NETCAMP

an ISO 9001 : 2015 Certified Company

Android project report

Under the supervision of: **Santu Purkait** (Director- Netcamp Solutions Pvt. Ltd.)
Submitted by: **Shubh Srivastava** (College Of Engineering Roorkee)

android

This android project works as an all-in-one tools app. I put effort into making the project. It would not have been possible without the kind support and help of many individuals and the organization. I would like to extend my sincere thanks to all of them. Their guidance and constant supervision provided necessary information regarding the project and also thanks for their support in completing the project. I would like to express my gratitude toward Mr. Santu Purkait, Director: Netcamp Solutions Pvt. Ltd. For their kind cooperation and encouragement which helped me in the completion of this project.

Name: **Shubh Srivastava**
College: **College Of Engineering Roorkee, ROORKEE (uttarakhand)**
College ID: **20040020**
Date: **31-08-2022**

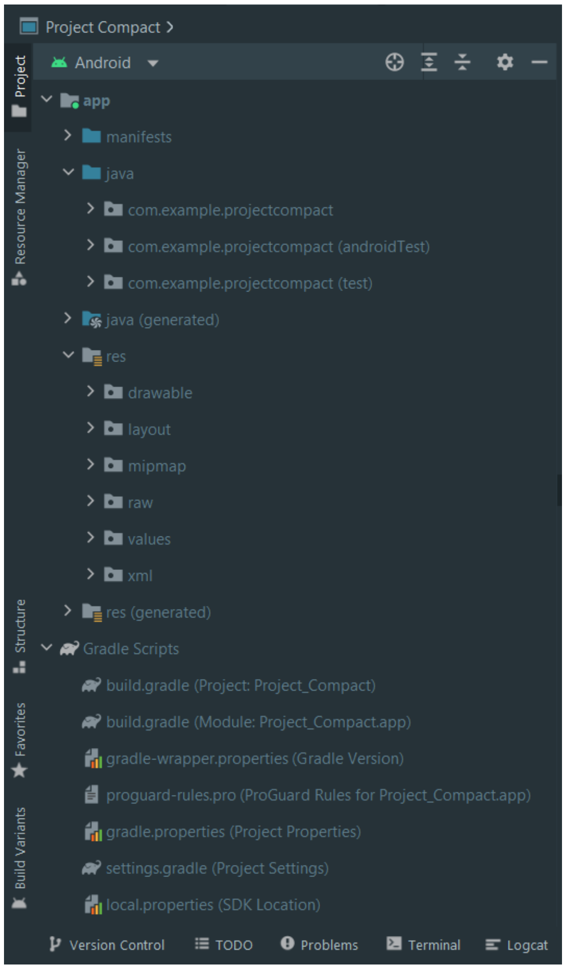
Signature: Shubh Srivastava

Project name:- **PROJECT COMPACT**

The project is all about making a tools app which includes Wi-Fi, Bluetooth, torch, text to speech, calculator, etc. with a login and sign-up page for users using Firebase authentication and real time database. By making this kind of project in my learning phase, I can understand the concept of android working, permissions & task managing.
I made it on Android Studio (by intellij) with java.

Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.
The Android Studio project contains one or more modules with resource files and source code files. These include different types of modules-

- Android app modules
- Library modules
- Google App Engine modules


Project Compact >
Android
Project
Resource Manager
app
manifests
java
com.example.projectcompact
com.example.projectcompact (androidTest)
com.example.projectcompact (test)
java (generated)
res
drawable
layout
mipmap
raw
values
xml
res (generated)
Gradle Scripts
build.gradle (Project: Project_Compact)
build.gradle (Module: Project_Compact.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Project_Compact.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
Structure
Favorites
Build Variants
Version Control
TODO
Problems
Terminal
Logcat

By default, Android Studio displays our project files in the Android project view, as shown in the image. This view is formed by modules to provide quick access to our project's key source files.

The app module contains the following folders :-

manifests – contains permissions and essential information in AndroidManifest.xml file.

java – it includes the source code java files and also JUnit test code. The JUnit test case is the set of code that ensures whether our program code works as expected or not. In Java, there are two types of unit testing possible, Manual testing and Automated testing. Manual testing is a special type of testing in which the test cases are executed without using any tool.

res – contains all non-code resources, UI strings, xml layout & bitmap images. In res folder, drawable contains images and icons used in the app, layout contains xml files, mipmap contains images or icons which require higher resolution during app execution, raw can be created to store mp3, mp4 like files, values folder have color & themes (for light & dark mode) with strings.xml file to store string variables which will be used frequently in the project such as app name.

Gradle is a build system that is used to automate building, testing, deployment, etc. “build.gradle” are scripts where one can automate the tasks.

Building blocks of an android project

An android component is simply a piece of code that has a well-defined life cycle e.g. Activity, Receiver, Service etc.
The core building blocks or fundamental components of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

Activity is a class that represents a single screen. There exist many types of layouts for an activity.

View is the UI element such as button, label, text field etc. Anything that we see is a view.

Intent is used to invoke components. It is mainly used to: start the service, launch an activity, display a web page, display a list of contacts, broadcast a message, dial a phone call etc.

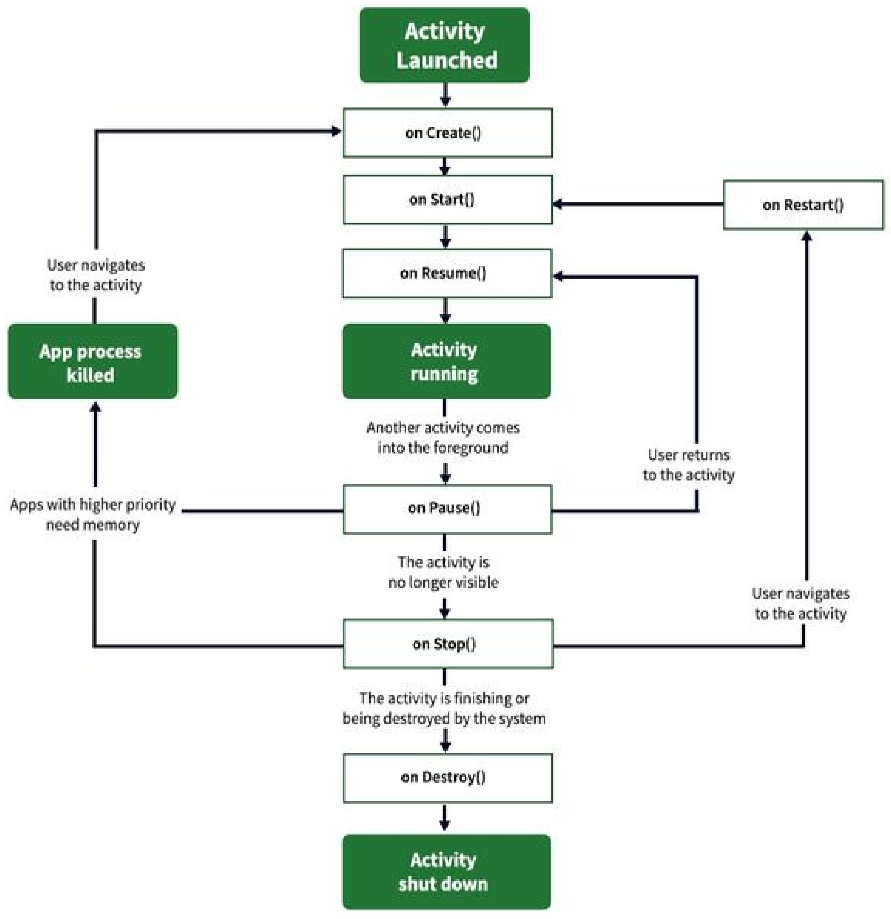
Service is a background process that can run for a long time. There are two types of services:
local and remote. Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

Content Providers are used to share data between the applications. They encapsulate data and provide it to applications through the single ContentResolver interface.

Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

Android activity cycle

As a user navigates through, out of, and back to our app, the **Activity** instances in our app transition through different states in their lifecycle. The **Activity** class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides. Within the lifecycle callback methods, we can declare how your activity behaves when the user leaves and re-enters the activity.
For example, if we're building a streaming video player, we might pause the video and terminate the network connection when the user switches to another app. When the user returns, we can reconnect to the network and allow the user to resume the video from the same spot.
In other words, each callback allows us to perform specific work that's appropriate to a given change of state. Doing the right work at the right time & handling transitions properly make our app more robust & performant.
Good implementation of the lifecycle callbacks can help ensure that our app avoids crashes.
To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: **onCreate(), onStart(), onResume(), onPause(), onStop() & onDestroy()**.The system invokes each of these callbacks as an activity enters a new state.


Activity Launched
on Create()
on Start()
on Restart()
User navigates to the activity
on Resume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down

Activity Lifecycle in Android

We must implement **onCreate()** callback, which fires when the system first creates the activity. On activity creation, the activity enters the *Created* state. In the **onCreate()** method, we perform basic application startup logic that should happen only once for the entire life of the activity.

For example, your implementation of **onCreate()** might bind data to lists, associate the activity with a **ViewModel**， and instantiate some class-scope variables.

This method receives the parameter **savedInstanceState,** which is a **Bundle** object containing the activity's previously saved state. If the activity has never existed before, the value of the **Bundle** object is null. Our activity does not reside in the Created state.

After the **onCreate()** method finishes execution, the activity enters the *Started* state, and the system calls the **onStart()** and **onResume()** methods in quick succession.

The system calls **onPause()** method as the first indication that the user is leaving our activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode). When our activity is no longer visible to the user, it has entered the stopped state, and the system invokes the **onStop()** callback.

This may occur, for example, when a newly launched activity needs large space in memory. The system may also call **onStop()** when the activity has finished running, and is about to be terminated.

From the Stopped state, the activity either comes back to interact with the user, or the activity is finished running and goes away. If the activity comes back, the system invokes **onRestart()**. If the Activity is finished running, the system calls **onDestroy()**. **onDestroy()** is called before the activity is destroyed. The system invokes this callback either because the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode).

An activity often needs to start another activity at some point. This need arises, for instance, when an app needs to move from the current screen to a new one. Depending on whether our activity wants a result back from the new activity it's about to start, we start the new activity using either the **startActivity()** or the **startActivityForResult()** method. In either case, we pass in an **Intent** object.

The **Intent** object specifies either the exact activity we want to start or describes the type of action we want to perform (and the system selects the appropriate activity for us, which can even be from a different application). An **Intent** object can also carry small amounts of data to be used by the activity that is started. If the newly started activity does not need to return a result, the current activity can start it by calling the **startActivity()** method. Sometimes we want to get a result back from an activity when it ends. To do this, we call the **startActivityForResult(Intent, int)** method, where the integer parameter identifies the call. This identifier is meant to disambiguate between multiple calls to **startActivityForResult(Intent, int)** from the same activity.

It's not a global identifier and is not at risk of conflicting with other apps or activities.The result comes back through your **startActivityForResult(Intent, int)** method. When a child activity exits, it can call **setResult (int)** to return data to its parent. The child activity must always supply a result code, which can be the standard results **RESULT_CANCELED, RESULT_OK,** or any custom values starting at **RESULT_FIRST_USER.** In addition, the child activity can optionally return an `Intent` object containing any additional data it wants. The parent activity uses the method **onActivityResult(Intent, int)**, along with the integer identifier the parent activity originally supplied, to receive the information. If a child activity fails for any reason, such as crashing, the parent activity receives a result with the code **RESULT_CANCELED**.

Project objectives

To make an all in one tool app with a user login page.

On start, the app shows a login page by which the user can login with email & password. This will get saved in firebase authentication. And if he/she is a new user, by clicking create new account button, the app directs the user to the create account page. By entering an email address with password and confirmation, the user needs to verify the given email. A register button appears but will not work until the email gets verified.

Now the app will direct the user to a details page in which the user enters his/her details, which will get stored in firebase realtime database by clicking a continue button and also directs to the app's main page.

Every time on opening the app, if the user is already logged in, the app shows that user details page. This details page will already get filled by the user details fetched from the app's database. And then the main page appears.

App's main page also consists of an "about" button for the user to change his/her details and a "logout " button to sign out from the app.

Main page has many buttons with images for different functions of this app.

For example after login, by clicking on the "text to speech" button from the menu, the app will direct the user to that page in which user input text is converted into speech.

Additional necessary requirements for the app

- The app should not crash at any moment during its working phase.
- Easy navigation for users throughout the app usage.
- User cannot access the app's features without verifying his/her email address.
- User cannot access the app’s features without giving his/her details.
- Updatable and deletable user authentication system and database.
- If any user has an invalid email then the authentication for that email should be deleted.
- After installing the app, the first page shown to the user should be the login page who should also have a button to create a new account. Then the database page will appear.
- User details should automatically be filled if the user is an old user.
- If the temporarily paused for email verification, it should not lose user progress rather starts from the current page.
- Once the app’s activity is destroyed, it should not create stacks but finishes properly without crashing.
- Should support light & dark themes for proper text visibility.

ANDROID BASICS

View in android (public class View extends Object)

android.view.View

A View is a simple building block of a user interface. It is a small rectangular box that can be TextView, EditText, or even a button. It occupies the area on the screen in a rectangular area and is responsible for drawing and event handling. View is a superclass of all the graphical user interface components. The use of a view is to draw content on the screen of the user’s Android device. A view can be easily implemented in an Application using the java code. Its creation is more easy in the XML layout file of the project.

There are many types of views like

- TextView
- EditText
- Button
- Image Button
- Date Picker
- RadioButton
- CheckBox buttons
- Image View

eg-> public class **TextView** extends **View**

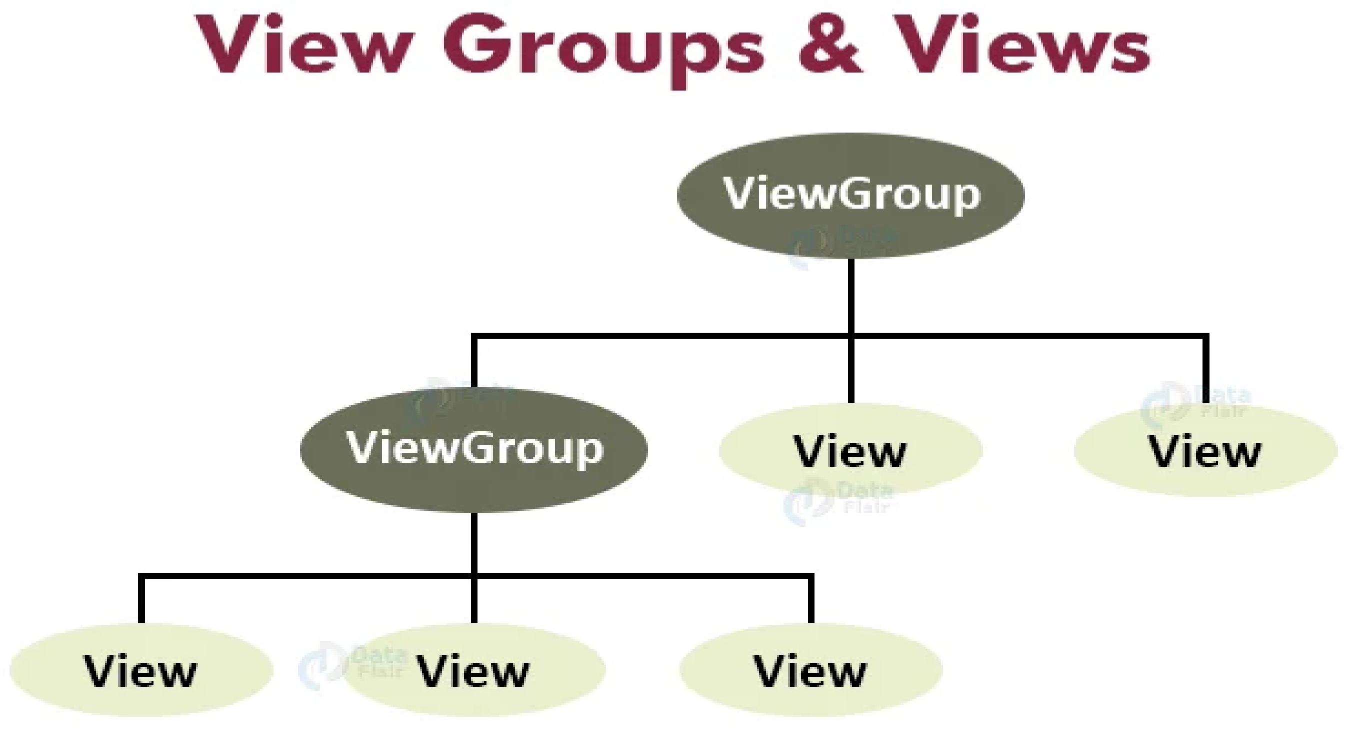
ViewGroup in android (public class ViewGroup extends View)

android.view.ViewGroup
A ViewGroup is a subclass of the View class and can be considered as a superclass of Layouts. It provides an invisible container to hold the views or layouts. ViewGroup instances and views work together as a container for Layouts. To understand in simpler words it can be understood as a special view that can hold other views that are often known as a child view.

