Installation Procedure for Android Studio

To create your new Android project, follow these steps:

1. Install the latest version of Android Studio.
    - https://developer.android.com
2. Once the installation procedure is complete,In the Welcome to Android Studio window, click Create New Project.

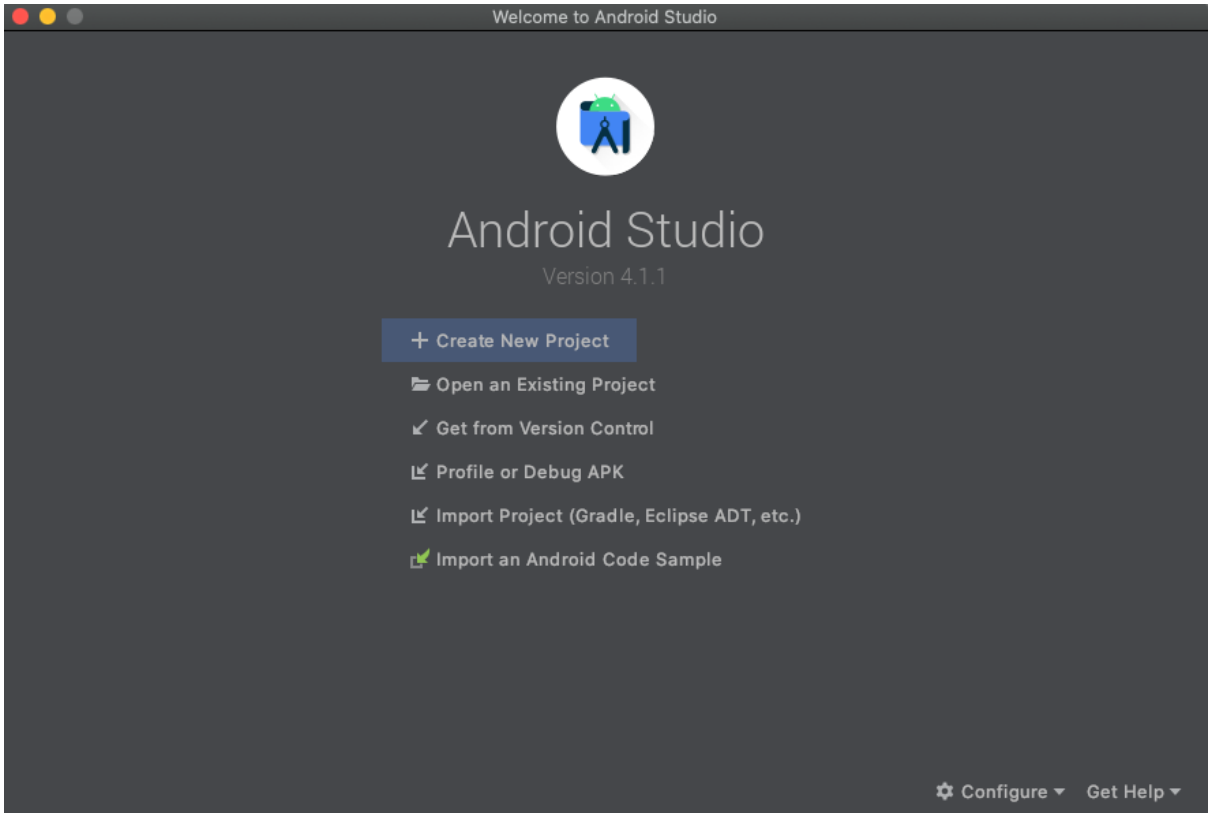


Figure 1. Android Studio welcome screen

If you have a project already opened, select File > New > New Project.

3. In the Select a Project Template window, select Empty Activity and click Next.
4. In the Configure your project window, complete the following:
    - Enter "My First App" in the Name field.
    - Enter "com.example.myfirstapp" in the Package name field.
    - If you'd like to place the project in a different folder, change its Save location.
    - Select either Java or Kotlin from the Language drop-down menu.
    - Select the lowest version of Android your app will support in the Minimum SDK field.
    - If your app will require legacy library support, mark the Use legacy android.support libraries checkbox.
    - Leave the other options as they are.
5. Click Finish.

After some processing time, the Android Studio main window appears.

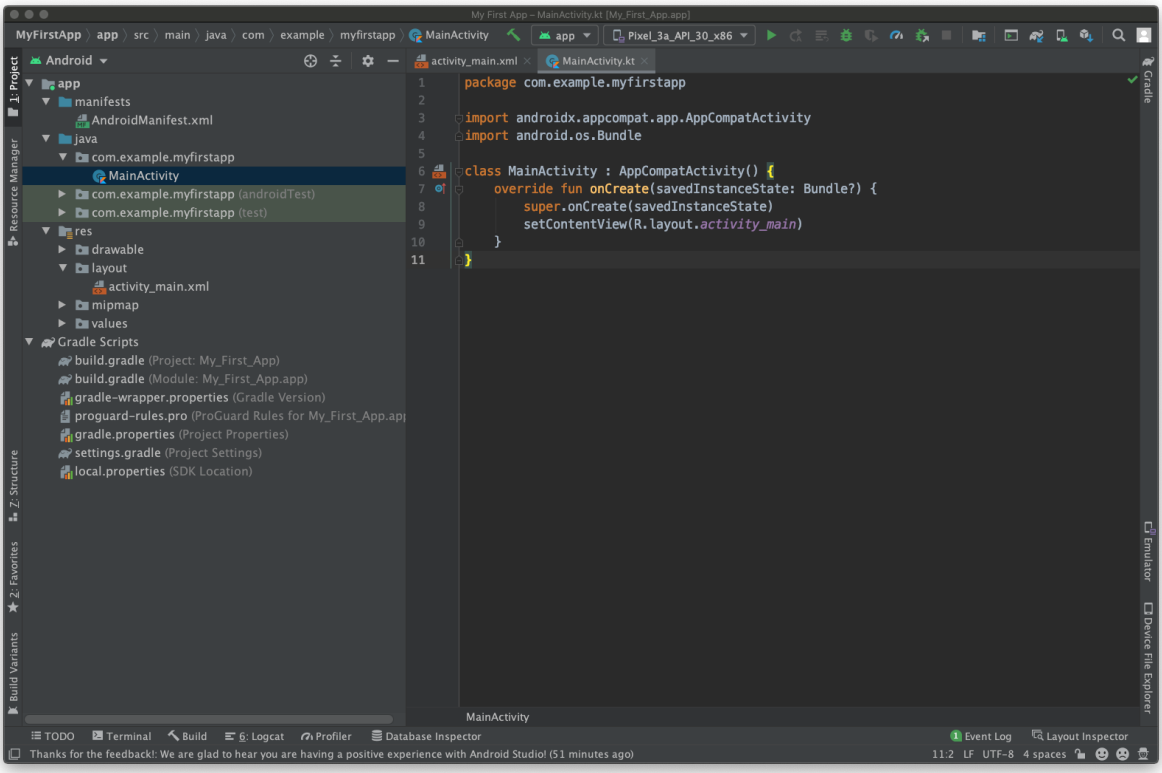


Figure 2. Android Studio main window

Now take a moment to review the most important files.

First, be sure the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

There are two files with this name: one for the project, "Project: My_First_App," and one for the app module, "Module: My_First_App.app." Each module has its own build.gradle file, but this project currently has just one module. Use each module's build.gradle file to control how the Gradle plugin builds your app. For more information about this file, see Configure your build.

To run the app, check if the AVD is enabled with one of the emulators to emulate your app for the selected device and build your application.

CHANGING TEXT ON CLICK USING ANDROID STUDIO

Ex.No: 1

Date: 02.08.2023

AIM:

To develop a simple Android application to change the text on click using Android Studio.

PROCEDURE:

1. Create an Android Studio project with the name “My Application Sample”.
2. Leave the default layout activity_main.xml as it is with text view as Hello World!
3. Open MainActivity.java. With the existing text view with id textView add OnClickListener to set the text as “Hello User!”
4. Now, run the application in the emulator for viewing the app.

