML

```
<TextView
   android:text="Hello, World!"
   android:fontFamily="@font/roboto_medium"/>
```