Following are certain commonly used subclasses for ViewGroup:

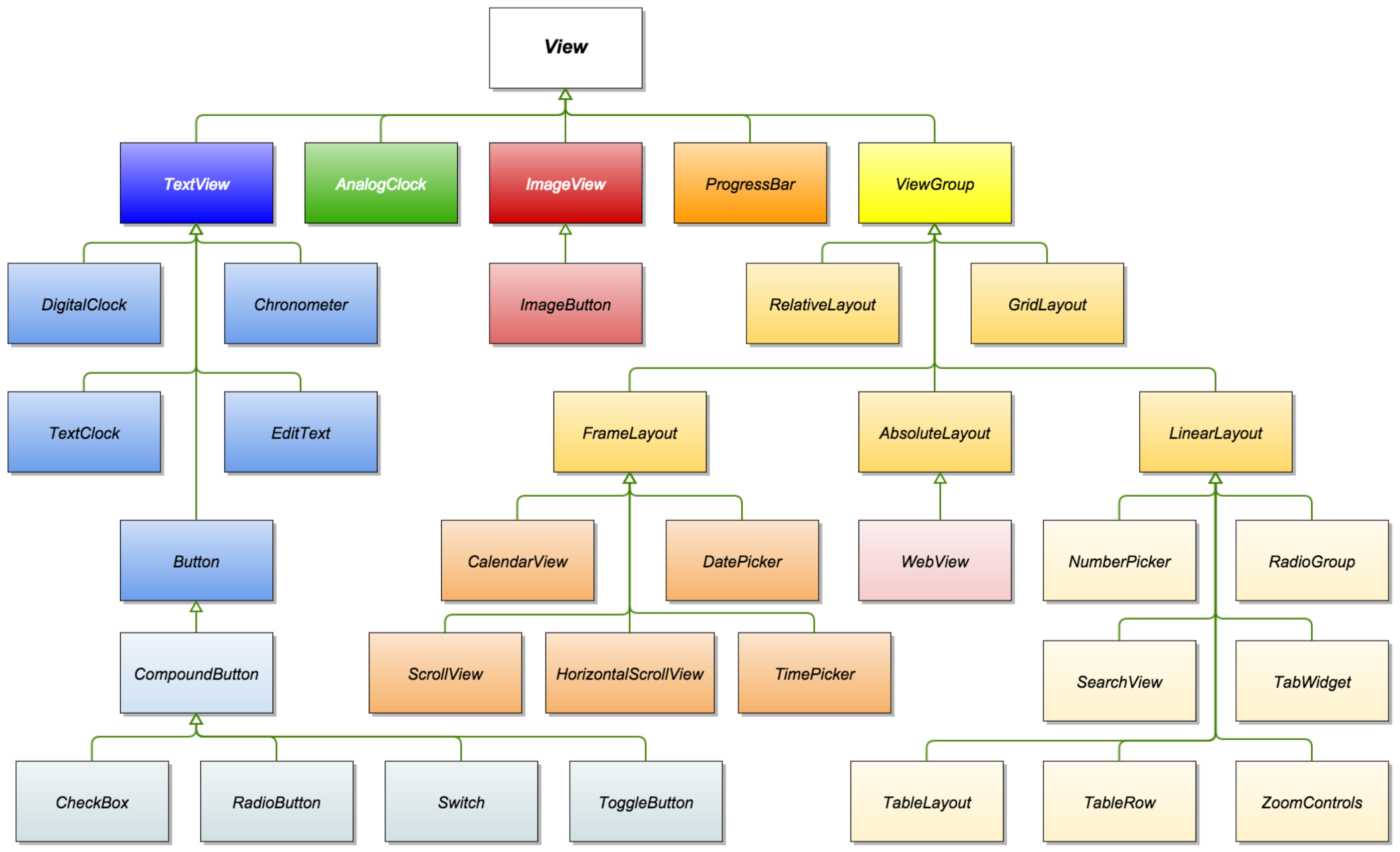
- LinearLayout
- RelativeLayout
- ConstraintLayout
- FrameLayout
- GridView
- ListView

eg -> public class **ConstraintLayout** extends **ViewGroup**



Views may have an integer id associated with them. These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree.

The Android View Class



Boilerplate java code

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

An **activity** is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for us in which we can place our UI with **setContentView(View).**

onCreate(Bundle) is where we initialize our activity. Most importantly, here we will usually call **setContentView(int)** with a layout resource defining our UI, and using **findViewById(int)** to retrieve the widgets in that UI that we need to interact with programmatically.

An activity provides a window in which the app draws its UI.

We have two classes for this purpose :- **android.app.Activity** & **androidx.appcompat.app.AppCompatActivity**

We should use **AppCompatActivity** instead of **Activity**.
AppCompatActivity is one of the main classes in the **Android Support Library**.

With every release of Android, new features are added, so the Android team works on supporting those new features for older devices. To solve this issue, they introduced the Android Support Library that can deliver the latest features to older devices (as old as API v7). That is, they backport those new features and allow them to be present on older devices. In this library, the default base class used for activities is **AppCompatActivity** instead of the normal Activity class. Some of those features include Material Design widgets and support action bars, etc.

Therefore, to get the most out of the Android Support Library (that is, to get the latest features supported across the highest number of devices), we must use AppCompatActivity instead of the normal Activity class.

The **savedInstanceState** is a reference to a **Bundle** object that is passed into the onCreate method of every Android Activity. Activities have the ability, under special circumstances, to restore themselves to a previous state using the data stored in this bundle. If there is no available instance data, the savedInstanceState will be null.

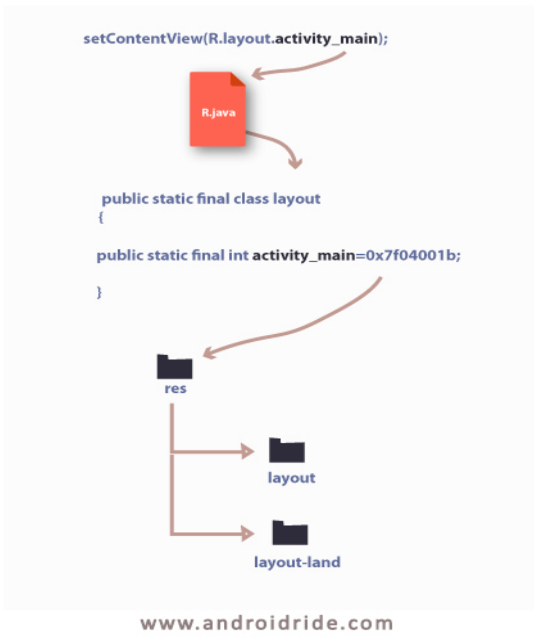
For example, the savedInstanceState will always be null the first time an Activity is started, but may be non-null if an Activity is destroyed during rotation.

Our Launcher activity in the manifest first gets called and it sets the layout view as specified in respective java files **setContentView(R.layout.activity_main);**.

Now this activity uses **setContentView(R.layout.activity_main)** to set xml layout to that activity which will actually render as the UI of our activity.

setContentView(int layoutid) - method of activity class. It shows the layout on screen.

R.layout.actiity_main - is an integer number implemented in a nested layout class of R.java class file.


setContentView(R.layout.activity_main);
R.java
public static final class layout
{
public static final int activity_main=0x7f04001b;
}
res
layout
layout-land
www.androidride.com

At the runtime device will pick up their layout based on the id given in setcontentview() method.
In android all the ui is written in xml. When an activity starts it needs to know what ui xml to display to the user, so when we call setContentView with a layout file, that particular xml is rendered.

setContentView() is used to display content for our Activity UI. If our Activity doesn't call this method, then no UI is present and the system will display a blank screen.

findViewById()

In the XML file when we add an element and set the properties to it we only set the values for these properties, now for each element, there is a class that draws it on the screen, this class has attributes with the same names of the properties in the XML file. Now using algorithms of read-write from files, these values from the XML file are transmitted to the java file (class) of the element, and then the class draws that element on the screen.
Before the class of the element draws it, there is a superclass of all elements, the parent for all of them it is called **View** (every element in the activity is called view) this class has the basic properties of all elements and this class is the one which the properties from the XML file will be transmitted to.
Method **findViewById()** returns an object of View type, this object holds the properties values then we need to cast it to the specific element, for example, TextView which is a class to draw the text view. This class and all elements' classes are subclasses of View class so what we do is downcasting, that when this method returns the object of View type we downcast it to the element class.
How does it find the properties?... it finds the properties of the element using the id of the tag in the XML file. First it searches in the XML file for the tag that holds the element's name and then it looks at the id if it is the id which it wants then it takes it otherwise it searches for another tag (with the same element name).
We give it the id by this approach, there is a class called **R** (resources), this class has nested classes (id, string, colors) these classes have attributes from the same type and hold specific values, for instance, the id class it has attributes that stores each id of each element in the XML file, so when we want to give findViewById() method the id, we go to this class and tell it to enter id class and choose the id of the element we want.
And the process goes that it enters the XML file and looks for the element that has this id and it takes the properties and passes them to the View class object and when it returns the object we downcast it to the element's class that we want to draw it and deal with it.

`View findViewById ( int resourceId );`

This method returns a **`View`** object.

All the views such as **`Button`, `TextView`, `ImageView`, `RelativeLayout`, `LinearLayout`** etc. extends/inherits **`ViewGroup`** class, which in turn extends **`View`** class. Thus, all the views used in the XML layout, one way or other extends to **`View`** class. So this method returns an instance of that class.

By Type Casting the **`View`** returned by this method to the View class we have assigned our **`resourceId`.**

For example :-

Let's say we have an TextView with ID : **textView1**

```
<TextView
      android:layout_width = "match_parent"
      android:layout_height = "wrap_content"
      android:id = "@+id/textView1"  />
```

Now, **`TextView`** extends **`ViewGroup`** which then extends **`View`**,

Thus, **`TextView`** is the child class for **`View`**.

Now, in our Activity, we need an instance of **`TextView`** which corresponds to the provided View ID. This is achieved by this method as follows:

```
public class MainActivity extends AppCompatActivity {
      private TextView mTextView;

            @Override
            protected void onCreate ( Bundle savedInstanceState )  {
             super.onCreate( savedInstanceState );
             setContentView( R.layout.activity_main );
             mTextView = (TextView) findViewById ( R.id.textView1 );
             String text = "Some text for our TextView";
             mTextView.setText( text );
      }
  }
```

Once the Activity is created, we instantiate the **`TextView`** with the corresponding ID of the resource (in this case, `textView1`). We have passed an ID as **`R.id.textView1`** which is an integer. This is used to identify the view in the layout the activity inflates.

`findViewById()` just returns an instance of **`View`**. We have to type-cast it to **`TextView`**. So just add **`(TextView)`** before **`findViewById()`** and we have our instance of **`TextView`** in control.

Some notes:

- Let's say we have two activities named **ActivityA** and **ActivityB** with layouts **layoutA** and **layoutB** containing TextViews with Resource IDs as **textA** and **textB** respectively.
- In **ActivityA**, if we try to instantiate **textB**, we will encounter **NullPointerException**. This is because **textB** does not exist in **layoutA** which was the content for **ActivityA**.
  It is important to note that **findViewById()** works on the layout as the parameter given to the method **setContentView()**.

- If we have a view and we try to cast it to another view class, we will encounter a **ClassCastException**.
  For example :

```
1. private EditText mEditText;
2. mEditText = (TextView) findViewById ( R.id.edit_text );
```

- This is because `mEditText` is an object of type `EditText`, but we are type-casting the returned `View` to `TextView`.

Starting with API 26, **findViewById uses inference for its return type**, so we no longer have to cast.

<T extends View> T **findViewById**(int id)

Some setups:

- We can configure the starting activity (default activity) of our application via the **intent-filter** in **AndroidManifest.xml**.

```
<activity
   android:name=".MainActivity"
   android:exported="true">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
<activity
   android:name=".JavaFile2"
   android:exported="false">
</activity>
<activity
   android:name=".JavaFile3"
   android:exported="false">
</activity>
```

- We can also **remove action bar** from a particular activity by adding this line in its activity tag `android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"`

- To support colors in **dark theme,** copy the colors.xml file in \res\values but with \res\values-night. colors.xml (night) should not have extra attributes for colors other than those present in colors.xml. colors.xml can have more number of attributes than colors.xml (night).

  For example in colors.xml we have :-

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <resources>
    . . .
    <color name="myColor">#FF000000</color>
    . . .
    <drawable name="bgimg">@drawable/myBackground</drawable>
  </resources>
  ```

  And in any activity's xml file we have:-

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

R.java file

Android **R.java** is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of the res/ directory.

If we create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

```java
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package com.example.helloandroid;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello_world=0x7f040001;
        public static final int menu_settings=0x7f040002;
    }
    public static final class style {
        /**
        Base application theme, dependent on API level. This theme is replaced
        by AppBaseTheme from res/values-vXX/styles.xml on newer devices.


          Theme customizations available in newer API levels can go in
          res/values-vXX/styles.xml, while customizations related to
          backward-compatibility can go here.


        Base application theme for API 11+. This theme completely replaces
        AppBaseTheme from res/values/styles.xml on API 11+ devices.

 API 11 theme customizations can go here.

        Base application theme for API 14+. This theme completely replaces
        AppBaseTheme from BOTH res/values/styles.xml and
        res/values-v11/styles.xml on API 14+ devices.