PROGRAM:

activity_main.xml:

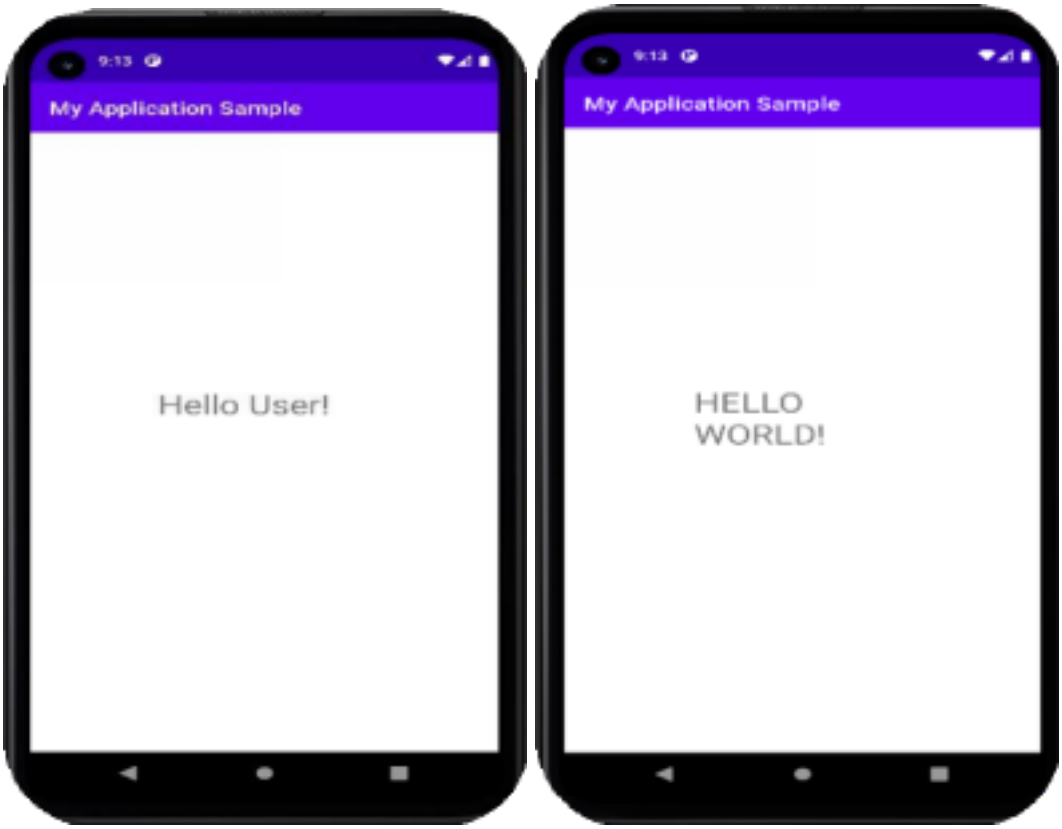
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent" tools:context=".MainActivity">

<TextView android:id="@+id/textView" android:layout_width="179dp"
android:layout_height="126dp" android:text="HELLO WORLD!"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java:

```
package com.example.myapplicationsample;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
        TextView t;
         @Override
                protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                t = findViewById(R.id.textView);
                t.setTextSize(30);
                t.setOnClickListener(new OnClickListener(){
                @Override
                public void onClick(View view)
                {
                        t.setText("Hello User!");
                }
                });
        }
}
```

OUTPUT:



RESULT:

Hence successfully developed and executed a simple Android application to change the text on click using Android Studio.

CHANGING TEXT COLOR, FONT SIZE AND STYLE IN ANDROID STUDIO

Ex No: 2

Date : 14-08-2023

AIM:

To develop a simple Android application to change the text color and font color, font size, font style on click using Android Studio.

PROCEDURE:

1. Create an Android Studio project with the name ex2.
2. Leave the default layout activity_main.xml as it is and drag and drop the following elements and add necessary constraints to the elements.
3. To change the font , font style and color of the text in the text view, Add and OnClicklistener() to each of the respective buttons. 4. Use setTextSize() to set the text size to a desired value. 5. Use setTextColor() to set the text color to a desired color. 6. Use setTypeface() to set the text font face to the desired font face. 7. Now, Run the application in the emulator for viewing the app.

PROGRAM:

MainActivity.java:

```
package com.example.ex2;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.graphics.Typeface;
import androidx.core.content.ContextCompat;

public class MainActivity extends AppCompatActivity {
   TextView t;
   Button b2;
   Button b3;
   Button b4;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
```

```
        t = findViewById(R.id.textView);
        b2 = findViewById(R.id.button2);
        b3 = findViewById(R.id.button3);
        b4 = findViewById(R.id.button4);

        b2.setOnClickListener( new OnClickListener() {
            @Override
            public void onClick(View view) {
                t.setTypeface(Typeface.defaultFromStyle(Typeface.BOLD_ITALIC));
            }
        });

        b3.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                t.setTextColor(ContextCompat.getColor(MainActivity.this,
android.R.color.holo_red_dark));
            }
        });

        b4.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                t.setText("FuckYou");
            }
        });
    }
}
```

Activity_main.xml

```
<?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"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="205dp"
        android:layout_height="62dp"
```

```
        android:layout_marginTop="68dp"
        android:text="HugYou"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.563"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="104dp"
        android:text="Button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.574"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button3" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="68dp"
        android:text="Button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.542"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.574"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button2"
        app:layout_constraintVertical_bias="0.424" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

OUTPUT:

RESULT:

Hence successfully developed and executed a simple Android application to change the text color, font size, font style on click using Android Studio.

DATABASE CONNECTIVITY IN ANDROID STUDIO

Ex.No: 3

Date: 30.08.2023

AIM:

To implement database connectivity with SQLite in Android Studio and create a simple DB application to add a person's name, location to the database and retrieve a person's location for the given name.

PROCEDURE:

1. Create an Android Studio project with the name "MyDatabase". 2. Leave the default layout activity_main.xml as it is and drag and drop three editText elements, two buttons and a TextView element. Add necessary constraints to the elements.
3. To establish connection with the sqlite database, create a "DB.java" class file, and import the necessary packages. Create an object of SQLiteDatabase and give the necessary parameters to make the database connection.
4. To insert the user data into the database, retrieve the user's database and display it in the textview, add the suitable OnClicklistener() to each of the respective buttons.
5. To insert the user data, use the ContentValues.put() method to add the student name and location entered in the two editText elements, and call sqLiteDatabase.insert() method to insert the added data into the table.
6. To retrieve the location for a particular student, create a cursor C, and use the sqLiteDatabase.query() method to execute a fetch query, passing the student name (taken from editText3) as the search condition.
7. Display the location obtained in step (6) in the TextView. 8. Now, run the application in the emulator for viewing the app.

PROGRAM

Main_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical"
   tools:context=".MainActivity" >

   <EditText
      android:id="@+id/name"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Name" />

<EditText
    android:id="@+id/loc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="location" />

<Button
    android:id="@+id/save"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="sav"
    android:text="SAVE" />

<EditText
    android:id="@+id/name2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Name" />

<Button
    android:id="@+id/retrieve"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Retrieve"
    android:onClick="retrieve"/>

<TextView
    android:id="@+id/location1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="" />

<EditText
    android:id="@+id/name3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Name" />

<Button
    android:id="@+id/delete"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Delete"
        android:onClick="retrieve"/>

    <TextView
        android:id="@+id/delete1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
```

MAINACTIVITY.JAVA

```
package com.example.db;

import androidx.appcompat.app.AppCompatActivity;

import android.widget.*;

import android.os.Bundle;

import android.view.*;


public class MainActivity extends AppCompatActivity {

    EditText name;

    EditText loc;

    EditText name2;

    EditText name3;

    Button save;

    Button retrieve;

    Button delete;

    TextView location1;

    TextView delete1;

    String s1,s2,s3,s4;


    db db;


    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    name=findViewById(R.id.name);
    loc=findViewById(R.id.loc);
    name2=findViewById(R.id.name2);
    name3=findViewById(R.id.name3);

    save=findViewById(R.id.save);
    retrieve=findViewById(R.id.retrieve);
    delete=findViewById(R.id.delete);

    location1=findViewById(R.id.location1);
    delete1=findViewById(R.id.delete1);
    db=new db(MainActivity.this);

}

public void sav(View view) {
    s1=name.getText().toString();
    s2=loc.getText().toString();
    db.sav(s1,s2);
}

public void retrieve(View view) {
    s3=name2.getText().toString();
    String loc=db.retrieve(s3);
    location1.setText(loc);
    location1.setTextSize(30);
```

```
    }

    public void del(View view) {
        s4 = name3.getText().toString();
        int rowsAffected = db.delete(s4);
        String deletionMessage = "Deleted: " + rowsAffected + " records";
        delete1.setText(deletionMessage);
        delete1.setTextSize(20);
    }
}
```

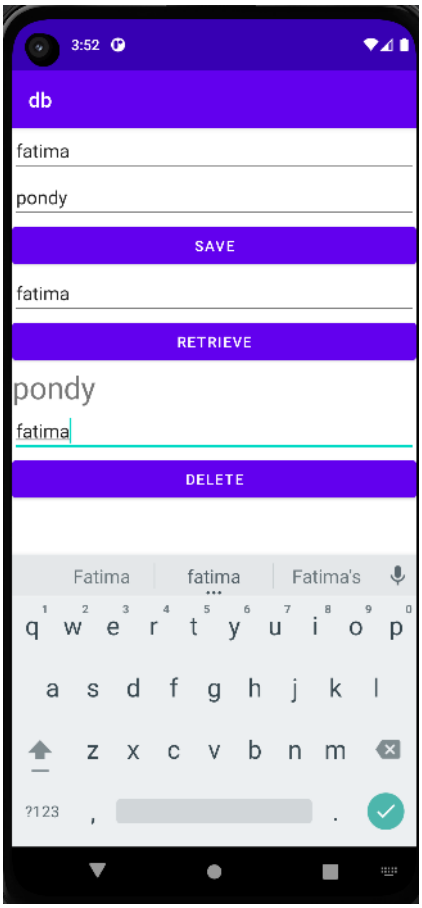
DB.JAVA

```
package com.example.db;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import androidx.annotation.Nullable;

public class db extends SQLiteOpenHelper {
    SQLiteDatabase sqLiteDatabase;
    public db( Context context) {
        super(context, "data.db", null, 1);
        sqLiteDatabase=getWritableDatabase();
    }

    public db(@Nullable Context context, @Nullable String name, @Nullable
SQLiteDatabase.CursorFactory factory, int version) {
```

```
    super(context, name, factory, version);
}

@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    sqLiteDatabase.execSQL("create table students(Name text,location text)");

}

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {



}

public void sav(String s1, String s2) {
    ContentValues cv=new ContentValues();
    cv.put("Name",s1);
    cv.put("location",s2);
    sqLiteDatabase.insert("students",null,cv);
}

public String retrieve(String s3) {
    Cursor c=sqLiteDatabase.query("students",null,"Name=?",new String[]{s3},null,null,null);
    c.moveToFirst();
    if(c.getCount()<1)
    {
        return "doesnot exist";
    }
    String l=c.getString(c.getColumnIndex("location"));
```

```
        return l;
    }

    public int delete(String s4) {
        return sqLiteDatabase.delete("students", "Name=?", new String[]{s4});
    }

}
```

OUTPUT



RESULT:

Hence successfully developed and executed a simple Database application to store and retrieve a given user's location using Android Studio.

Ex.no : 4
Date : 20-09-2023

LIBRARY MANAGEMENT SYSTEM

Aim:

To implement a simple Library Management System using Android Studio.

Procedure:

Program:

Algorithm: Book Management App

1. Initialize UI Elements:
   - Initialize EditText fields for Book Name and Author input.
   - Initialize buttons for Save, View, Update, and Delete actions.
   - Initialize TextView to display information.
2. Initialize Database:
   - Create a SQLite database named "data.db" with a table named "bookinsert".
   - The table contains columns: "BookName" (text) and "author" (text).
3. MainActivity:
   - onCreate():
     - Connect UI elements to their respective views.
     - Create a database instance using the DB class.
   - Method sav(View view):
     - Get Book Name and Author input from EditText fields.
     - Save Book Name and Author into the "bookinsert" table of the database.
   - Method retrieve(View view):
     - Get Book Name input from EditText field.
     - Retrieve Author corresponding to the provided Book Name from the database.
     - Display the retrieved Author in the TextView.
   - Method update(View view):
     - Get Book Name and updated Author input from EditText fields.
     - Update the Author for the specified Book Name in the database.
   - Method delete(View view):
     - Get Book Name input from EditText field.
     - Delete the record associated with the provided Book Name from the database.
4. DB Class (SQLiteOpenHelper):
   - onCreate():
     - Execute SQL command to create the "bookinsert" table in the database.
   - onUpgrade():
     - Handle any upgrades by calling onCreate().
   - Method sav(String s, String s1):
     - Insert Book Name and Author into the "bookinsert" table of the database.
5. User Interaction:
   - User inputs Book Name and Author details.
   - Clicking on corresponding buttons triggers actions:
     - SAVE: Saves Book Name and Author into the database.
     - VIEW: Retrieves and displays the Author for a specified Book Name.
     - UPDATE: Updates the Author for a given Book Name.
     - DELETE: Removes the record associated with the provided Book Name.

MainActivity:

```
package com.example.library_management;
```

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

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    EditText editText;
    EditText editText1;
    EditText editText2;
    EditText editText3;
    EditText editText4;
    EditText editText6;
    Button b;
    Button b1;
    Button b2;
    Button b4;
    TextView textView;
    String s,s1,s3,s4,s5,s6;

    DB db;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText=findViewById(R.id.editTextTextPersonName);
        editText1=findViewById(R.id.editTextTextPersonName2);
        editText2=findViewById(R.id.editTextTextPersonName3);
        editText3=findViewById(R.id.editTextTextPersonName4);
        editText4=findViewById(R.id.editTextTextPersonName5);
        editText6=findViewById(R.id.editTextTextPersonName6);
        b=findViewById(R.id.button);
        b1=findViewById(R.id.button2);
        b2=findViewById(R.id.button3);
        b4=findViewById(R.id.button4);
        textView=findViewById(R.id.textView);
        db=new DB(MainActivity.this);

    }
    public void sav(View view) {
        s=editText.getText().toString();
        s1=editText1.getText().toString();
        db.sav(s,s1);
    }

    public void retrieve(View view) {
        s3=editText2.getText().toString();
        String loc=db.retrieve(s3);
```

```
        textView.setText(loc);
    }
    public void update(View view){
        s4=editText3.getText().toString();
        s5=editText4.getText().toString();
        db.update(s4,s5);
    }
    public void delete(View view) {
        s6 = editText6.getText().toString().trim();
        db.delete(s6);
        db.close();
    }



}
```

DB

```
package com.example.library_management;

import android.annotation.SuppressLint;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DB extends SQLiteOpenHelper {
    SQLiteDatabase sqLiteDatabase;
    public DB( Context context) {
        super(context, "data.db", null, 1);
        sqLiteDatabase=getWritableDatabase();
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("create table bookinsert(BookName text,author text)");

    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {

        onCreate(sqLiteDatabase);
    }

    public void sav(String s, String s1) {
        ContentValues cv=new ContentValues();
        cv.put("BookName",s);
        cv.put("author",s1);
```

```
        sqLiteDatabase.insert("bookinsert",null,cv);
    }

    public String retrieve(String s3) {
        Cursor c=sqLiteDatabase.query("bookinsert",null,"BookName=?",new
String[]{s3},null,null,null);
        c.moveToFirst();
        if(c.getCount()<=0)
        {
            return "does not exist";
        }
        @SuppressLint("Range") String l=c.getString(c.getColumnIndex("author"));
        return l;
    }

    public void update(String s4, String s5) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("author", s5); // Update the "author" column

        int rowsAffected = db.update("bookinsert", values, "BookName=?", new String[]{s4});

        if (rowsAffected > 0) {
            // The update was successful
        } else {
            // No record with the specified BookName was found
        }

        db.close();
    }

    public void delete(String s6) {
        SQLiteDatabase db = this.getWritableDatabase();
        int rowsAffected = db.delete("bookinsert", "BookName=?", new String[]{s6});
        if (rowsAffected > 0) {
            // The update was successful
        } else {
            // No record with the specified BookName was found
        }
        db.close();
    }



}
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical"
tools:context=".MainActivity" >

<EditText
    android:id="@+id/editTextTextPersonName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Book Name" />

<EditText
    android:id="@+id/editTextTextPersonName2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Author" />

<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="sav"
    android:text="SAVE" />

<EditText
    android:id="@+id/editTextTextPersonName3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint=" Book Name" />

<Button
    android:id="@+id/button2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="View"
    android:onClick="retrieve"/>

<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="" />

<EditText
    android:id="@+id/editTextTextPersonName4"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Book Name" />

    <EditText
        android:id="@+id/editTextTextPersonName5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Author" />

    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="UPDATE"
        android:onClick="update"/>

    <EditText
        android:id="@+id/editTextTextPersonName6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Book Name" />

    <Button
        android:id="@+id/button4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="delete"
        android:text="DELETE" />