 API 14 theme customizations can go here.
      */
        public static final int AppBaseTheme=0x7f050000;
        /**  Application theme.
 All customizations that are NOT specific to a particular API-level can go here.
      */
        public static final int AppTheme=0x7f050001;
    }
}
```

xml activity file example

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@drawable/asbackground"
   tools:context=".MainActivity">
        <TextView
           android:id="@+id/textView"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="login to my netcamp project app"
           android:textColor="@color/textColor"
           android:textSize="20dp"
        />
        <Button
           android:id="@+id/button"
           android:layout_width="176dp"
           android:layout_height="51dp"
           android:background="#1C848E"
           android:text="login"
           android:textColor="#FFFFFF"
        />
        . . .
</androidx.constraintlayout.widget.ConstraintLayout>
```

Dimensions of an item in xml can be

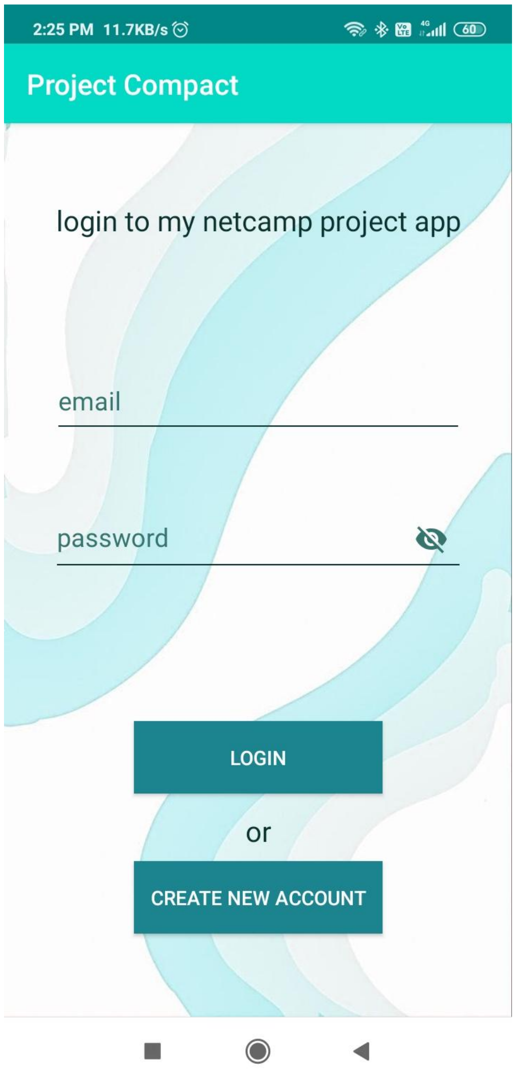
- an exact number
- **match_parent**, which means the view wants to be as big as its parent (minus padding)
- **wrap_content**, which means that the view wants to be just big enough to enclose its content (plus padding)

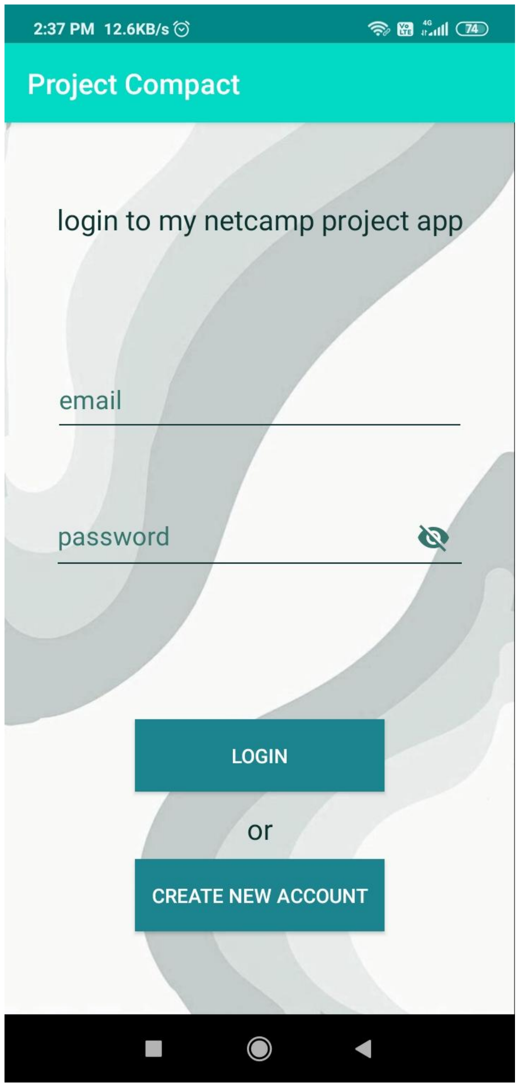
App activities

Login page - for users to sign in with email and password.

Design - two edit texts, one for user email address and other for password. One login button, one create new account button.

Functions - login button sign in the user with given email & password and directs to user details page, create new account button directs to new account creation page. If the password will not match an error message will get displayed.





MainActivity.java

```java
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;

public class MainActivity extends AppCompatActivity {
```

`   Button login, create;` object of **Button** class

`   TextInputEditText tiet1, tiet2;` object of **TextInputEditText** which is a special subclass of **EditText** designed to use as a child of **TextInputLayout.** TextInputLayout is a layout which wraps TextInputEditText, EditText, or descendant to show a floating label when the hint is hidden while the user inputs text.

`   ProgressBar pb;` object of **ProgressBar,** a user interface which indicates the progress of an operation. It supports two modes to represent progress, **determinate** & **indeterminate.**

`   public static FirebaseAuth fba;` com.google.firebase.auth.**FirebaseAuth** is a gateway to the firebase authentication API. With this we can reference FirebaseAuth objects to manage user accounts and credentials.

To instantiate FirebaseAuth, we use the getInstance() method.

It has many methods to work with users.

Firebase uses INTERNET permission in AndroidManifest.xml

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="com.example.projectcompact">

   <uses-permission android:name="android.permission.INTERNET" />

   <application . . .
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main); set the UI layout for our Activity
       login = findViewById(R.id.button);  finds a View that was identified by the id
attribute from xml layout resource
       create = findViewById(R.id.button2);
       tiet1 = findViewById(R.id.textInputEditText);
       tiet2 = findViewById(R.id.textInputEditText2);
       pb = findViewById(R.id.progressBar);
       fba = FirebaseAuth.getInstance();  instantiating FirebaseAuth object
```

setOnClickListener(View.OnClickListener) is a method in **View** class which helps us to link a Listener with certain attributes. It is basically used with Button, ImageButton etc.

OnClickListener is an inner functional interface in the View class which defines the **onClick(View)** method that is called when the view(component) is clicked.

We can make a seperate class implementing OnClickListener and can also be used as lambda expression in the parameter of setOnClickListener().

```
       login.setOnClickListener(new View.OnClickListener(){
           @Override
           public void onClick(View view) {
               Code to check if any field is not empty
               String s1 = tiet1.getText().toString(), s2 = tiet2.getText().toString();
               if(s1.isEmpty())
               {
                   tiet1.setError("fill email"); set error message if email is not filled
                   return;
               }
               if(s2.isEmpty())
               {
                   tiet2.setError("fill password"); set error message if password is empty
                   return;
               }
```

Progress bar xml code :-

```
<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="invisible" />
```

```
               pb.setVisibility(View.VISIBLE); set ProgressBar visible
            //processing authentication
```

Code for firebase sign in user with email & password

and

adding a task complete listener, we have:-

public abstract class **Task**, represents an asynchronous operation.

It have many abstract methods :-

abstract Task<ResultT> **addOnCompleteListener(OnCompleteListener**<ResultT> listener **)**

abstract Task<ResultT> **addOnFailiureListener(OnFailiureListener** listener **)**

abstract Task<ResultT> **addOnSuccessListener(OnSuccessListener**<? super ResultT> lt **)**

abstract ResultT **getResult()**

abstract boolean **isComplete()**

abstract boolean **isSuccessful()**

and we have public interface **OnCompleteListener**<ResultT>, which is a functional interface

it have a public method:-

abstract void **onComplete(Task**<ResultT> task **)**

```
               fba.signInWithEmailAndPassword(s1,s2).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
                   @Override
                   public void onComplete(@NonNull Task<AuthResult> task) {
                       if(task.isSuccessful()){
                           pb.setVisibility(View.INVISIBLE); set ProgressBar invisible
                 Toast.makeText(MainActivity.this,"logged in", Toast.LENGTH_SHORT).show();
                           Intent i = new Intent(MainActivity.this, Third.class);
                           startActivity(i); finish();
                       }
```

```
                        else
                        {
                            pb.setVisibility(View.INVISIBLE);
    Toast.makeText(MainActivity.this, "something went wrong", Toast.LENGTH_SHORT).show();
                        }
                    }
                });
            }
        });
```

Toast provides simple feedback about an operation in a small popup. It only fills the amount of space required and the current activity remains visible & interactive. It automatically disappears after a timeout.

To instantiate a **Toast**, use **makeText()** method which takes following parameters:-

The application **Context**

Text that should appear to the user

Duration that the toast should appear on the screen

public static Toast **makeText(Context** context, **CharSequence** text, **int** duration**)** returns a properly initialized Toast object. To display the Toast, call **show()** method.

Have two constants, public static final int **LENGTH_SHORT** & **LENGTH_LONG.**

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

We can chain our methods to avoid holding on to the Toast object

```
Toast.makeText(context, text, duration).show();
```

public abstract class **Context** gives global information about an object environment. This is an abstract class whose implementation is provided by the android system. It allows access to application-specific resources and classes, as well as up-calls for application level operations such as launching activities, broadcasting and receiving intents, etc.

On click listener for create new account button

```
       create.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               Intent i = new Intent(MainActivity.this, Second.class);
               startActivity(i); finish();
           }
       });
   }
```

public class **Intent**, gives a description of an operation to be performed. It can be used with **startActivity()** to launch an **Activity**, **broadcastIntent()** to send it to any interested **BroadcastReceiver** components, and **Context.startService(Intent)** or **Context.bindService(Intent, ServiceConnection, int)** to communicate with a background service. Intents are objects of the **android.content.Intent** type. Our code can send them to the android system defining the components we are targeting.

For example, via the **startActivity(Intent)** method we can define that the Intent can be used to start an **Activity**. This method is defined on the **Context** object which **Activity** extends. Following code demonstrates how we can start another Activity via an **Intent:**

```
Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

For example, the following code tells the android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter:

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.vogella.com/"));
startActivity(i);
```

Activity.finish() can be called to kill (destroy) an **Activity** instance. If we don't need to close our Activity manual, which is true in many cases, we don't need to call this method. But if we require a button somewhere in our activity that says "close", then we should use this method. But in general the back button behavior in Android will handle things like this. The back button does not actually finish our activity, **finish()** calls the **onDestroy()** method right away, while the back button does not. When the back button is pressed, the **onStop()** method is called, but the **onDestroy()** method call might be delayed by the system, so that the Activity can be resumed by the system which is cheaper (in resources) than a full restart.

It can also be used while starting an **Activity**, because it closes the current Activity to and that Activity will no longer be available on the Activity stack.

onStart() method used here to configure the app starting according to conditions.

```
    @Override
    protected void onStart() {
        super.onStart();
```

If user present but didn't verify the email, delete user from FirebaseAuth for re-login

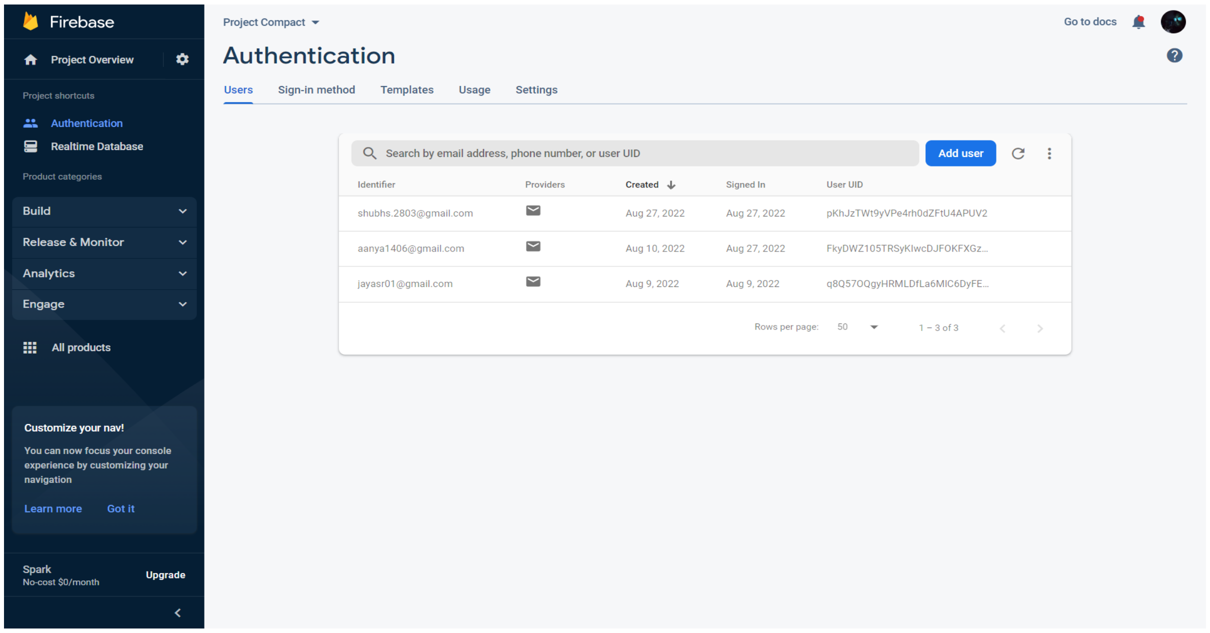
```
        if(fba.getCurrentUser()!=null && !fba.getCurrentUser().isEmailVerified())
        {
            MainActivity.fba.getCurrentUser().delete();
            Toast.makeText(this, "login requires", Toast.LENGTH_SHORT).show();
        }
```

If user present and email is also verified, app should starts from about page

```
        else if(fba.getCurrentUser()!=null && fba.getCurrentUser().isEmailVerified())
        {
            Intent i = new Intent(MainActivity.this, Third.class);
            startActivity(i); finish();
        }
```

Else if user is not present (not logged in or sign in yet), stay on login page

```
        else
        {
            Toast.makeText(this, "login requires", Toast.LENGTH_SHORT).show();
        }
    }
}
```



Authentication

Customize your nav!

You can now focus your console experience by customizing your navigation

Identifier	Providers	Created	Signed In	User UID
shubhs.2803@gmail.com	✉	Aug 27, 2022	Aug 27, 2022	pKhJzTWt9yVPe4rh0dZFtU4APUV2
aanya1406@gmail.com	✉	Aug 10, 2022	Aug 27, 2022	FkyDWZ105TRSyKIwcDJFOKFXGz...
jayasr01@gmail.com	✉	Aug 9, 2022	Aug 9, 2022	q8Q57OQgyHRMLDfLa6MIC6DyFE...

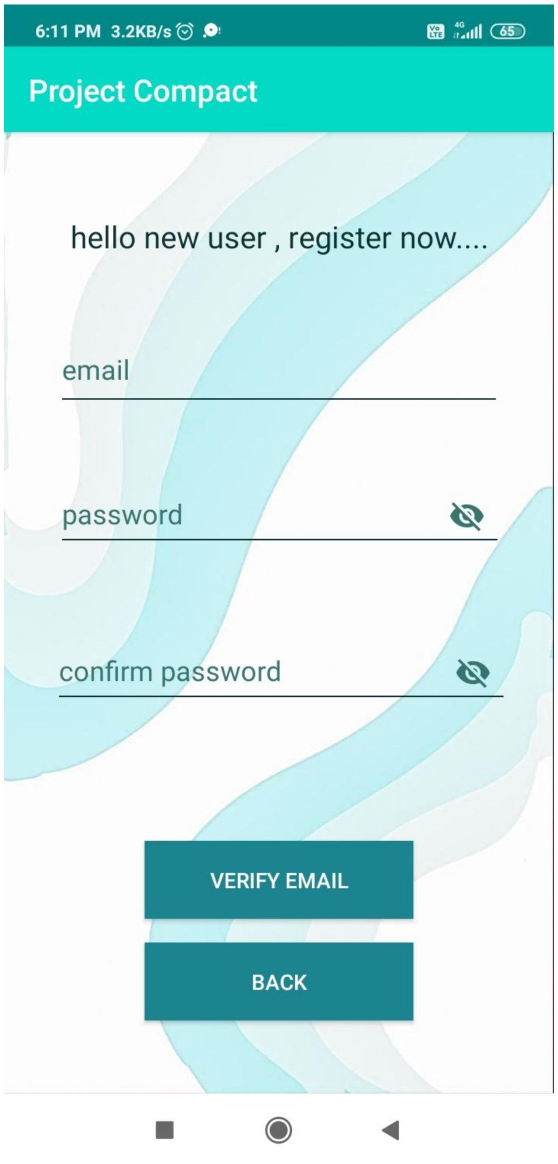
Rows per page: 50 1 – 3 of 3

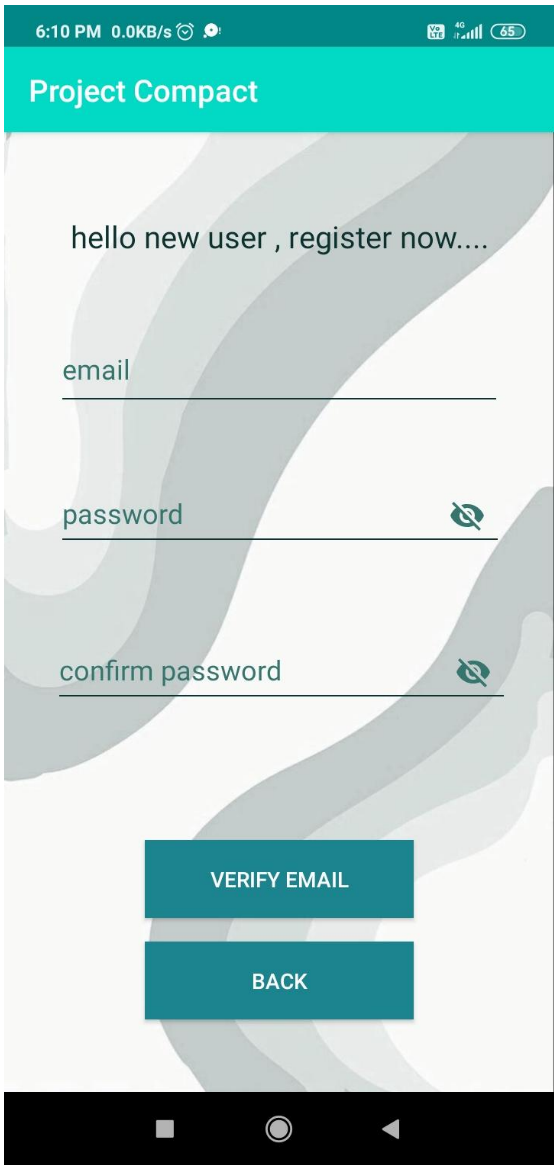
Firebase Authentication of my app

Create new account page - for users to sign in with email and password.

Design - three edit texts, one for user email address and other two for password & conformation. One verify button, one register button and one back button.

Functions - After entering details click on verify button to send an email verification link on the given email. The verify button should disappear after that and a register button should appear which directs to the about page only when email gets verified. A back button to return to the previous page (login)





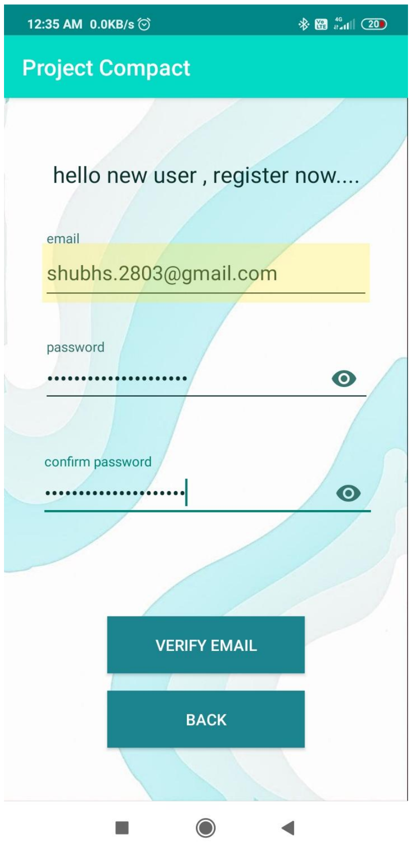
Precautions - app should not crash in any stage of these processes.

Users cannot access app functions in any way without verifying email.

App should not destroy the progress when it gets minimized for email verification.

Verification email should not be expired.

Proper toasts to guide users throughout the process.


12:35 AM 0.0KB/s

Project Compact

hello new user , register now....

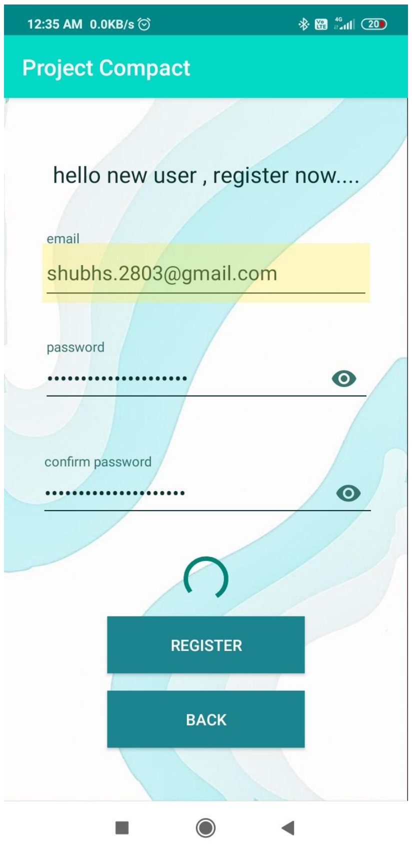
email
shubhs.2803@gmail.com

password
••••••••••••••••••••

confirm password
••••••••••••••••••••

VERIFY EMAIL

BACK


12:35 AM 0.0KB/s

Project Compact

hello new user , register now....

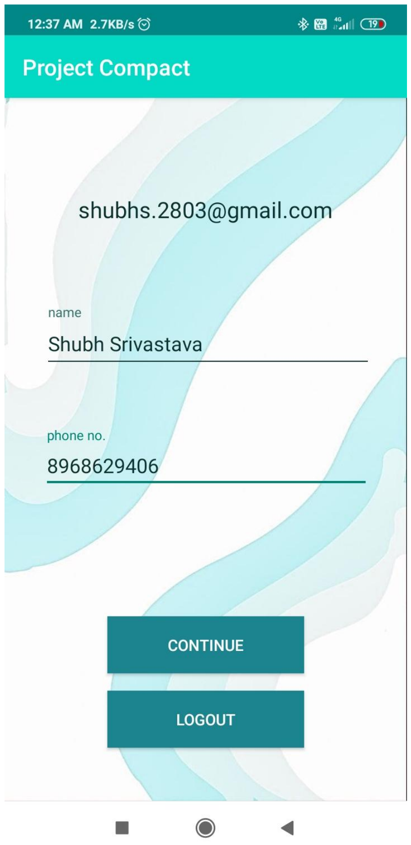
email
shubhs.2803@gmail.com

password
••••••••••••••••••••

confirm password
••••••••••••••••••••

REGISTER

BACK


12:37 AM 2.7KB/s

Project Compact

shubhs.2803@gmail.com

name
Shubh Srivastava

phone no.
8968629406

CONTINUE

LOGOUT

Second.java

```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.Objects;
```

```
public class Second extends AppCompatActivity
{
   Button verify, register, back;
   TextInputEditText tiet1, tiet2, tiet3;
   ProgressBar pb;
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_second);
       verify = findViewById(R.id.button3);
       register = findViewById(R.id.button20);
       back = findViewById(R.id.button4);
       tiet1 = findViewById(R.id.textInputEditText3);
       tiet2 = findViewById(R.id.textInputEditText4);
       tiet3 = findViewById(R.id.textInputEditText5);
       pb = findViewById(R.id.progressBar2);
       verify.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view)
           {
            Checking user inputs ....
               String s1 = tiet1.getText().toString(), s2 = tiet2.getText().toString();
               String s3 = tiet3.getText().toString();
               if(s1.isEmpty())
               {
                   tiet1.setError("fill email");
                   return;
               }
               if(s2.isEmpty())
               {
                   tiet2.setError("fill password");
                   return;
               }
```

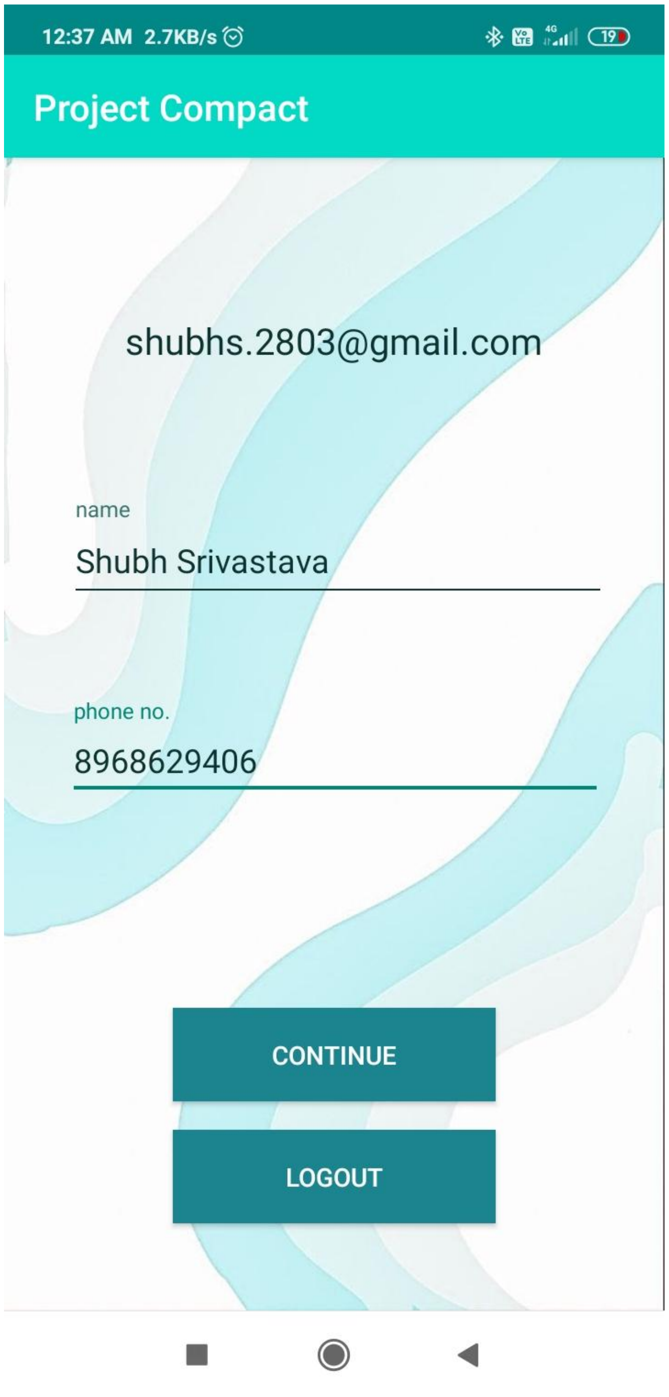
```
                if(s3.isEmpty())
                {
                    tiet3.setError("confirm password");
                    return;
                }
                if(!s2.equals(s3))
                {
                    tiet3.setText("");
                    tiet3.setError("incorrect");
                    return;
                }
                //processing registration
                pb.setVisibility(View.VISIBLE);

MainActivity.fba.createUserWithEmailAndPassword(s1,s2).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if(task.isSuccessful())
                        {
                            //an email verification will be sent by sendEmailVerification()

MainActivity.fba.getCurrentUser().sendEmailVerification().addOnCompleteListener(new
OnCompleteListener<Void>() {
                                @Override
                                public void onComplete(@NonNull Task<Void> task) {
                                    if (task.isSuccessful())
                                    {
      Toast.makeText(Second.this, "verify your email ...", Toast.LENGTH_SHORT).show();
      Verify button gets invisible, register button gets visible and lock the edit which
      prevent users from changing details after successful email sending.
                                        verify.setVisibility(View.INVISIBLE);
                                        register.setVisibility(View.VISIBLE);
                                        tiet1.setFocusable(false);
                                        tiet2.setFocusable(false);
                                        tiet3.setFocusable(false);
                                    }
                                    else {
                                        pb.setVisibility(View.INVISIBLE);
                  If firebase is unable to send email on a particular email address
                  Toast.makeText(Second.this, "invalid email", Toast.LENGTH_SHORT).show();
                                        MainActivity.fba.getCurrentUser().delete();
                                  Intent i = new Intent(Second.this, MainActivity.class);
                                        startActivity(i);
                                        finish();
                                    }
                                }
                            });
                        }
                        else
                        {
                  If firebase unable to create a new user due to some issues
                            pb.setVisibility(View.INVISIBLE);
              Toast.makeText(Second.this, "something went wrong", Toast.LENGTH_SHORT).show();
                        }
                    }
                });
            }
        });

            register.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view)
                {
                MainActivity.fba.getCurrentUser().reload();
                  Need to reload user after verification to check correctly
                if(!MainActivity.fba.getCurrentUser().isEmailVerified() &&
                  MainActivity.fba.getCurrentUser()!=null)
                  {
                  If email is not yet verified by user
                Toast.makeText(Second.this, "verification pending", Toast.LENGTH_SHORT).show();
                }
                if(MainActivity.fba.getCurrentUser().isEmailVerified() &&
                   MainActivity.fba.getCurrentUser()!=null)
                   {
                   If email gets verified, goto third page (the about user page)
        Toast.makeText(Second.this, "registered successfully", Toast.LENGTH_SHORT).show();
                    Intent i = new Intent(Second.this, Third.class);
                    startActivity(i); finish();
                }
            }
        });

        back.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View view)
            {
                Intent i = new Intent(Second.this, MainActivity.class);
                startActivity(i); finish();
            }
        });
    }