</LinearLayout>
```

Result

Hence successfully developed and executed an Android application for a simple library management system having Databases using Android Studio.

NOTIFICATION IN ANDROID

PROGRAM

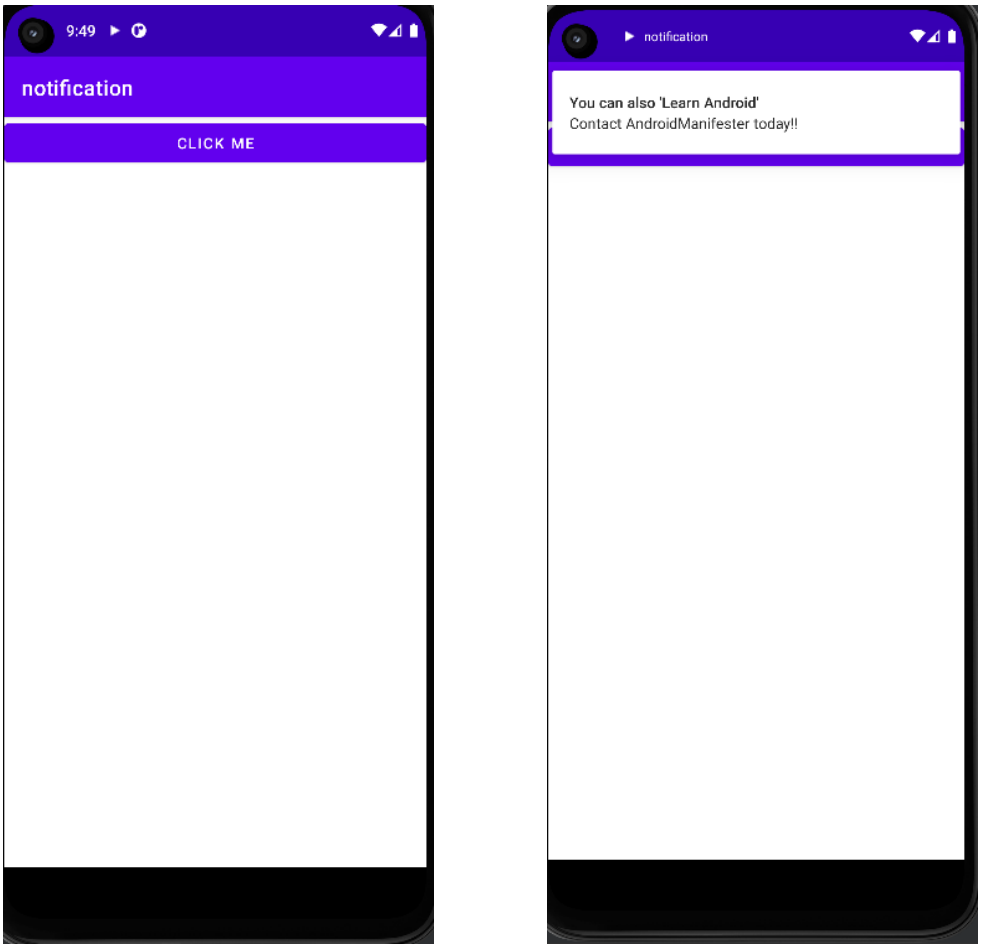
main_activity.java

```
package com.example.notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.NotificationCompat;

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

   public void showNotificationOreoAndAbove(View view) {
      NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

      int notificationId = 1;
      String channelId = "channel-01";
      String channelName = "Channel Name";
      int importance = NotificationManager.IMPORTANCE_HIGH;

      if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
         NotificationChannel mChannel = new NotificationChannel(
               channelId, channelName, importance);
         notificationManager.createNotificationChannel(mChannel);
      }

      NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(MainActivity.this,
channelId)
            .setSmallIcon(android.R.drawable.ic_media_play)
            .setContentTitle("You can also 'Learn Android'")
            .setContentText("Contact AndroidManifester today!!");

      Intent intent = new Intent(MainActivity.this, MainActivity.class);
      PendingIntent pendingIntent = PendingIntent.getActivity(MainActivity.this, 0, intent, 0);
      mBuilder.setContentIntent(pendingIntent);
      notificationManager.notify(notificationId, mBuilder.build());
   }
}
```

main_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:onClick="showNotificationOreoAndAbove"/>

</LinearLayout>
```

OUTPUT



E-MAIL USING REACT

```
package com.example.email;

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 com.example.email.R;

public class MainActivity extends AppCompatActivity {
  EditText editText;
  EditText edittext2;
  EditText editText3;
  String s,s1,s2;
  Button b;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     editText=findViewById(R.id.editTextTextPersonName);
     edittext2=findViewById(R.id.editTextTextPersonName2);
     editText3=findViewById(R.id.editTextTextPersonName3);
     b=findViewById(R.id.button);
  }

  public void send(View view) {
     s=editText.getText().toString();
     s1=edittext2.getText().toString();
     s2=editText3.getText().toString();

     Intent emailIntent=new Intent(Intent.ACTION_SEND);
     String[] recipients= new String[]{s};
     emailIntent.putExtra(Intent.EXTRA_EMAIL,recipients);
     emailIntent.putExtra(Intent.EXTRA_SUBJECT,s1);
     emailIntent.putExtra(Intent.EXTRA_TEXT,s2);
     emailIntent.setType("text/plain");
     startActivity(Intent.createChooser(emailIntent,"send mail"));



  }
}
```

INSTALLATION OF REACT JS

Prerequisites:
1. Node.js and npm: Ensure you have Node.js installed. You can download it from [Node.js website](https://nodejs.org/). npm (Node Package Manager) comes bundled with Node.js.

Steps to Install React:

1. Create a New React App:
Open your terminal and run the following command to create a new React app using Create React App (a tool built by Facebook for easily creating React applications):

```bash
npx create-react-app my-react-app
```

Replace `my-react-app` with the desired name of your app.

2. Navigate to Your React App Directory:
```bash
cd my-react-app
```

3. Start the Development Server
Once inside your React app directory, run the following command to start the development server:

```bash
npm start
```

This command will start a development server and automatically open your default web browser to display your React app. It also sets up a live reloading mechanism to update the app in real-time as you make changes.

4. Explore and Develop:
You can now explore the React app structure created for you. The main entry point is usually the `src` directory, particularly the `App.js` file.

Additional Notes:

- Package Manager: If you prefer using `yarn` as a package manager, you can replace `npm` with `yarn` in the above commands.

- Text Editor/IDE: Use your preferred text editor or integrated development environment (IDE) to modify and work on your React app files.

- Dependencies: React apps created using Create React App come with all necessary dependencies preconfigured, allowing you to start coding right away without worrying about setup.

Ex.no : 5
Date : 27-09-2023

BMI Calculator

Aim:

To implement and build a simple BMI Calculator using React JS.

Procedure:

Algorithm: Building BMI Calculator App in React

1. Create a React App:
   - Use 'npx create-react-app bmi-calculator-app' to generate a new React application named 'bmi-calculator-app'.
   - Navigate to the created directory: 'cd bmi-calculator-app'.
2. Component Creation:
   - Create a functional component 'BmiCalculator' in 'src/App.js'.
   - Define states using the 'useState' hook for heightValue, weightValue, bmiValue, and bmiMessage.
3. Styling:
   - Create a CSS file 'App.css' in the 'src' directory.
   - Add styles for the components and layout as provided in the example.
4. BmiCalculator Component Structure:
   - Render a form containing:
     - Input fields for height in centimeters and weight in kilograms.
     - A button to trigger BMI calculation.
     - Display area for BMI value and result message.
5. Event Handling:
   - Implement onChange handlers for height and weight inputs to update state values.
   - Implement calculateBmi function:
     - Calculate BMI using the provided formula (weight / (heightInMeters * heightInMeters)).
     - Set BMI value in the state.
     - Determine BMI category based on calculated BMI and set the appropriate message.
6. UI Rendering:
   - Render the BMI Calculator UI using JSX:
     - Display header with 'BMI Calculator'.
     - Input fields for height and weight.
     - Calculate button to trigger BMI calculation.
     - Display area for BMI value and category message.
7. Start Development Server:
   - Run 'npm start' to start the development server.
   - Access the app at 'http://localhost:3000/' in the default browser.
8. Testing and Refinement:
   - Test the app functionality by entering various height and weight values.
   - Verify that BMI calculation and category messages are displayed correctly.
   - Refine styles and layout as needed.
9. Enhancements:
   - Add error handling for invalid inputs (non-numeric values, negative values, etc.).
   - Customize styles or layout to match design preferences.
   - Extend functionality by adding more BMI categories or additional features.
10. Deployment:
   - Deploy the React app to a hosting platform like Vercel, Netlify, GitHub Pages, etc., following their deployment guidelines.