    @Override
    protected void onDestroy()
    {
if a user closes the app without email verification then his/her email should not be saved
to Firebase Auth, else when he/she reopens the app he will directly move to our app
content even if he/she has not a verified email.
        super.onDestroy();
        if(MainActivity.fba.getCurrentUser()!=null &&
                                !MainActivity.fba.getCurrentUser().isEmailVerified())
        {
            MainActivity.fba.getCurrentUser().delete();
        }
    }
}
```

About page - a page which gets the data of a new user by their input or shows the user data if a user already has an account. Users can update his/her data by this page.

Design - one text field which shows the logged in user email. Two edit texts to get user input (name and mobile number). One submit button to update the firebase realtime database. One logout button.

Functions - user data must get fetched from firebase realtime database if user already exists in firebase authentication. Logout button should sign out the user out of the app and directs back to the login page.

Precautions - App should not crash while fetching or updating data because of database security rules or any kind of error. Logout button should not delete the user data from auth and database.



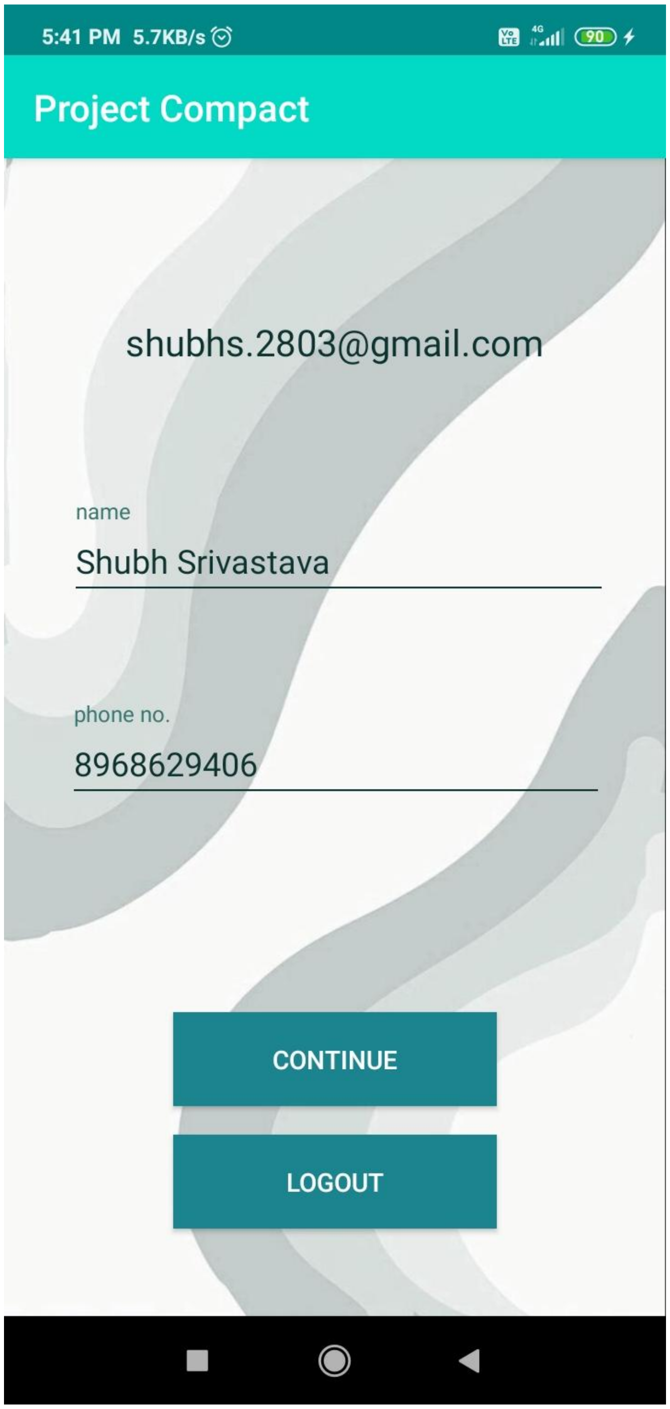
shubhs.2803@gmail.com

name

Shubh Srivastava

phone no.

8968629406



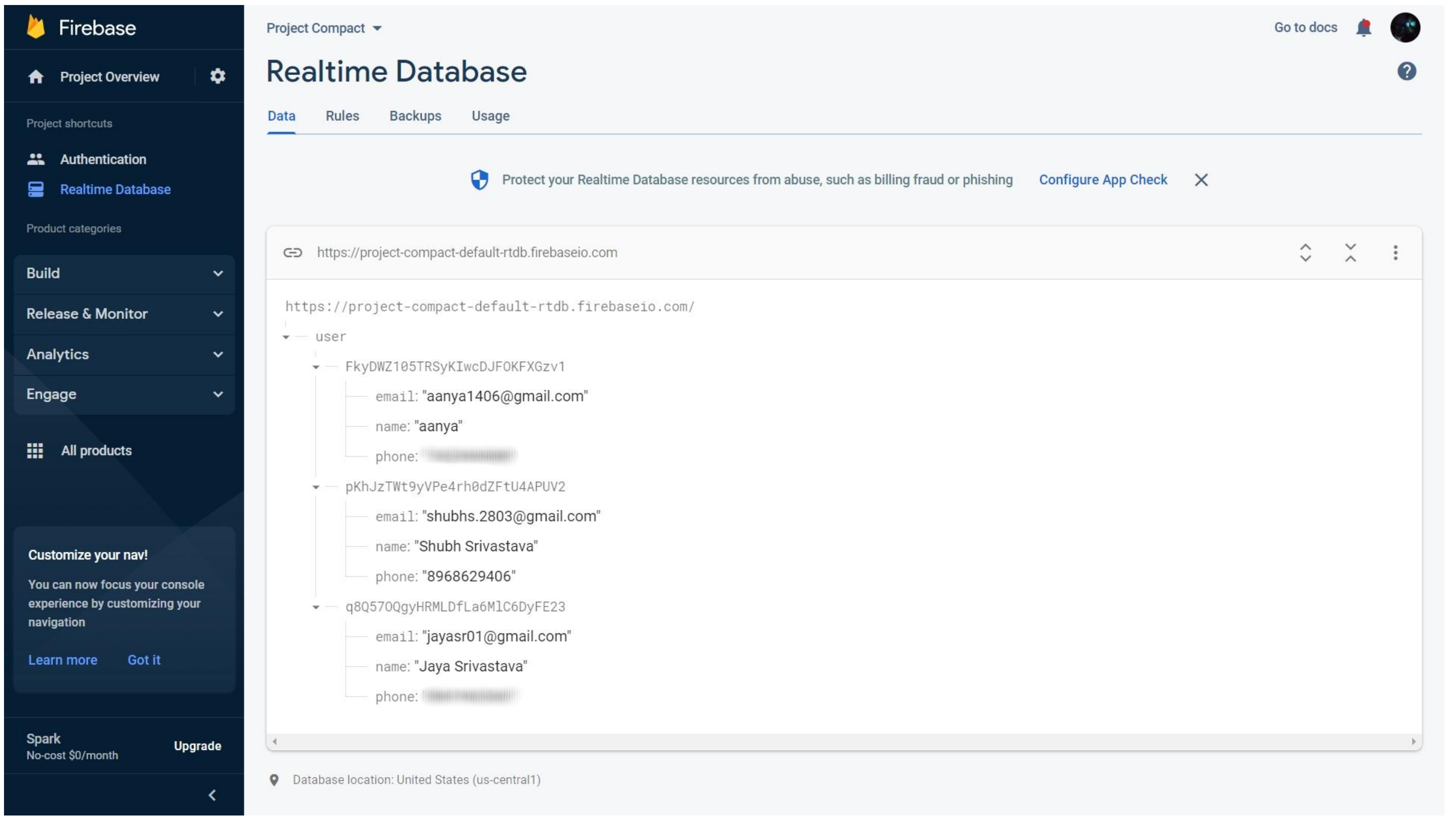
shubhs.2803@gmail.com

name

Shubh Srivastava

phone no.

8968629406



Protect your Realtime Database resources from abuse, such as billing fraud or phishing

https://project-compact-default-rtdb.firebaseio.com

```
https://project-compact-default-rtdb.firebaseio.com/
- user
  - FkyDWZ105TRSyKIwcDJFOKFXGzv1
      email: "aanya1406@gmail.com"
      name: "aanya"
      phone: [redacted]
  - pKhJzTWt9yVPe4rh0dZFtU4APUV2
      email: "shubhs.2803@gmail.com"
      name: "Shubh Srivastava"
      phone: "8968629406"
  - q8Q570QgyHRMLDfLa6MlC6DyFE23
      email: "jayasr01@gmail.com"
      name: "Jaya Srivastava"
      phone: [redacted]
```

Database location: United States (us-central1)

Firebase Realtime Database of my app.

It has a key called “user” which further has many UIDs as its key to determine every user uniquely.

The UID key of each user is further branched into three key value pairs, “email”, “name” & “mobile”.

Third.java

```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
```

```
public class Third extends AppCompatActivity
{
   Button submit, logout;
   TextView tv;
   TextInputEditText tiet1,tiet2;
   FirebaseDatabase fbdb;
   DatabaseReference node;
   public static String user_email="", user_name="", user_phone="";
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_third);

       submit = findViewById(R.id.button5);
       logout = findViewById(R.id.button8);
       tv = findViewById(R.id.textView4);
       tiet1 = findViewById(R.id.textInputEditText6);
       tiet2 = findViewById(R.id.textInputEditText7);
       fbdb = FirebaseDatabase.getInstance();
```

Public class **FirebaseDatabase** extends Object

The entry point for accessing firebase database, we can get an instance by calling **getInstance()** method. To access a location in the database or to read/write use the **getReference()** (return reference to the root node) or use the **getReference(String** path**)** (returns the specified reference of the database)

Public static **DatabaseReference getReference()**

```
       submit.setOnClickListener(new View.OnClickListener()
       {
           @Override
           public void onClick(View view)
           {
               node = fbdb.getReference("user");
```

Give "user" as our DatabaseReference object bcs we need to work on it.

This is the required reference which contains all UIDs of the users.

```
                String s1 = tv.getText().toString(), s2 = tiet1.getText().toString();
                String s3 = tiet2.getText().toString();
```

Checking user inputs ....

```
                if(s2.isEmpty())
                {
                    tiet1.setError("fill name");
                    return;
                }
                if(s3.length()!=10)
                {
                    tiet2.setError("fill phone");
                    return;
                }
                User u = new User(s1,s2,s3);
```

A class which has fields email, name & mobile, a constructor with parameters for our use, a constructor without parameters for Firebase use and getters setters.

```
                node.child(MainActivity.fba.getCurrentUser().getUid()).setValue(u);
```

Inserting data into firebase realtime database. **child()** gets a reference to a location in the database related to this. We've passed UID as a parameter to access a particular child in the "user" key.

For basic write operations we can use **setValue()** to save data to a specified reference, replacing any existing data at that path. We can also pass a custom java object in it, if the class that defines it has a default constructor that takes no arguments and has public getters for the properties to be assigned. If we use a java object like this, the content of our object is automatically mapped to child location in a nested fashion. Using a java object also makes our code readable and easy to maintain.

DatabaseReference has many useful methods like these:

String getKey() -> the last token in the location pointed to by this reference or null if this reference points to the database root.

DatabaseReference getParent() -> a database reference to the parent location or null if the instance references the root location.

DatabaseReference getRoot() -> a reference to the root location of this database.

```
            Toast.makeText(Third.this, "successfully submitted", Toast.LENGTH_SHORT).show();
                Intent i = new Intent(Third.this, Fourth.class);
                startActivity(i); finish();
            }
        });
        logout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MainActivity.fba.signOut();
                Intent i = new Intent(Third.this, MainActivity.class);
                startActivity(i); finish();
            }
        });
    }

    @Override
    protected void onStart()
    {
        super.onStart();
        if(MainActivity.fba.getCurrentUser() != null)
        {
            user_email = MainActivity.fba.getCurrentUser().getEmail();
            tv.setText(user_email);
            fetching data from Firebase's Realtime Database
```

UID is the safest way to provide as a key in a database, as it is configured by firebase itself. We cannot give email as a key bcs firebase key doesn't accept special characters like '@','.'

```
FirebaseDatabase.getInstance().getReference("user").child(MainActivity.fba.getCurrentUser(
).getUid()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
                @Override
                public void onComplete(@NonNull Task<DataSnapshot> task) {
                    if(task.getResult().exists())
                    {
                        DataSnapshot dbsnap = task.getResult();
                        user_name = String.valueOf(dbsnap.child("name").getValue());
                        user_phone = String.valueOf(dbsnap.child("phone").getValue());
                    }
```

A **DataSnapshot** instance contains data from a Firebase Database location. Any time you read Database data, you receive the data as a DataSnapshot. **getValue()** returns the data contained in this data snapshot.

```
                    tiet1.setText(user_name);
                    tiet2.setText(user_phone);
                }
            });
        }
    }
}
```

User.java model class for firebase database

```
public class User
{
   String email, name, phone;
   // constructor for our use
   public User(String email,String name,String phone)
   {
       this.email = email;
       this.name = name;
       this.phone = phone;
   }
   //constructor for firebase use
   public User()
   {
            // firebase uses this constructor to instantiate object and
            // use getter setter to set values in database
   }
   // getters
   public String getEmail() {
       return email;
   }
   public String getName() {
       return name;
   }
   public String getPhone() {
       return phone;
   }
```

```
    // setters
    public void setEmail(String email) {
        this.email = email;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

After all this authentication and database work, my main app structure gets displayed to the user.

App's main page - displays all the contents of our app and its functions.

Design - In this page I removed the action bar from the top to add a small horizontal scroll view at the top through xml, which contains app name and two buttons (about and logout). Below this the app has a vertical scroll view which contains large interactive buttons for navigation to different features/tools.

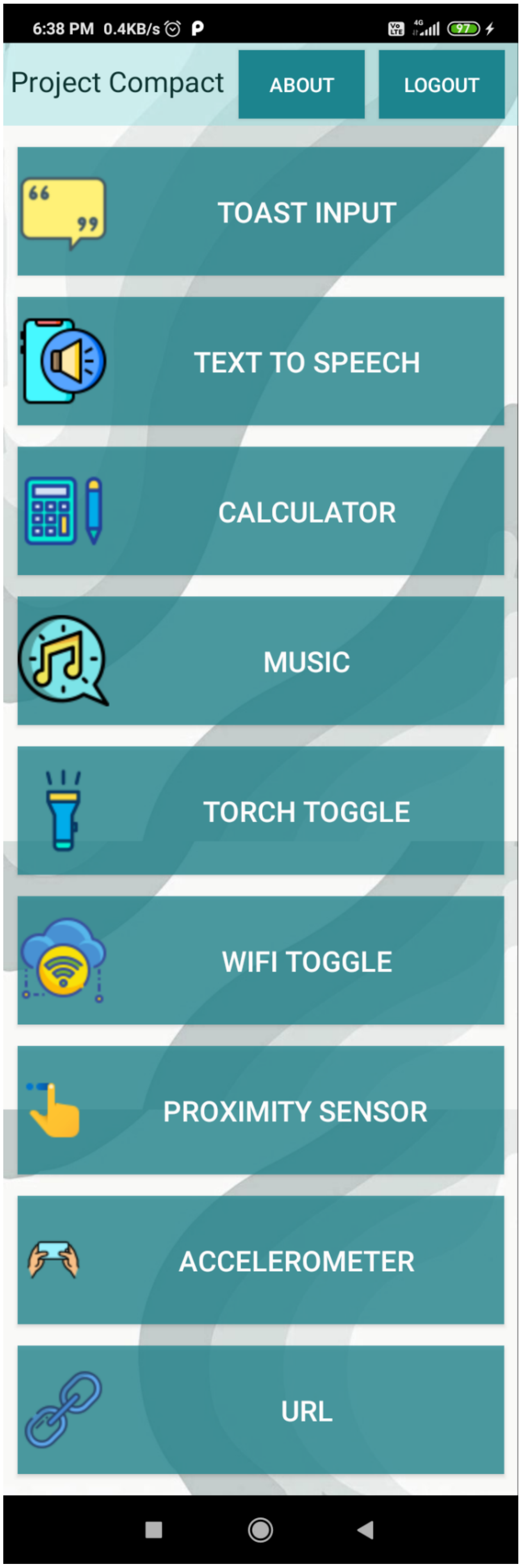
Functions - logout button at the top should sign out the current user from the app and directs back to the login page. The About button will show the user data fetched from firebase realtime database by directing to the about user page. Every button in the vertical scroll view directs the user to different activities.

Precautions - layout should be properly managed. It should be interactive on every screen size.

Each & every button should work properly and directs user to required activity.

No runtime exceptions should occur, which can cause crashes.

Every tool activity page of our app must have a back button to get back to the Fourth page.



ABOUT - goto user data page (show & update)

LOGOUT - logout user from firebase authentication

TOAST INPUT - small floating text appears on the bottom of the screen, which will show the input text.

TEXT TO SPEECH - convert user input text into speech.

CALCULATOR - a basic calculator, supports floating point numbers and speaks the answer.

MUSIC - plays music, have stop option and pause option also.

TORCH TOGGLE - on/off mobile phone's torch

WIFI TOGGLE - on/off mobile wifi, can determine current state of wifi in phone.

PROXIMITY SENSOR - sense the amount of light intensity coming on sensor and plays a song when gets covered or in dark.

ACCELEROMETER - Show a green bar if mobile is at rest facing upwards with x,y,z coordinates.

URL - directs user either to a given url with default browser or on shortcuts displayed on screen.

Fourth.java

```java
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;


public class Fourth extends AppCompatActivity

{

   Button logout, about;

   Button toast_input, text_to_speech, calculator, music, torch;

   Button wifi, proximity, accelerometer, url;

   @Override

   protected void onCreate(Bundle savedInstanceState)

   {

       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_fourth);

       logout = findViewById(R.id.button6);

       about = findViewById(R.id.button7);

       toast_input = findViewById(R.id.materialButton);

       text_to_speech = findViewById(R.id.materialButton2);

       calculator = findViewById(R.id.materialButton3);

       music = findViewById(R.id.materialButton4);

       torch = findViewById(R.id.materialButton5);

       wifi = findViewById(R.id.materialButton6);

       proximity = findViewById(R.id.materialButton8);

       accelerometer = findViewById(R.id.materialButton9);

       url = findViewById(R.id.materialButton10);
```

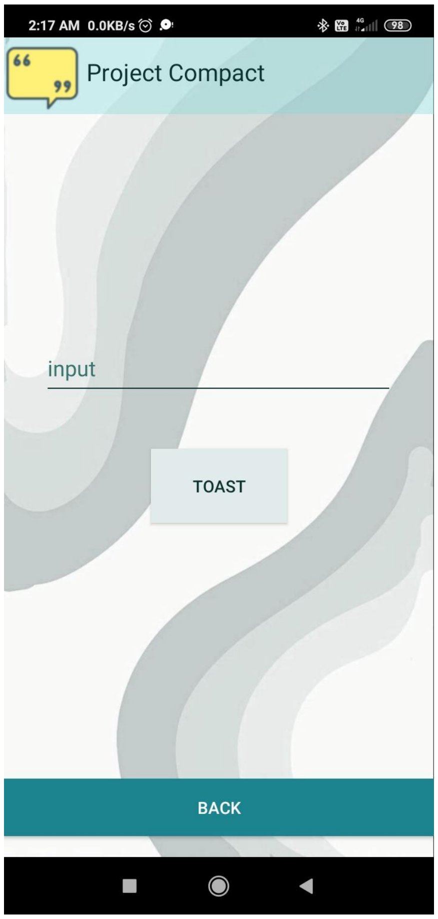
Task of every button in this activity is to direct the app to some other page logout button

```
        logout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MainActivity.fba.signOut();
                Intent i = new Intent(Fourth.this, MainActivity.class);
                startActivity(i); finish();
            }
        });
```

about button

```
        about.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Third.class);
                startActivity(i); finish();
            }
        });

        // app functional work intents
        toast_input.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, ToastInput.class);
                startActivity(i); finish();
            }
        });
        text_to_speech.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, TextSpeech.class);
                startActivity(i); finish();
            }
        });
        calculator.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Calculator.class);
                startActivity(i); finish();
            }
        });
        music.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Music.class);
                startActivity(i); finish();
            }
        });
        torch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Torch.class);
                startActivity(i); finish();
            }
        });
        wifi.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Wifi.class);
                startActivity(i); finish();
            }
        });
        proximity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Proximity.class);
                startActivity(i); finish();
            }
        });
        accelerometer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Accelerometer.class);
                startActivity(i); finish();
            }
        });
        url.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Fourth.this, Url.class);
                startActivity(i); finish();
            }
        });
    }
}
```

PROJECT TOOLS ACTIVITIES PAGES



Toast input page - show a small floating text on bottom of the screen with text given by the user by clicking on a button.

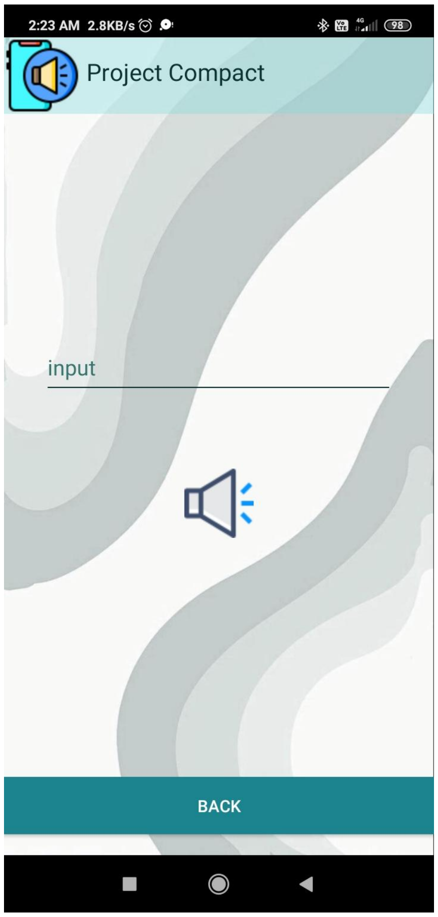
Design - one edit text for user input, one toast button, one back button

Functions - by clicking on the toast button, a toast message should appear with given input as its text. Back button should direct the user to the Fourth page of the app.

Precautions - toast should be visible to the user with proper background and text contrast. App should not crash at any stage.

ToastInput.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class ToastInput extends AppCompatActivity
{
   Button back, toast;
   EditText et;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_toast_input);
       back = findViewById(R.id.button9);
       toast = findViewById(R.id.button10);
       et = findViewById(R.id.editText);
       toast.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
      Toast.makeText(ToastInput.this, et.getText().toString(), Toast.LENGTH_SHORT).show();
           }
       });
       back.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               Intent i = new Intent(ToastInput.this, Fourth.class);
               startActivity(i); finish();
           }
       });
   }
}
```



Text to speech page - converts the user text input into speech by clicking on an image button.

Design - one edit text for user input, on image button with speaker icon and a back button.

Functions - by clicking on the image button the system should speak the text given by the user in edit text. Back button should direct the user to the Fourth page of the app.

Precautions - sound should not be very low and not either very fast or very slow. App should not crash at any stage.

TextSpeech.java

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.speech.tts.TextToSpeech;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageButton;

import java.util.Locale;


public class TextSpeech extends AppCompatActivity

{

   Button back;

   ImageButton speak;

   EditText et;

   TextToSpeech tts;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_text_speech);
        back = findViewById(R.id.button11);
        speak = findViewById(R.id.imageButton);
        et = findViewById(R.id.editText2);
```

public class **TextToSpeech**

Synthesizes speech from text for immediate playback or to create a sound file.

constructor - public **TextToSpeech(Context** context, **TextToSpeech.OnInitListener** listener)

The constructor for the TextToSpeech class, using the default TTS engine. This will initialize the associated TextToSpeech engine if it isn't already running.

OnInitListener is an inner interface in **TextToSpeech** class.

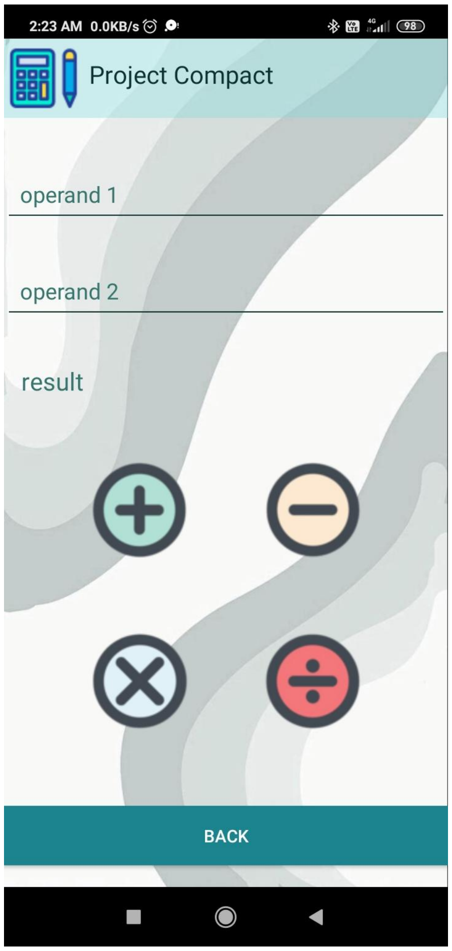
    public abstract void **onInit(**int status**)**

```
        tts = new TextToSpeech(this, new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int i) {
                if(i!=TextToSpeech.ERROR) { int ERROR denotes a generic operation failure
                    tts.setLanguage(Locale.ENGLISH); Sets the text-to-speech language. The
TTS engine will try to use the closest match to the specified language as represented by the
Locale
                    tts.setSpeechRate(0.9f); float: Speech rate. 1.0 is the normal speech
rate, lower values slow down t he speech & greater values accelerate it
                }
            }
        });
        speak.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                tts.speak(et.getText().toString(), TextToSpeech.QUEUE_FLUSH, null);
```

public int **speak(**CharSequence text,int queueMode,Bundle params,String utteranceId**)**

Public static final int **QUEUE_FLUSH**, Queue mode where all entries in the playback queue (media to be played and text to be synthesized) are dropped and replaced by the new entry. Queues are flushed with respect to a given calling app. Entries in the queue from other callees are not discarded.

```
            }
        });
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(TextSpeech.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }
}
```



Calculator page - a simple calculator which takes only two operands and does add, subtract, multiply or divide. The result obtained should be converted to speech also.

Design - two edit text for user operands input, one text view to display result, four image buttons displaying (+, -, x, ÷) and a back button.

Functions - do specified calculations with operands. Speaks the floating result. Back button should direct the user to the Fourth page of the app.

Precautions - calculations should be correct upto decimals. Sound should not be very low and not either very fast or very slow. App should not crash at any stage.

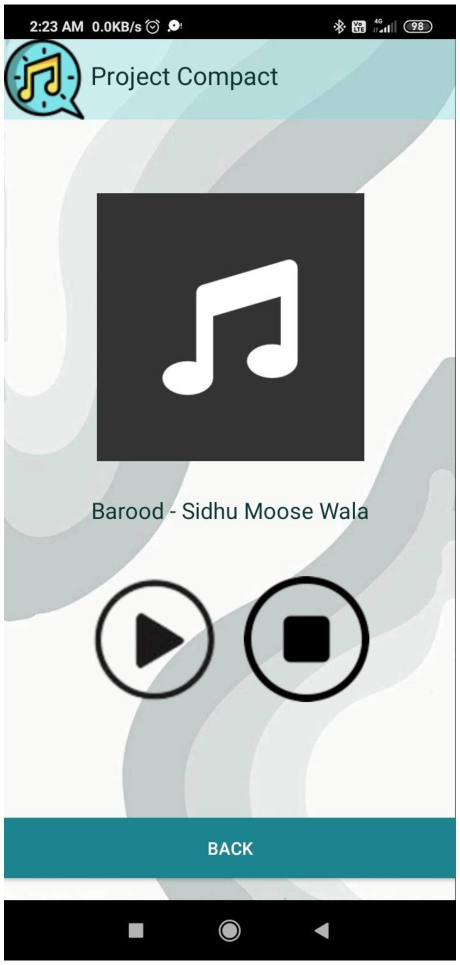
Calculator.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;
import android.widget.Button;
```

```java
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;
import java.util.Locale;

public class Calculator extends AppCompatActivity {
   Button back;
   EditText opd1, opd2;
   TextView res;
   ImageButton add, sub, mul, div;
   TextToSpeech tts;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calculator);
        back = findViewById(R.id.button12);
        opd1 = findViewById(R.id.editText3);
        opd2 = findViewById(R.id.editText4);
        res = findViewById(R.id.textView5);
        add = findViewById(R.id.imageButton2);
        sub = findViewById(R.id.imageButton3);
        mul = findViewById(R.id.imageButton4);
        div = findViewById(R.id.imageButton5);
        tts = new TextToSpeech(this, new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int i) {
                if(i!=TextToSpeech.ERROR) {
                    tts.setLanguage(Locale.ENGLISH);
                    tts.setSpeechRate(0.9f);
                }
            }
        });
        add.setOnClickListener(view -> {
            try {
```

```
            String s1 = opd1.getText().toString();
            String s2 = opd2.getText().toString();
            Float f = Float.valueOf(s1) + Float.valueOf(s2);
            String rs = Float.toString(f);
            res.setText(rs);
            tts.speak(rs,TextToSpeech.QUEUE_FLUSH,null,null);
        }catch(Exception e)
        {
           Toast.makeText(Calculator.this, "invalid input", Toast.LENGTH_SHORT).show();
            opd1.setText("");
            opd2.setText("");
        }
    });

    sub.setOnClickListener(view -> {
        try {
            String s1 = opd1.getText().toString();
            String s2 = opd2.getText().toString();
            Float f = Float.valueOf(s1) - Float.valueOf(s2);
            String rs = Float.toString(f);
            res.setText(rs);
            tts.speak(rs,TextToSpeech.QUEUE_FLUSH,null,null);
        }catch(Exception e)
        {
           Toast.makeText(Calculator.this, "invalid input", Toast.LENGTH_SHORT).show();
            opd1.setText("");
            opd2.setText("");
        }
    });

    mul.setOnClickListener(view -> {
        try {
            String s1 = opd1.getText().toString();
            String s2 = opd2.getText().toString();
            Float f = Float.valueOf(s1) * Float.valueOf(s2);
```

```
                String rs = Float.toString(f);
                res.setText(rs);
                tts.speak(rs,TextToSpeech.QUEUE_FLUSH,null,null);
            }catch(Exception e)
            {
               Toast.makeText(Calculator.this, "invalid input", Toast.LENGTH_SHORT).show();
                opd1.setText("");
                opd2.setText("");
            }
        });

        div.setOnClickListener(view -> {
            try {
                String s1 = opd1.getText().toString();
                String s2 = opd2.getText().toString();
                Float f = Float.valueOf(s1) / Float.valueOf(s2);
                String rs = Float.toString(f);
                res.setText(rs);
                tts.speak(rs,TextToSpeech.QUEUE_FLUSH,null,null);
            }catch(Exception e) {
               Toast.makeText(Calculator.this, "invalid input", Toast.LENGTH_SHORT).show();
                opd1.setText("");
                opd2.setText("");
            }
        });

        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Calculator.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }
}
```



Music page - play a song when the play button gets clicked. This song will stop on activity focus loss.

Design - an image for a good looking UI, two image buttons for play and stop, text view to display info about the song below the image and a back button.

Functions - when the play button gets clicked, a song starts playing and that play button image changes from play to pause. When we click that button again the song should pause. Song should completely stop by clicking on the stop button. Back button should direct the user to the Fourth page of the app.

Precautions - on clicking the pause button then play again, song should start playing where it was paused. Image of the play/pause button should change correctly. App should not crash at any stage.

Create a new android resource directory named **raw** in **Android > app > res**, paste the required songs in this. We can access this song as **R.raw.song_name** (first letter of the song name should be small)

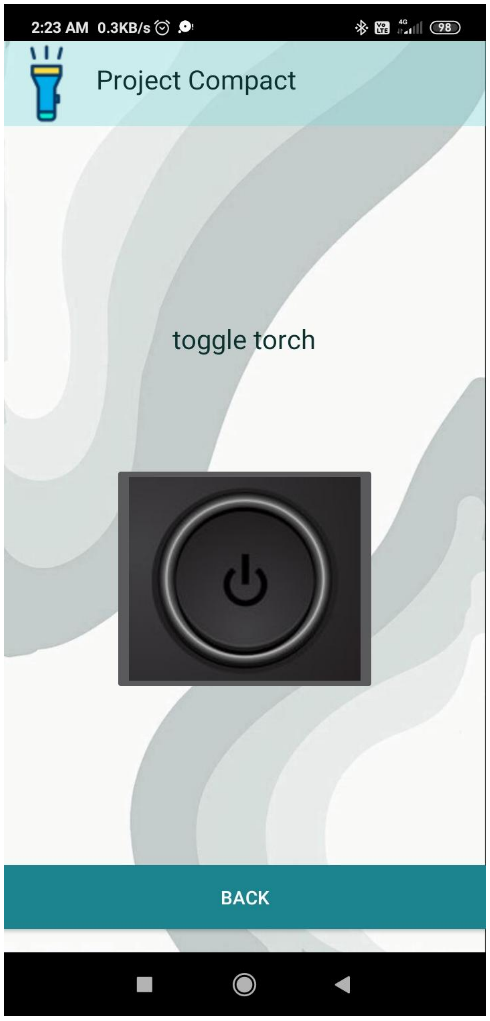
Music.java

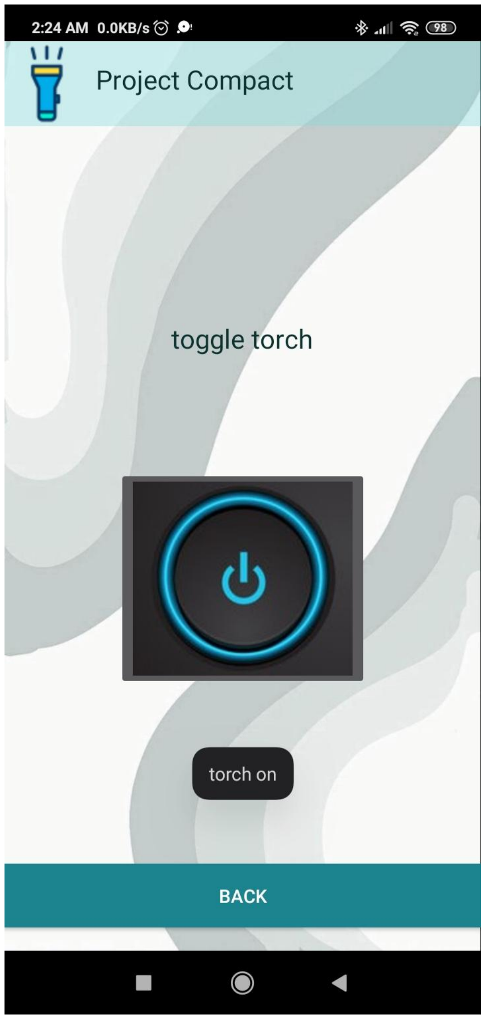
```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;

public class Music extends AppCompatActivity
{
   Button back;
   ImageButton play_pause, stop;
   MediaPlayer mp;  public class MediaPlayer, can be used to control playback of
audio/video files and streams.
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_music);
    back = findViewById(R.id.button13);
    play_pause = findViewById(R.id.imageButton6);
    stop = findViewById(R.id.imageButton7);

    play_pause.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if(mp!=null)
                if(mp.isPlaying())
                    pause();
                else
                    play();
            else
                play();
        }
    });
    stop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            stop();
        }
    });

    back.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent i = new Intent(Music.this, Fourth.class);
            startActivity(i); finish();
        }
    });
}
```

```
    void play() {
        if(mp==null) mp = MediaPlayer.create(this,R.raw.barood);
                    Static MediaPlayer create(Context context, int resid)
        mp.start();  start or resume playback
        play_pause.setImageResource(R.drawable.pause1); changing image to pause image


public void setOnCompletionListener (MediaPlayer.OnCompletionListener listener)

Register a callback to be invoked when the end of a media source has been reached.

public static interface MediaPlayer.OnCompletionListener

      abstract void onCompletion (MediaPlayer mp), called the end of media player source

        mp.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mediaPlayer) {
                stop(); song should stop after completion
            }
        });
    }

    void pause() {
        if(mp!=null)
        {
            mp.pause();  pause playback
            play_pause.setImageResource(R.drawable.play1); changing image to play image
        }
    }
    void stop() {
        if(mp!=null)
        {
            mp.stop();  stops playback after playback has been started or paused
            mp.release(); release resources associated with its MediaPlayer object
            mp=null;
            play_pause.setImageResource(R.drawable.play1); changing image to play image
        }
    }
    @Override
    protected void onPause() {
        super.onPause();
        pause(); pause on activity focus loss, also due to this song will not gets played on
activity destroy. Example - if home button is pressed by the user on mobile, then the song
should pause only not stop, but stop when activity is removed bcs of finish() call.
    }
}
```

Torch toggle page - on/off mobile torch with an interactive button.





Design - an image button to show on/off image while torch toggle and a back button.

Functions - toggle torch on clicking the image button with a toast giving info about torch state. Back button should direct the user to the Fourth page of the app.

Precautions - image buttons should work correctly according to torch state. App should not crash at any stage.

Torch.java

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.hardware.camera2.CameraAccessException;

import android.hardware.camera2.CameraManager;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.ImageButton;

import android.widget.Toast;


public class Torch extends AppCompatActivity {

   ImageButton ib;

   Button back;

   CameraManager cm;
```

public final class **CameraManager**, a system service manager for detecting, characterizing and connecting to camera devices.

```
   boolean torch_togg;

   @Override

   protected void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_torch);

       back = findViewById(R.id.button14);

       ib = findViewById(R.id.imageButton8);


       cm = (CameraManager) getSystemService(CAMERA_SERVICE);
```

public abstract **Object getSystemService(**String name**)** is an abstract method in public abstract class **Context.** This method returns the handle to a system-level service by class.