Program:
//App.js

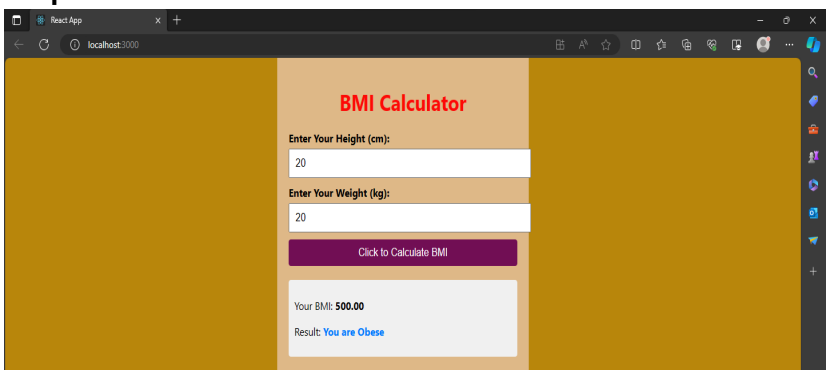
```
import './App.css';
import React, { useState } from 'react';

function App() {
  const [height, setHeight] = useState('');
  const [weight, setWeight] = useState('');
  const [unitSystem, setUnitSystem] = useState('metric');
  const [bmi, setBmi] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    const heightInMeters = unitSystem === 'metric' ? height / 100 : height / 39.37;
    const weightInKg = unitSystem === 'metric' ? weight : weight / 2.205;
    const bmi = weightInKg / (heightInMeters * heightInMeters);
    setBmi(bmi.toFixed(2));
  };

  const handleHeightChange = (e) => {
    setHeight(e.target.value);
  };

  const handleWeightChange = (e) => {
    setWeight(e.target.value);
  };

  const handleUnitSystemChange = (e) => {
    setUnitSystem(e.target.value);
  };

  return (
    <div className="App">
      <h1>BMI Calculator</h1>
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="height">Height ({unitSystem === 'metric' ? 'cm' : 'inches'})</label>
          <input type="number" id="height" value={height} onChange={handleHeightChange}
required />
        </div>
        <div>
          <label htmlFor="weight">Weight ({unitSystem === 'metric' ? 'kg' : 'lbs'})</label>
          <input type="number" id="weight" value={weight} onChange={handleWeightChange}
required />
        </div>
        <div>
          <label htmlFor="unitSystem">Unit System</label>
          <select id="unitSystem" value={unitSystem} onChange={handleUnitSystemChange}>
            <option value="metric">Metric</option>
```

```
          <option value="imperial">Imperial</option>
        </select>
      </div>
      <button type="submit">Calculate BMI</button>
    </form>
    {bmi && <p>Your BMI is {bmi} ({unitSystem === 'metric' ? 'kg/m²' : 'lbs/in²'})</p>}
  </div>
 );
}

export default App;
```

```
/* App.css */
.back{
 background-color: darkgoldenrod;
}

.container {
        max-width: 400px;
        margin: 0 auto;
        padding: 20px;
 background-color: burlywood;
}

h1 {
        color: hsla(0, 100%, 50%, 0.953);
        text-align: center;
        margin-bottom: 20px;
}

.input-container {
        margin-bottom: 10px;
}

label {
        display: block;
        font-weight: bold;
        margin-bottom: 5px;
}
input[type='number'] {
        width: 100%;
        padding: 10px;
        font-size: 16px;
}
.calculate-btn {
        display: block;
        width: 100%;
        padding: 10px;
        background-color: hsl(318, 78%, 25%);
```

```
        color: #fff;
        font-size: 16px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
}
.result {
        margin-top: 20px;
        padding: 10px;
        background-color: #f0f0f0;
        border-radius: 4px;
}
.bmi-value {
        font-weight: bold;
}
.bmi-message {
        color: #007bff;
        font-weight: bold;
}
```

Output:



Result

Hence a BMI Calculator is built using React Js.

Ex.no : 6
Date : 11-10-2023

TO-DO LIST

Aim:

To implement and build a simple to-do list application using React JS.

Procedure:

1. Initialize a React App:
   - Set up a React project using `create-react-app` or any other preferred React setup.
   - Create necessary files like `App.js`, `index.js`, `App.css`, and set up the project structure.

2. App Component (`App.js`):
   - Import React and the necessary hooks (`useState`).
   - Define the `App` functional component.
   - Set up state variables using the `useState` hook for managing `newItem`, `items` array, `showEdit` (for tracking editing), and `updatedText` (for updated input).
   - Implement functions `addItem`, `deleteItem`, and `editItem` to add, delete, and edit items respectively in the list.
   - Return a JSX structure representing the todo list application.
   - Utilize inputs, buttons, and list elements to manage the todo items and editing functionality.
   - Export the `App` component as the default export.

3. Rendering the App (`index.js`):
   - Import React and ReactDOM.
   - Import the `App` component.
   - Render the `App` component within the `ReactDOM.render` method.
   - Ensure it's being rendered in the root HTML element (`<div id="root"></div>`).

4. Styling (`App.css`):
   - Define CSS styles for the todo list application elements (input, buttons, list, etc.).
   - Style the application according to the provided CSS snippets (input, list, delete button, etc.).

5. Run the Application:
   - Start the React development server using `npm start` or `yarn start`.
   - View the application in the browser to verify the functionality and appearance of the todo list.

6. Additional Considerations:
   - Test the application thoroughly, especially adding, deleting, and editing items.
   - Refactor and optimize the code where necessary for better readability and performance.
   - Add comments or documentation to clarify complex logic or functionalities.

Program:
App.js:

```
import React, { useState } from "react";
import "./App.css";

function App() {
  // State Hook - `useState`
  const [newItem, setNewItem] = useState("");
  const [items, setItems] = useState([]);
```

```
const [showEdit, setShowEdit] = useState(-1);
const [updatedText, setUpdatedText] = useState("");

// Helper Functions

/* Adds a new item to the list array*/
function addItem() {
  // ! Check for empty item
  if (!newItem) {
    alert("Press enter an item.");
    return;
  }

  const item = {
    id: Math.floor(Math.random() * 1000),
    value: newItem,
  };

  // Add new item to items array
  setItems((oldList) => [...oldList, item]);

  // Reset newItem back to original state
  setNewItem("");
}

/* Deletes an item based on the `item.id` key */
function deleteItem(id) {
  const newArray = items.filter((item) => item.id !== id);
  setItems(newArray);
}

/* Edit an item text after creating it. */
function editItem(id, newText) {
  // Get the current item
  const currentItem = items.filter((item) => item.id === id);

  // Create a new item with same id
  const newItem = {
    id: currentItem.id,
    value: newText,
  };

  deleteItem(id);

  // Replace item in the item list
  setItems((oldList) => [...oldList, newItem]);
  setUpdatedText("");
  setShowEdit(-1);
}
```

```
  // Main part of app
  return (
    <div className="app">
      {/* 1. Header  */}
      <h1>My Todo List</h1>

      {/* 2. Add new item (input) */}
      <input
        type="text"
        placeholder="Add an item..."
        value={newItem}
        onChange={(e) => setNewItem(e.target.value)}
      />

      {/* Add (button) */}
      <button onClick={() => addItem()}>Add</button>

      {/* 3. List of todos (unordered list) */}
      <ul>
        {items.map((item) => {
          return (
            <div>
              <li key={item.id} onClick={() => setShowEdit(item.id)}>
                {item.value}
                <button
                  className="delete-button"
                  onClick={() => deleteItem(item.id)}
                >

                </button>
              </li>

              {showEdit == item.id ? (
                <div>
                  <input
                    type="text"
                    value={updatedText}
                    onChange={(e) => setUpdatedText(e.target.value)}
                  />
                  <button onClick={() => editItem(item.id, updatedText)}>
                    Update
                  </button>
                </div>
              ) : null}
            </div>
          );
        })}
      </ul>
    </div>
  );
}
```

```
export default App;
```

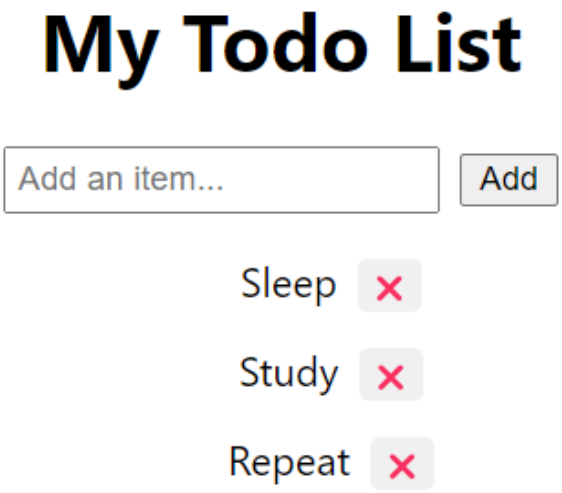
index.js:

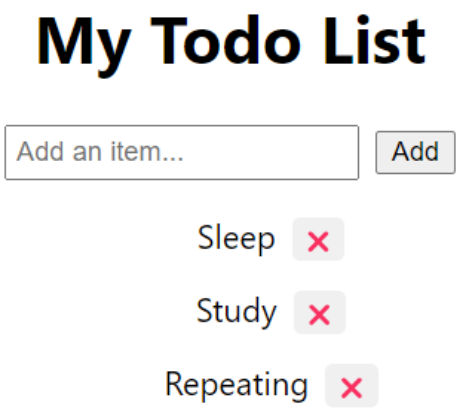
```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

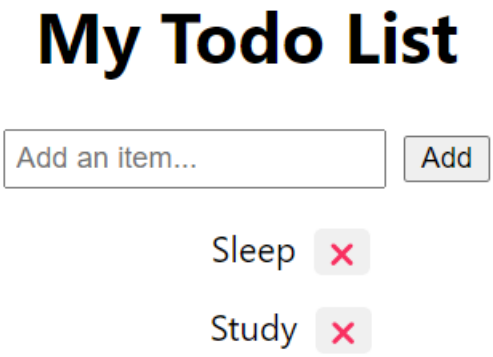
App.css:

```
.app {
  text-align: center;
}

input {
  padding: 4px;
  margin-right: 8px;
}


ul {
  /* Remove bullet point  */
  list-style: none;
}

li {
  margin: 12px 0;
}

.delete-button {
  font-size: 10px;
  margin-left: 8px;

  border: none;
  color: white;
  border-radius: 4px;

  padding-top: 4px;
  padding-bottom: 4px;
}
```

Output:







Result:

Hence a To-Do List is built using React JS and the application is tested.

INSTALLATION OF APACHE CORDOVA

Prerequisites:

- Node.js: Cordova requires Node.js, so make sure you have it installed.
- Java Development Kit (JDK): For Android development with Cordova, you'll need the JDK.
- Android Studio (Optional): If you're targeting Android, having Android Studio installed can be helpful for managing SDKs and emulators.
- Xcode (Mac): For iOS development with Cordova, you'll need Xcode.

Installation Steps:

1. Install Node.js:
   - Download and install Node.js from the [official website](https://nodejs.org/).
   - Verify the installation by running `node -v` and `npm -v` in your terminal/command prompt.
2. Install Cordova:
   - Once Node.js is installed, install Cordova globally via npm:
     ```
     npm install -g cordova
     ```
   - Verify the installation by running `cordova -v`.
3. Install JDK (For Android):
   - Download and install the JDK from the [Oracle website](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html).
   - Set the JAVA_HOME environment variable to point to your JDK installation.
4. Set Up Android Studio (For Android):
   - Download and install Android Studio from the [official website](https://developer.android.com/studio).
   - Open Android Studio and install required SDKs and tools via the SDK Manager.
5. Set Up Xcode (For iOS - Mac Only):
   - Install Xcode from the App Store if you haven't already.
   - Accept the Xcode license agreement by opening Xcode once before using Cordova.
6. Create a Cordova Project:
   - Open your terminal/command prompt.
   - Create a new Cordova project by running:
     ```
     ```

```
cordova create myApp com.example.myApp MyAppName
```

Replace `myApp`, `com.example.myApp`, and `MyAppName` with your preferred app name, package name, and display name.

- Navigate into the project directory:

```
cd myApp
```

7. Add Platforms:

- Add the platforms you want to target (Android, iOS, etc.):

```
cordova platform add android
cordova platform add ios
```

- Make sure you meet the platform-specific requirements (e.g., for Android, the SDKs should be installed).

8. Build and Run:

- To build the app for a specific platform:

```
cordova build android
cordova build ios
```

- To run the app on a connected device or emulator:

```
cordova run android
cordova run ios
```

Ex.no : 7

Date : 18-10-2023

EXPENSE MANAGER

Aim:

To implement and build a simple Expense Manager using Apache Cordova.

Procedure:

1.HTML (index.html):

Create index.html and construct the necessary structure with form inputs and a submit button. Link the CSS file (index.css) to style the app.

2.CSS (index.css):

Write CSS styles for various components, ensuring they match the provided styles for the desired layout and appearance.

3.JavaScript (index.js):

Develop JavaScript logic to handle the deviceready event, retrieve form input values, perform calculations, and display

the data using document.write().

4.Testing and Refinement:

Test the application in a browser, verifying functionality and appearance.

Refine the code as needed to enhance the user interface or optimize functionality.

5.Deployment:

Host the HTML, CSS, and JavaScript files on a server or deploy them to a web hosting service for accessibility.

Program:

HTML file

index.html:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="initial-scale=1, width=device-width, viewport-fit=cover">
      <meta name="color-scheme" content="light dark">
      <link rel="stylesheet" href="css/index.css">
      <title>Hello World</title>
   </head>
```

```
  <body>
    <div class="app">
      <h1 style="font-weight: bolder;">Expense Manager</h1>
      <br><br><br>
      <form name="form1" onsubmit="return onDisplay()">
        <span>Enter Your Income
(Monthly):            &
nbsp;            <input
type="number" id="income" name="income"></span>
        <br>
        <br>
        <br>
        <span>Enter your Expenses spent for the month:</span>
        <br> <br>

Utilities:            &n
bsp;  <input type="number" id="util" name="util">
        <br><br>
        Transportation:<input type="number" id="trans" name="trans">
        <br><br>

Food:            &nbs
p;       <input type="number" id="food"
name="food">
        <br><br>

Housing:            &n
bsp; <input type="number" id="house" name="house">
        <br><br>
        Entertainment:   <input type="number" id="enter" name="enter">
        <br><br>
        <input type="submit" id="submit" value="Submit" name="submit" style="width: 300px;
height: 60px; background-color: bisque; color: black; font-size: 20px; border-radius: 20px;
font-weight: bold;">
      </form>
    </div>
  </body>
</html>
```

Index.css:

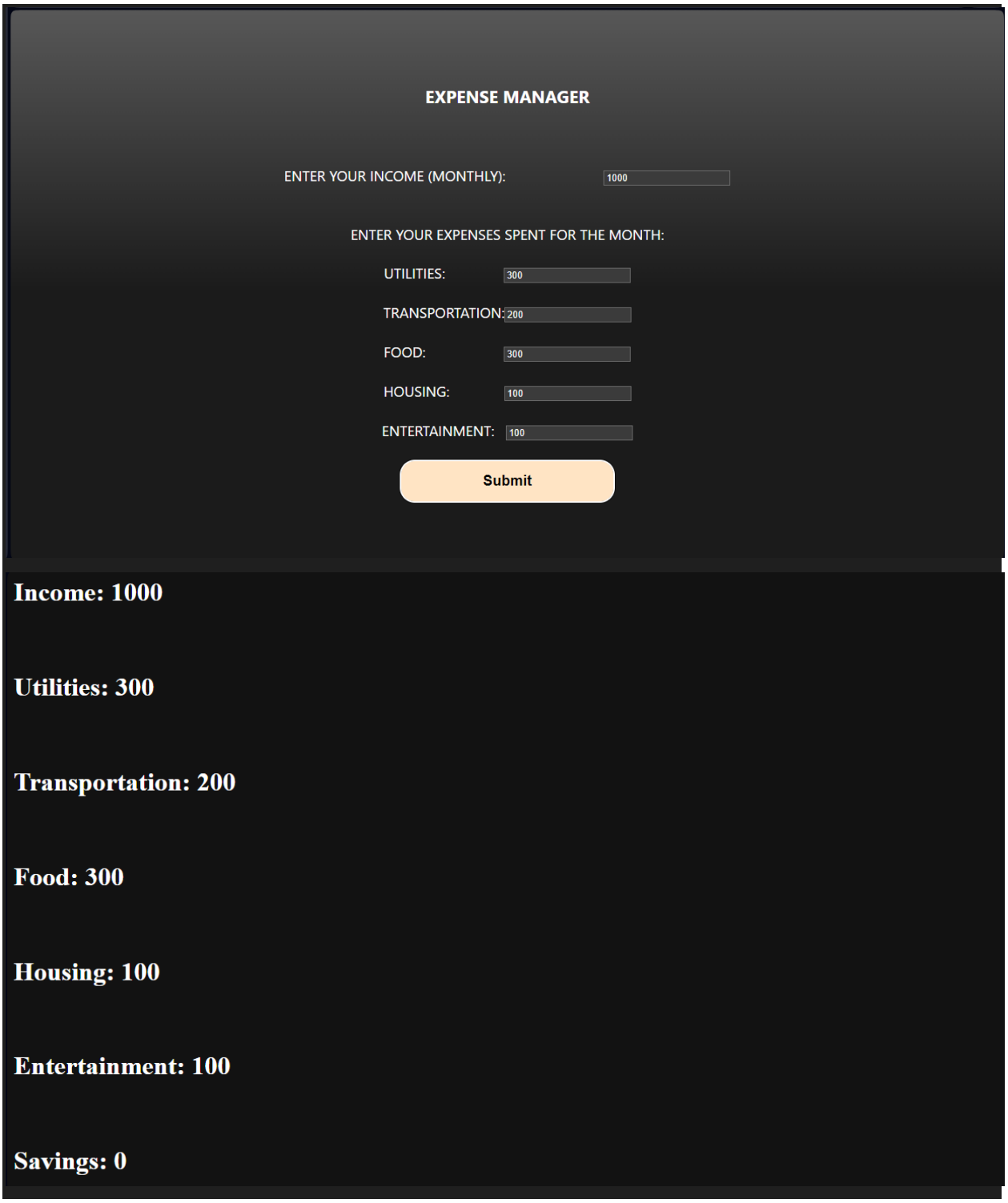
```
* {
    -webkit-tap-highlight-color: rgba(0,0,0,0); /* make transparent link selection, adjust last value
opacity 0 to 1.0 */
}


body {
    -webkit-touch-callout: none;                /* prevent callout to copy image, etc when tap to hold */
    -webkit-text-size-adjust: none;             /* prevent webkit from resizing text to fit */
    -webkit-user-select: none;                  /* prevent copy paste, to allow, change 'none' to 'text' */
    background-color:#E4E4E4;
    background-image:linear-gradient(to bottom, #A7A7A7 0%, #E4E4E4 51%);
    font-family: system-ui, -apple-system, -apple-system-font, 'Segoe UI', 'Roboto', sans-serif;
    font-size:12px;
    height:100vh;
    margin:0px;
    padding:0px;
    /* Padding to avoid the "unsafe" areas behind notches in the screen */
    padding: env(safe-area-inset-top, 0px) env(safe-area-inset-right, 0px)
env(safe-area-inset-bottom, 0px) env(safe-area-inset-left, 0px);
    text-transform:uppercase;
    width:100%;
}


/* Portrait layout (default) */
.app {
    /*background:url(../img/logo.png) no-repeat center top; /* 170px x 200px */
    position:relative;              /* position in the center of the screen */
    top:10%;
    height:50px;                    /* text area height */
    width:300px;                    /* text area width */
    text-align:center;
    padding:180px 0px 0px 0px;     /* image height is 200px (bottom 20px are overlapped with
text) */
    margin:-115px 0px 0px -112px;
    display: inline; /* offset vertical: half of image height and text area height */
    font-size: 20px;                            /* offset horizontal: half of text area width */
}


h1 {
    font-size:24px;
```

```
    font-weight:normal;
    margin:0px;
    overflow:visible;
    padding:0px;
    text-align:center;
}


.event {
    border-radius:4px;
    color:#FFFFFF;
    font-size:12px;
    margin:0px 30px;
    padding:2px 0px;
}


.event.listening {
    background-color:#333333;
    display:block;
}


.event.received {
    background-color:#4B946A;
    display:none;
}


#deviceready.ready .event.listening { display: none; }
#deviceready.ready .event.received { display: block; }


.font
{
    font-size: 20px;
    display: block;
    font-weight: 700;
}
```