Context also has many string constants, here we will work with camera service constant public static final **String CAMERA_SERVICE**, we used it with **getSystemService()** to retrieve a **CameraManager** for interacting with camera devices.

Although **getSystemService()** is abstract in **Context** class, but it is implemented in some other class by android itself which is then indirectly extended by **AppCompatActivity** which we are using in our every activity.

```
java.lang.Object
   ↳ android.content.Context
       ↳ android.content.ContextWrapper
           ↳ android.view.ContextThemeWrapper
               ↳ android.app.Activity
                   ↳ androidx.activity.ComponentActivity
                       ↳ androidx.fragment.app.FragmentActivity
                           ↳ androidx.appcompat.app.AppCompatActivity
```

```
        ib.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                try {
                    String str = cm.getCameraIdList()[0];
```

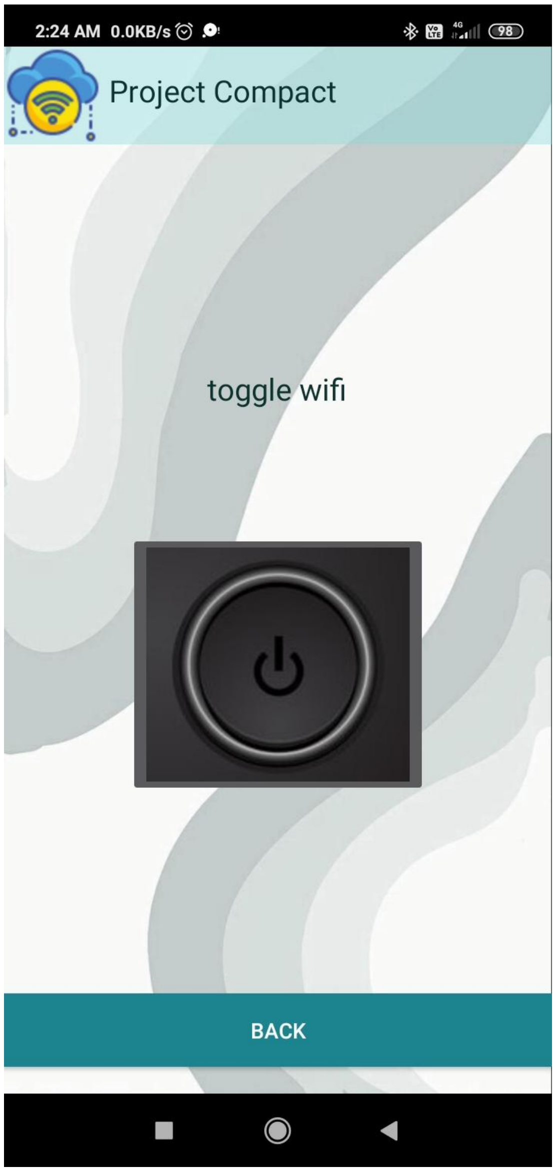
public String[] **getCameraIdList()** returns the list of currently connected camera devices by using identifiers, including cameras that may be in use by other camera API clients. Non-removable cameras use integers starting at 0 for their identifiers, while removable cameras have a unique identifier for each individual device, even if they are the same model.

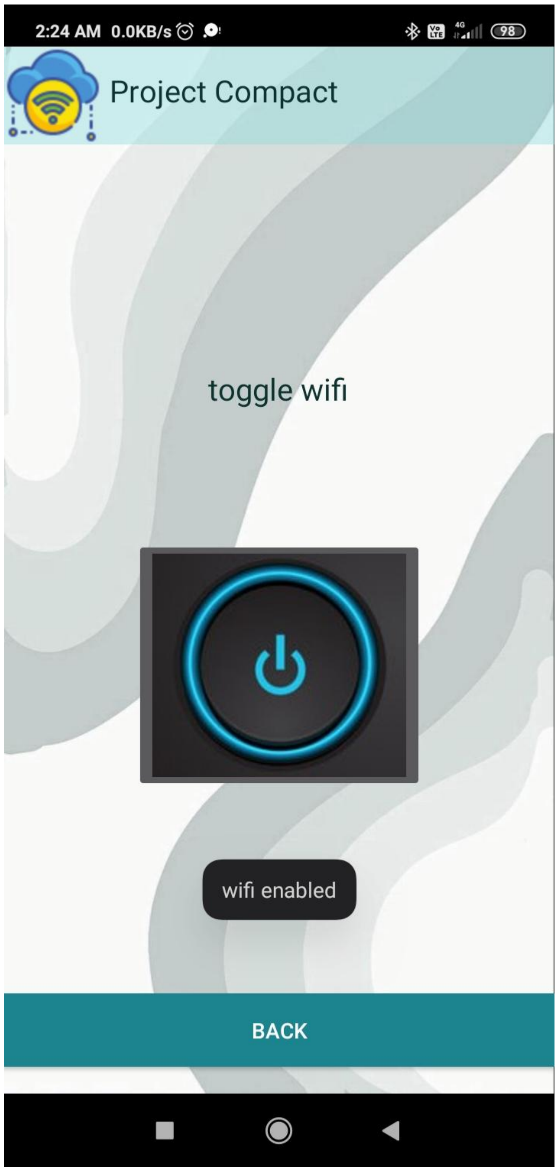
```
                    if(!torch_togg) {
```

public void **setTorchMode(**String cameraId, boolean enabled**)**, Set the flash unit's torch mode of the camera of the given ID without opening the camera device.

```
                        cm.setTorchMode(str,true);
                        torch_togg = true;
                        ib.setImageResource(R.drawable.on);
                        Toast.makeText(Torch.this, "torch on", Toast.LENGTH_SHORT).show();
                    }
                    else {
                        cm.setTorchMode(str,false);
                        torch_togg = false;
                        ib.setImageResource(R.drawable.off);
                        Toast.makeText(Torch.this, "torch off", Toast.LENGTH_SHORT).show();
                    }
                }
                catch (CameraAccessException cae)
                {
             Toast.makeText(Torch.this, "something went wrong", Toast.LENGTH_SHORT).show();
                }
            }
        });
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Torch.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }
}
```

Wifi toggle page - on/off mobile wifi with an interactive button.





Design - an image button to show on/off image while wifi toggle and a back button.

Functions - toggle wifi on clicking the image button with a toast giving info about wifi state. Back button should direct the user to the Fourth page of the app.

Precautions - image buttons should work correctly according to current wifi state. App should not crash at any stage.

In AndroidManifest.xml :-

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="com.example.projectcompact">

   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

   <application . . .
```

Wifi.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.Toast;

public class Wifi extends AppCompatActivity {
   ImageButton ib;
   Button back;
   boolean wifi_togg;

   WifiManager wm;  public class WifiManager, provides the primary API for managing all
aspects of WI-FI connectivity.
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_wifi);
        ib = findViewById(R.id.imageButton9);
        back = findViewById(R.id.button15);
```

public abstract **Context getApplicationContext()** present in the abstract class **Context** which is implemented by android in **Activity.** Return the context of the single, global Application object of the current process. This generally should only be used if you need a Context whose lifecycle is separate from the current context, that is tied to the lifetime of the process rather than the current component.

```
        wm = (WifiManager)getApplicationContext().getSystemService(WIFI_SERVICE);
```

public static final String **WIFI_SERVICE,** Use with **getSystemService(**java.lang.String**)** to retrieve a **WifiManager** for handling management of Wi-Fi access.

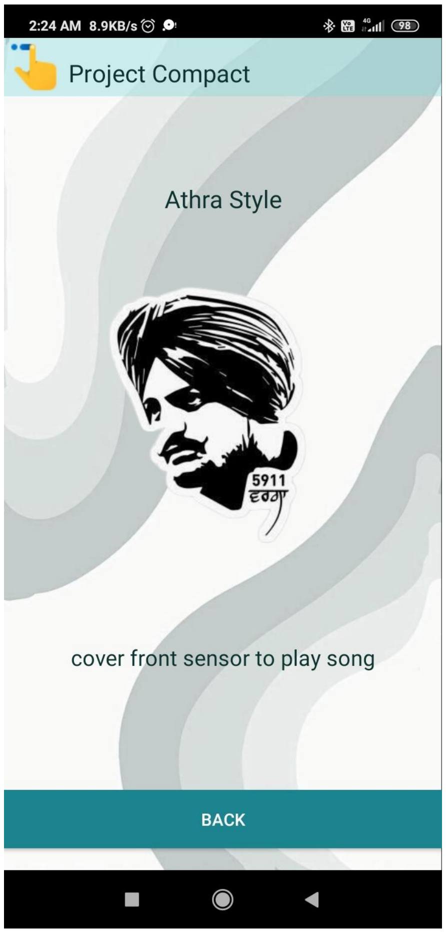
```
        ib.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(!wifi_togg)
                {
                    wm.setWifiEnabled(true);  enables wifi
                    wifi_togg = true;
                    ib.setImageResource(R.drawable.on);
                    Toast.makeText(Wifi.this, "wifi enabled", Toast.LENGTH_SHORT).show();
                }
                else
                {
                    wm.setWifiEnabled(false);  disables wifi
                    wifi_togg = false;
                    ib.setImageResource(R.drawable.off);
                    Toast.makeText(Wifi.this, "wifi disabled", Toast.LENGTH_SHORT).show();
                }
            }
        });
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Wifi.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }
    @Override to check if wifi is already enabled or not,
    protected void onResume() {
        super.onResume();
        if(wm.isWifiEnabled()) checking the current state of wifi
        {
            wifi_togg = true;
            ib.setImageResource(R.drawable.on);
        }
        else
        {
            wifi_togg = false;
            ib.setImageResource(R.drawable.off);
        }
    }
}
```

About contexts in android

Toast and **Intent**, both require reference to *context*. And **getApplication, getApplicationContext, LoginActivity.this** and **getBaseContext**, they all offer reference to the context.
To make things simple, we should count two types of context available in the Android framework.

1. Application Context
2. Activity Context

Application context is attached to the application's life-cycle and will always be same throughout the life of the application. So if we are using *Toast*, we can use application context or even activity context (both) because a toast can be raised from anywhere within our application and is not attached to a window.
Activity context is attached to the Activity's life-cycle and can be destroyed if the activity's onDestroy() is raised. If we want to launch a new activity, we must need to use the activity's context in its *Intent* so that the new launching activity is connected to the current activity (in terms of activity stack). However, we may use application's context too to launch a new activity but then we need to set flag Intent.FLAG_ACTIVITY_NEW_TASK in intent to treat it as a new task.



Proximity Sensor page - plays a song in dark or when proximity sensor (in front camera) gets covered.

Design - just simple text info to guide the user what this activity does, and a button to go back.

Functions - play a song when the sensor is in dark, pause in light and stop when the song gets completed. But starts playing again on sensor darkness ration change. Back button should direct the user to the Fourth page of the app.

Precautions - sensor should work correctly and song should not start or end abruptly. Song should pause when activity loses focus and sensor should not work at that time, otherwise it can cause clashing of two songs. App should not crash at any stage.

Proximity.java

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.hardware.Sensor;

import android.hardware.SensorEvent;

import android.hardware.SensorEventListener;

import android.hardware.SensorManager;

import android.media.MediaPlayer;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;


public class Proximity extends AppCompatActivity implements SensorEventListener {

   Button back;

   SensorManager sm;  public abstract class SensorManager, let us access device's sensors

   Sensor s;  public final class Sensor, represents a sensor

   MediaPlayer mp;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proximity);
        back = findViewById(R.id.button17);

        sm = (SensorManager)getSystemService(SENSOR_SERVICE); get system service

        s = sm.getDefaultSensor(Sensor.TYPE_LIGHT);
```

public sensor **getDefaultSensor(**int type**)**, use this method to get the default sensor for a given type. Note that the returned sensor could be a composite sensor, and its data could be averaged or filtered. If you need to access the raw sensors use getSensorList.

Sensor have many constants, one of them is public static final int **TYPE_LIGHT.**

```
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Proximity.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }

    void play()
    {
        if(mp==null) mp = MediaPlayer.create(this,R.raw.athrastyle);
        mp.start();
        mp.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mediaPlayer) {
                stop();
            }
        });
    }
```

```
   void pause()
   {
       if(mp!=null)
       {
           mp.pause();
       }
   }
   void stop()
   {
       if(mp!=null)
       {
           mp.stop();
           mp.release();
           mp=null;
       }
   }
```

Public interface **SensorEventListener,** used for receiving notifications from the SensorManager when there is new sensor data. It has two abstract methods that need to be overridden. void **onSensorChanged(**Sensor sensor, int accuracy**)** & void **onAccuracy**

```
   @Override
   public void onSensorChanged(SensorEvent sensorEvent) {
```

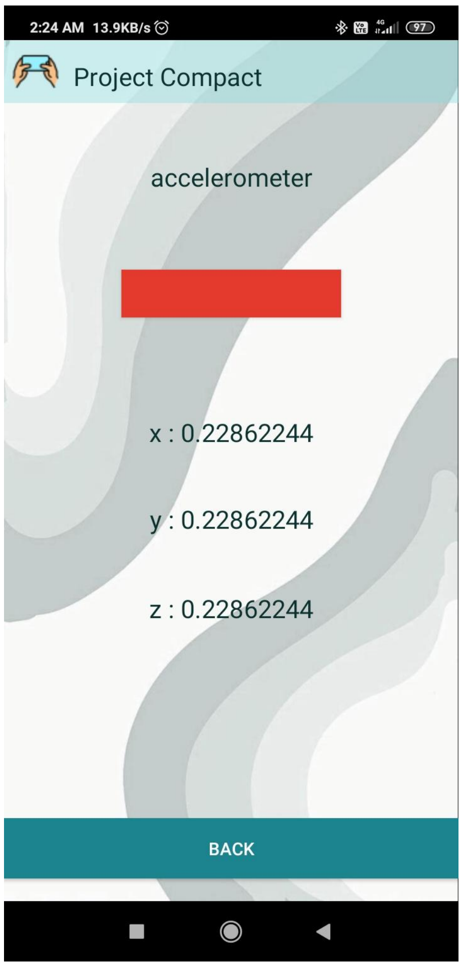
public final float[] **values,** the length and contents of the **values** array depends on which **Sensor** type is being monitored.

Sensor.TYPE_LIGHT :- values[0]: Ambient light level in SI lux units