index.js:

```
document.addEventListener('deviceready', onDeviceReady, false);
var income = parseInt(document.form1.income.value, 10);
        var util = parseInt(document.form1.util.value, 10);
        var trans = parseInt(document.form1.trans.value, 10);
        var food = parseInt(document.form1.food.value, 10);
        var house = parseInt(document.form1.house.value, 10);
        var enter = parseInt(document.form1.enter.value, 10);


        document.write("<div class='font'><h1>Income: " + income + "<h1><br></div>");

        document.write("<div class='font'><h1>Utilities: " + util  + "<h1><br></div>");

        document.write("<div class='font'><h1>Transportation: " + trans + "<h1><br></div>");

        document.write("<div class='font'><h1>Food: " + food + "<h1><br></div>");

        document.write("<div class='font'><h1>Housing: " + house + "<h1><br></div>");

        document.write("<div class='font'><h1>Entertainment: " + enter + "<h1><br></div>");

        var savings=parseInt(income-(util+trans+food+house+enter));
        document.write("<div class='font'><h1>Savings: " + savings + "<h1><br></div>");
        return true;
}
```

Output:



Result:

Thus the program to build a simple expense manager is done and verified.

Ex.no : 8

Date : 25-10-2023

UNIT CONVERSION

Aim:

To implement and build a simple Unit Conversion System using React JS.

Procedure:

1.Setup React Application:

Use 'npx create-react-app my-app' to create a new React app.

Navigate to the created directory: cd my-app.

2.Component Creation:

Create App.js, Currencyflow.js, KilogramsToPoundsConverter.js, and App.css files in the src directory.

Define components with state, input change handlers, unit change handlers, and rendering logic as described in the provided code.

3.Component Interactions:

Import necessary components in App.js.

Include UnitConverter and KilogramsToPoundsConverter components in the App component's return statement.

4.Styling:

Write CSS styles in App.css to define layout, input fields, dropdowns, and other visual aspects based on the provided styles.

5.Testing:

Run the app with npm start and test the functionality of both converters by entering values and selecting units.

Verify that the conversion between kilometers/miles and kilograms/pounds is accurate.

6.Refinement and Enhancement:

Refine the styles and layout as needed to improve the app's appearance.

Add error handling for invalid inputs or edge cases.

7.Deployment:

Deploy the React app to a hosting platform or serve it using a web server for public access.

Program:

App.js:

```
import React from 'react';
```

```
import './App.css';
import UnitConverter from './Currencyflow';
import KilogramsToPoundsConverter from './KilogramsToPoundsConverter';
function App() {
  return (
    <div className="App">
      <UnitConverter />
      <KilogramsToPoundsConverter />
    </div>

  );
}
export default App;
```

Currencyflow.js:

```
import React, { Component } from 'react';
class Currencyflow extends Component {
  constructor() {
    super();
    this.state = {
      inputValue: 0,
      outputValue: 0,
      selectedUnit: 'km', // Default to kilometers
    };
  }
  handleInputChange = (event) => {
    const inputValue = parseFloat(event.target.value);
    const selectedUnit = this.state.selectedUnit;
    if (selectedUnit === 'km') {
      const outputValue = inputValue * 0.621371; // Conversion from kilometers to miles
      this.setState({ inputValue, outputValue });
    } else if (selectedUnit === 'mi') {
      const outputValue = inputValue / 0.621371; // Conversion from miles to kilometers
      this.setState({ inputValue, outputValue });
    }
  }
  handleUnitChange = (event) => {
    const selectedUnit = event.target.value;
    this.setState({ selectedUnit });
    this.handleInputChange({ target: { value: this.state.inputValue } });
  }
  render() {
    return (
      <div>
```

```
      <h2>Unit Converter - Kilometers and Miles and vice versa</h2>
      <input
        type="number"
        placeholder="Enter value"
        value={this.state.inputValue}
        onChange={this.handleInputChange}
      />
      <select onChange={this.handleUnitChange} value={this.state.selectedUnit}>
        <option value="km">Kilometers (km)</option>
        <option value="mi">Miles (mi)</option>
      </select>
      <p>{this.state.selectedUnit === 'km' ? 'Miles' : 'Kilometers'}: {this.state.outputValue}</p>
    </div>
  );
 }
}
export default Currencyflow;
```

KilogramsToPoundsConverter.js:

```
import React, { Component } from 'react';
class KilogramsToPoundsConverter extends Component {
 constructor() {
  super();
  this.state = {
   inputValue: 0,
   outputValue: 0,
   selectedUnit: 'kg', // Default to kilograms
  };
 }
 handleInputChange = (event) => {
  const inputValue = parseFloat(event.target.value);
  const selectedUnit = this.state.selectedUnit;
  if (selectedUnit === 'kg') {
   const outputValue = inputValue * 2.20462; // Conversion from kilograms to pounds
   this.setState({ inputValue, outputValue });
  } else if (selectedUnit === 'lb') {
   const outputValue = inputValue / 2.20462; // Conversion from pounds to kilograms
   this.setState({ inputValue, outputValue });
  }
 }
 handleUnitChange = (event) => {
  const selectedUnit = event.target.value;
  this.setState({ selectedUnit });
  this.handleInputChange({ target: { value: this.state.inputValue } });
```

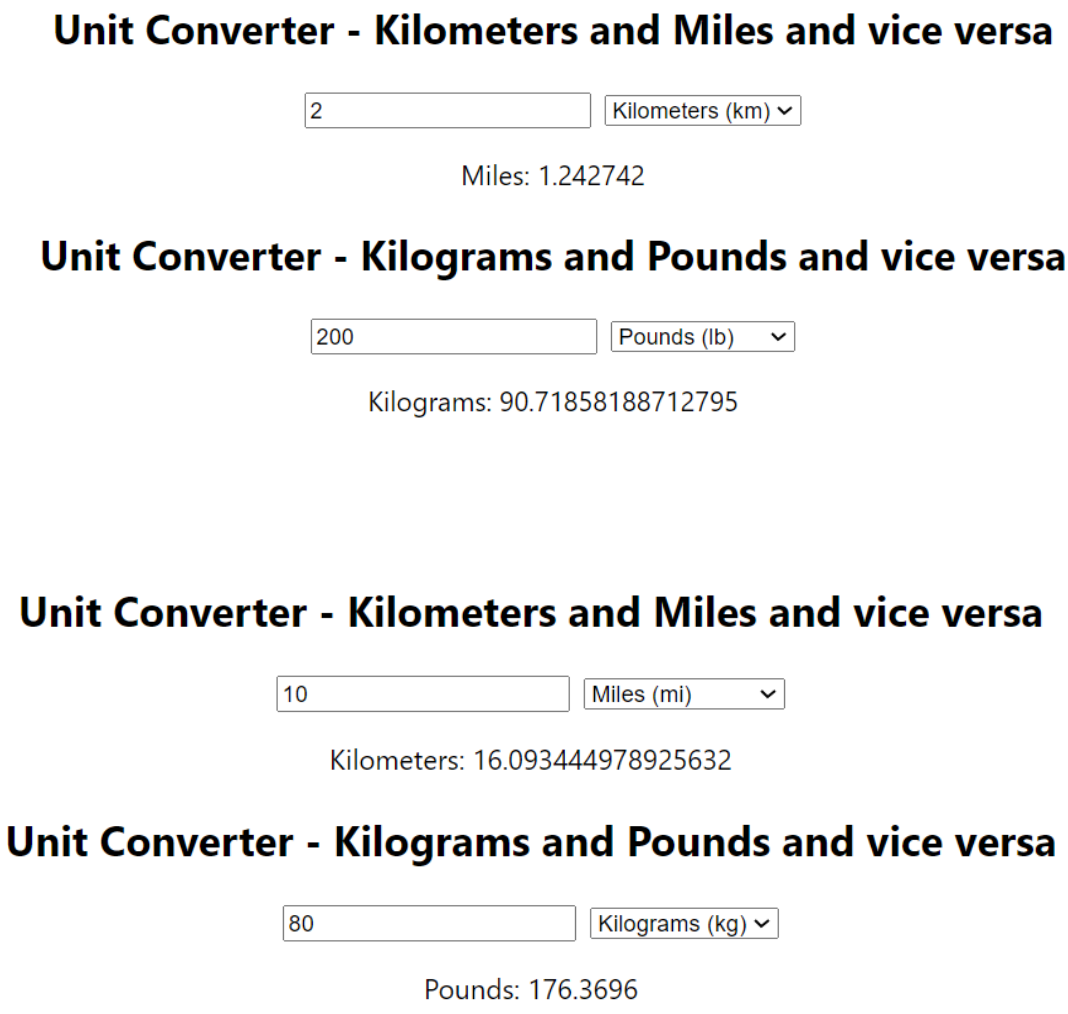
```
  }
  render() {
    return (
      <div>
        <h2>Unit Converter - Kilograms and Pounds and vice versa</h2>
        <input
          type="number"
          placeholder="Enter value"
          value={this.state.inputValue}
          onChange={this.handleInputChange}
        />
        <select onChange={this.handleUnitChange} value={this.state.selectedUnit}>
          <option value="kg">Kilograms (kg)</option>
          <option value="lb">Pounds (lb)</option>
        </select>
        <p>{this.state.selectedUnit === 'kg' ? 'Pounds' : 'Kilograms'}: {this.state.outputValue}</p>
      </div>
    );
  }
}
export default KilogramsToPoundsConverter;
```