```
       if(sensorEvent.values[0]>1) check light intensity
           pause();
       else
           play();
   }
   @Override
   public void onAccuracyChanged(Sensor sensor, int i) {


   }
```

```
    @Override
    protected void onPause() { when activity loses focus
        super.onPause();
        pause();
        sm.unregisterListener(this);  public void unregisterListener(SensorListener lsnr)
              Unregisters listener for all sensor
    }
    @Override
    protected void onResume() { when activity resumes
        super.onResume();
        sm.registerListener(this, s, SensorManager.SENSOR_DELAY_NORMAL);
boolean registerListener (SensorEventListener listener, Sensor s, int sampelingPeriodUs)
Registers a SensorEventListener for the given sensor at the given sampling frequency.
Here, public static final int SENSOR_DELAY_NORMAL is suitable for sensor changes
    }
}
```

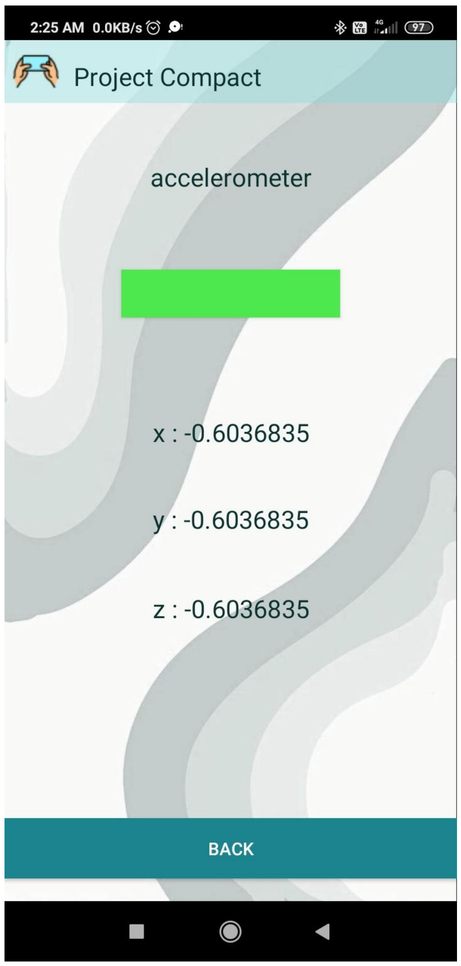
Accelerometer page - show device's coordinates in space & indicates when rest on a flat surface.



x : 0.22862244

y : 0.22862244

z : 0.22862244



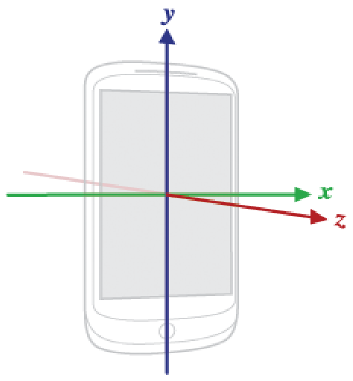
x : -0.6036835

y : -0.6036835

z : -0.6036835

An <u>accelerometer</u> is a device that measures the vibration, or acceleration of motion of a structure.

In electrical terms, an accelerometer is an electronic sensor that measures the acceleration forces acting on an object, in order to determine the object's position in space and monitor the object's movement.



Design - an image changing red/green, three text views showing x, y & z axis position of mobile phone in space and a back button.

Functions - this activity shows change in position of the user's device in space. The small rectangle image turns green when the mobile phone is placed on a flat surface. Back button should direct the user to the Fourth page of the app.

Precautions - accelerometer also measures acceleration due to gravity (9.8), so we need to take care about it while measuring phone coordinates when at rest. App should not crash at any stage.

Accelerometer.java

```java
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.hardware.Sensor;

import android.hardware.SensorEvent;

import android.hardware.SensorEventListener;

import android.hardware.SensorManager;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;


public class Accelerometer extends AppCompatActivity implements SensorEventListener {

   TextView x,y,z;

   Button acc,back;

   SensorManager sm;

   Sensor s;

   @Override

   protected void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_accelerometer);

       x = findViewById(R.id.textView12);

       y = findViewById(R.id.textView13);

       z = findViewById(R.id.textView14);

       back = findViewById(R.id.button18);

       acc = findViewById(R.id.button19);
```

```
        sm = (SensorManager)getSystemService(SENSOR_SERVICE);
        s = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  accelerometer sensor
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(Accelerometer.this, Fourth.class);
                startActivity(i); finish();
            }
        });
    }
    @Override
    public void onSensorChanged(SensorEvent sensorEvent)
    {
             x-y-z coordinate values
        int xi = (int)sensorEvent.values[0];
        int yi = (int)sensorEvent.values[1];
        int zi = (int)sensorEvent.values[2];
        x.setText("x : "+sensorEvent.values[0]);
        y.setText("y : "+sensorEvent.values[0]);
        z.setText("z : "+sensorEvent.values[0]);
        if(xi==0 && yi==0 && zi==9)
            acc.setBackgroundResource(R.color.acct);
        else
            acc.setBackgroundResource(R.color.accf);
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {


    }


    @Override
    protected void onPause() {
        super.onPause();
        sm.unregisterListener(this);
    }
    @Override
    protected void onResume() {
        super.onResume();
        sm.registerListener(this, s, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
```

SensorEvent values[] for **Sensor.TYPE_ACCELEROMETER:**

All values are in SI units (m/s^2)

- **values[0]:** Acceleration minus Gx on the x-axis
- **values[1]:** Acceleration minus Gy on the y-axis
- **values[2]:** Acceleration minus Gz on the z-axis

A sensor of this type measures the acceleration applied to the device (Ad). Conceptually, it does so by measuring forces applied to the sensor itself (Fs) using the relation:

Ad = - ∑Fs / mass

In particular, the force of gravity is always influencing the measured acceleration:

Ad = -g - ∑F / mass

For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of g = 9.81 m/s^2

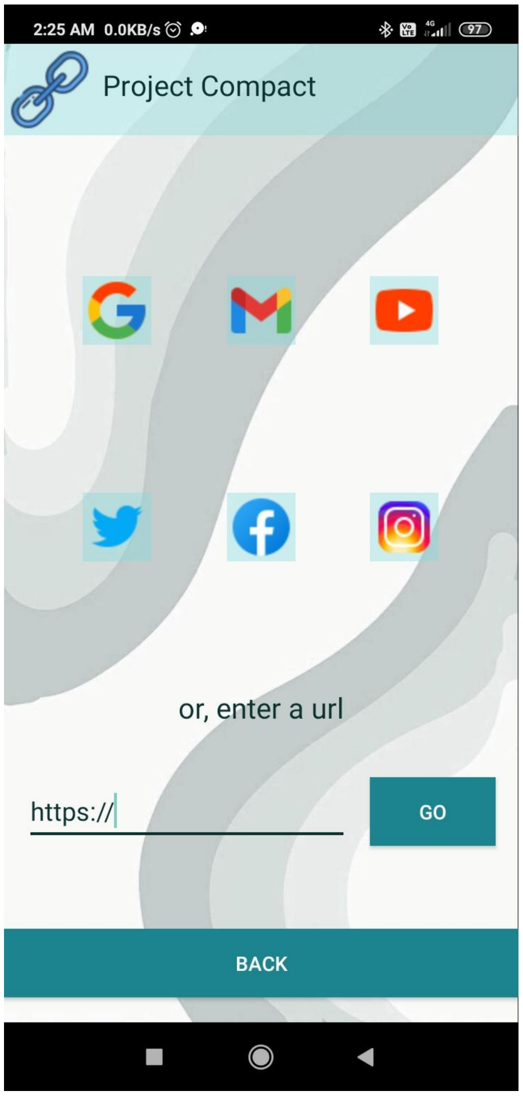
Similarly, when the device is in free-fall and therefore dangerously accelerating towards to ground at 9.81 m/s^2, its accelerometer reads a magnitude of 0 m/s^2.

URL page - navigate users to a url, either given by the user itself or from given icons in this activity.

Design - six image buttons with appropriate icons for common navigation like youtube, google, facebook, etc. One edit text partially filled with “https://” to get user input url, a go button for that url navigation and a back button.

Functions - each image button navigates the user to either a url through default browser or to the app (if exists in that device). Back button should direct the user to the Fourth page of the app.

Precautions - navigation to web or app should be correct according to image buttons. App should not crash at any stage.



- Google - `https://www.google.com/`
- Gmail - `https://www.mail.google.com/`
- Youtube - `https://www.youtube.com/`
- Twitter - `https://www.twitter.com/`
- Facebook - `https://www.facebook.com/`
- Instagram - `https://www.instagram.com/`

Url.java

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageButton;

import android.widget.Toast;

import java.net.URL;
```

```
public class Url extends AppCompatActivity {

   Button back, go;
   ImageButton google, gmail, youtube, twitter, facebook, instagram;
   EditText userInputUrl;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_url);

       back = findViewById(R.id.button20);
       go = findViewById(R.id.button21);
       google = findViewById(R.id.imageButton11);
       gmail = findViewById(R.id.imageButton15);
       youtube = findViewById(R.id.imageButton12);
       twitter = findViewById(R.id.imageButton13);
       facebook = findViewById(R.id.imageButton16);
       instagram = findViewById(R.id.imageButton14);
       userInputUrl = findViewById(R.id.editText5);

       google.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.google.com/");
           }
       });
       gmail.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.mail.google.com/");
           }
       });
       youtube.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.youtube.com/");
           }
       });
       twitter.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.twitter.com/");
           }
       });
       facebook.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.facebook.com/");
           }
       });
       instagram.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               gotoUrl("https://www.instagram.com/");
           }
       });

       go.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               String url = userInputUrl.getText().toString();
               try{
                   gotoUrl(url);  if url entered is in correct format
               }
               catch(Exception e)
               {
                   Toast.makeText(Url.this, "invalid url", Toast.LENGTH_SHORT).show();
               }
           }
       });
       back.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               Intent i = new Intent(Url.this, Fourth.class);
               startActivity(i); finish();
           }
       });
   }
   private void gotoUrl(String url)
   {
       Uri uri = Uri.parse(url);
```
public abstract class **Uri**

public static **Uri parse** (String uriString**)**, Creates a Uri which parses the given encoded URI string.

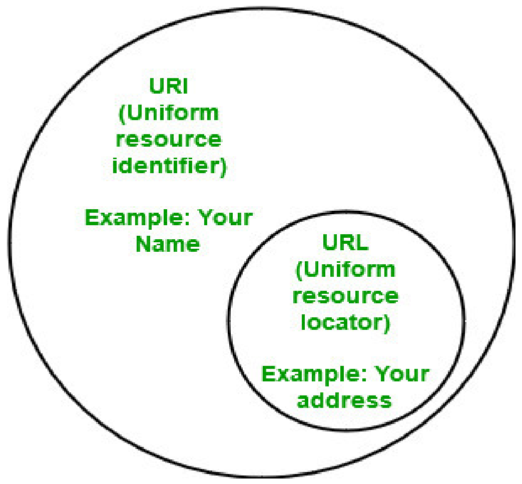
```
        startActivity(new Intent(Intent.ACTION_VIEW,uri));
```

public static final String **ACTION_VIEW**

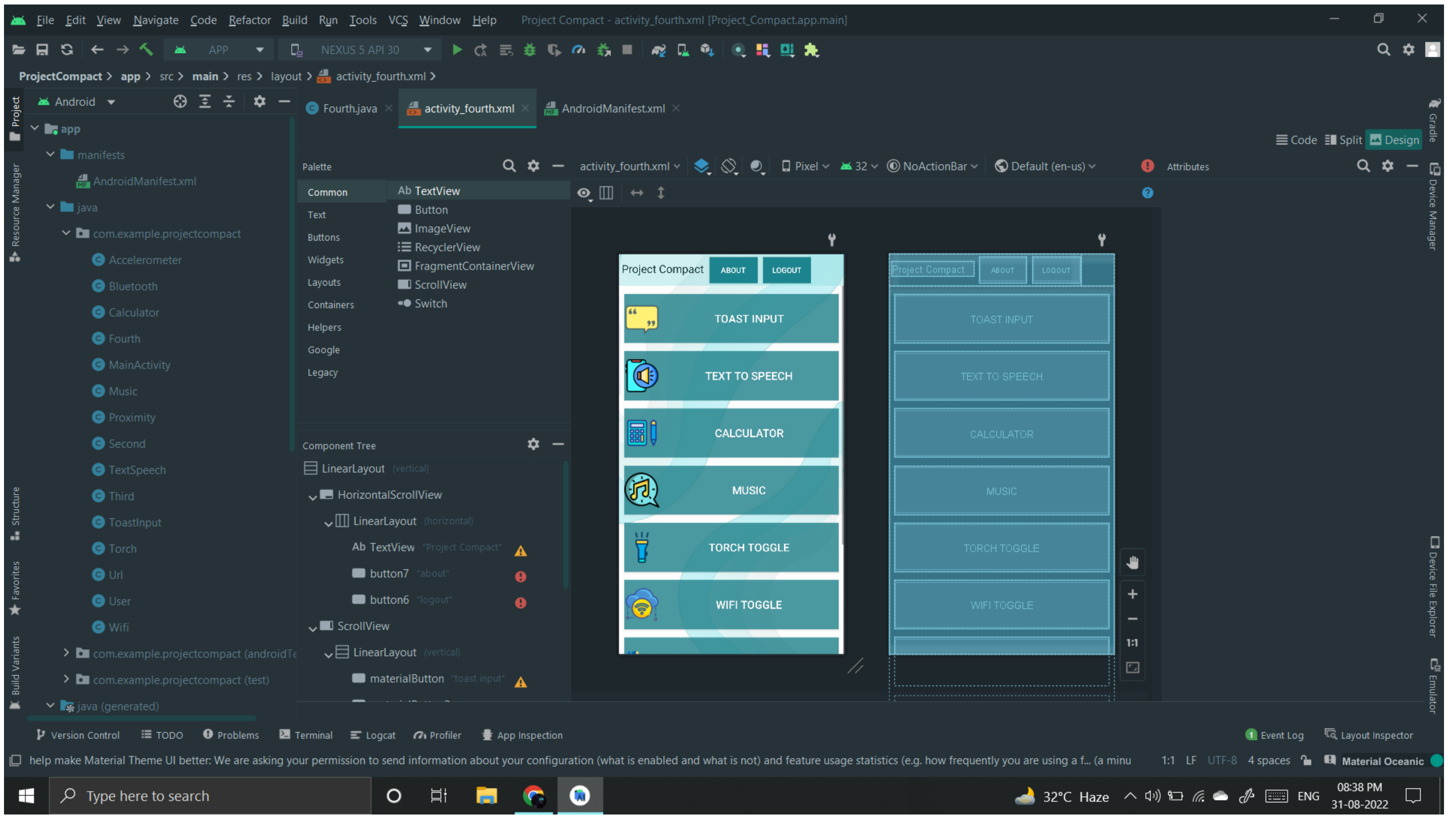
Activity Action: Display the data to the user. This is the most common action performed on data -- it is the generic action you can use on a piece of data to get the most reasonable thing to occur. For example, when used on a contacts entry it will view the entry; when used on a mailto: URI it will bring up a compose window filled with the information supplied by the URI; when used with a tel: URI it will invoke the dialer.

For example, the following code tells the android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter:

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.sidhu5911.com/"));
startActivity(i);

    }
}
```



URL is used to describe the identity of an item. **URI** provides a technique for defining the identity of an item. URL links a web page, a component of a web page or a program on a web page with the help of accessing methods like protocols. URI is used to distinguish one resource from another regardless of the method used.



help make Material Theme UI better: We are asking your permission to send information about your configuration (what is enabled and what is not) and feature usage statistics (e.g. how frequently you are using a f... (a minu