App.css:

```
body{
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  text-align: center;
}
h1{
  margin: 0;
  margin-bottom: .5rem;
}
.input{
  border: 1px solid #333;
  border-radius: .3em;
  padding: .25rem;
  width: 7em;
}
select{
  margin-left: .5rem;
}
```

```
.equals{
  font-weight: bold;
  font-size: 2em;
}
```

Output:



Result:

Hence a Unit Converter is Built using Apache Cordova and it is Tested successfully.

Ex.no : 10
Date : 30-10-2023

CURRENT LOCATION FINDER

Aim:

To implement and build a simple application which tells the current location of the user using Apache Cordova.

Procedure:

HTML (index.html):
Create the basic structure for the webpage, including elements like buttons to trigger geolocation functions and a space to display the results.

CSS (index.css):
Style the layout, positioning, and appearance of elements. Adjust fonts, colors, sizes, and layout properties as needed for your design.

JavaScript (index.js):
1. Get Current Position:
   - Create a function `getPosition()` triggered by a button click.
   - Define options for geolocation settings (e.g., accuracy, maximum age).
   - Use `navigator.geolocation.getCurrentPosition()` to retrieve the current position.
   - Handle success and error cases and display the retrieved information.

2. Watch Position:
   - Create a function `watchPosition()` triggered by another button click.
   - Define similar options for geolocation settings.
   - Use `navigator.geolocation.watchPosition()` to continuously monitor changes in position.
   - Handle success and error cases similarly to the "Get Current Position" function.

Algorithm for Construction:
1. HTML:
   - Create an HTML file (`index.html`).
   - Define the structure: HTML, head, meta tags, CSS linking, title, body, buttons, and a division for displaying the results.

2. CSS:
   - Create a CSS file (`index.css`).
   - Style the layout, fonts, colors, and positioning according to your design preferences.

3. JavaScript:
   - Create a JavaScript file (`index.js`).
   - Write functions `getPosition()` and `watchPosition()` as described above.
   - Link this JavaScript file at the end of the HTML body to ensure the DOM is loaded before executing the

Program:
HTML file:
index.html:

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="initial-scale=1, width=device-width, viewport-fit=cover">
        <meta name="color-scheme" content="light dark">
        <link rel="stylesheet" href="css/index.css">
        <title>Hello World</title>
    </head>
    <body>
        <div class="app">
            <h1 style="color: black; margin-left:-20px;">Current Location Tracker</h1>
            <div id="deviceready" class="blink">
                <br>
                <button id = "getPosition" style="width:300px;height:50px;margin-top:20px">Current
Position</button>
                <br><br>
                <button id = "watchPosition" style="width:300px;height:50px;margin-top:20px;">Watch
Position</button>
            </div>
        </div>
        <script src="cordova.js"></script>
        <script src="js/index.js"></script>
    </body>
</html>
```

index.css:

```
* {
    -webkit-tap-highlight-color: rgba(0,0,0,0); /* make transparent link selection, adjust last value
opacity 0 to 1.0 */
}

body {
    -webkit-touch-callout: none;                /* prevent callout to copy image, etc when tap to hold */
    -webkit-text-size-adjust: none;             /* prevent webkit from resizing text to fit */
    -webkit-user-select: none;                  /* prevent copy paste, to allow, change 'none' to 'text' */
    background-color:#E4E4E4;
    background-image:linear-gradient(to bottom, #A7A7A7 0%, #E4E4E4 51%);
    font-family: system-ui, -apple-system, -apple-system-font, 'Segoe UI', 'Roboto', sans-serif;
    font-size:12px;
    height:100vh;
    margin:0px;
    padding:0px;
    /* Padding to avoid the "unsafe" areas behind notches in the screen */
    padding: env(safe-area-inset-top, 0px) env(safe-area-inset-right, 0px)
env(safe-area-inset-bottom, 0px) env(safe-area-inset-left, 0px);
    text-transform:uppercase;
    width:100%;
}

/* Portrait layout (default) */
.app {
    background:url(../img/geolocation.jpg) no-repeat center top; /* 170px x 200px */
```

```
    position:absolute;            /* position in the center of the screen */
    left:40%;
    top:40%;
    height:200px;                 /* text area height */
    width:525px;                  /* text area width */
    text-align:center;
    padding:180px 0px 0px 0px;     /* image height is 200px (bottom 20px are overlapped with
text) */
    margin:-115px 0px 0px -112px;  /* offset vertical: half of image height and text area height */
                                   /* offset horizontal: half of text area width */
}

h1 {
    font-size:24px;
    font-weight:normal;
    margin:0px;
    overflow:visible;
    padding:0px;
    text-align:center;
}

.event {
    border-radius:4px;
    color:#FFFFFF;
    font-size:12px;
    margin:0px 30px;
    padding:2px 0px;
}



.blink {
    animation:fade 3000ms infinite;
    -webkit-animation:fade 3000ms infinite;
}
```

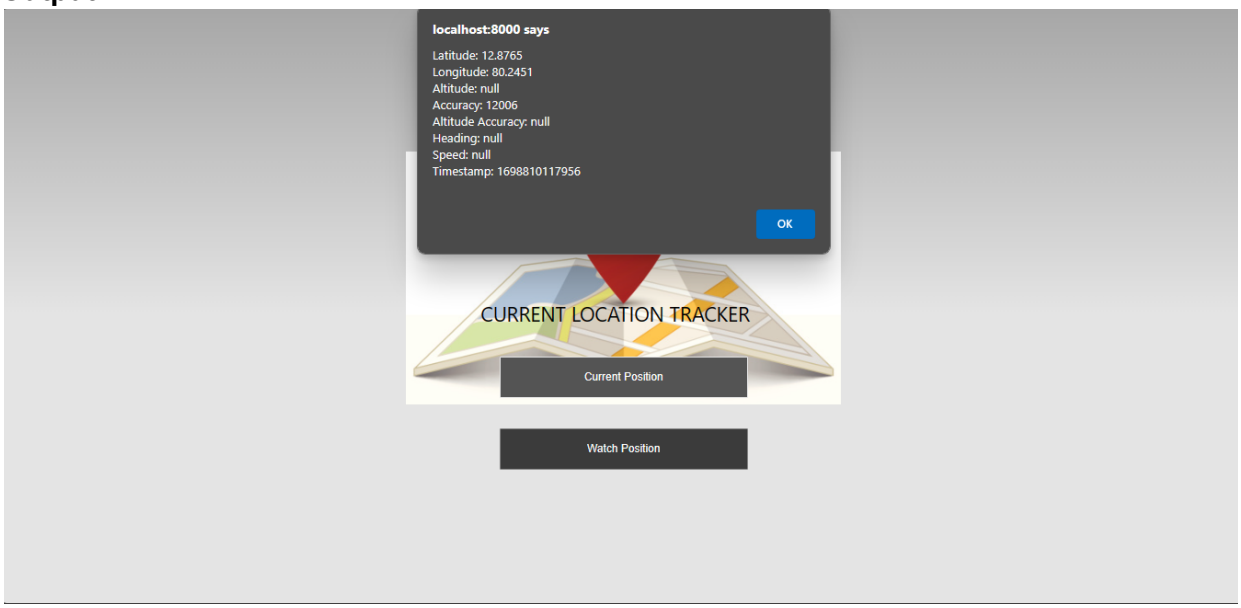
index.js:

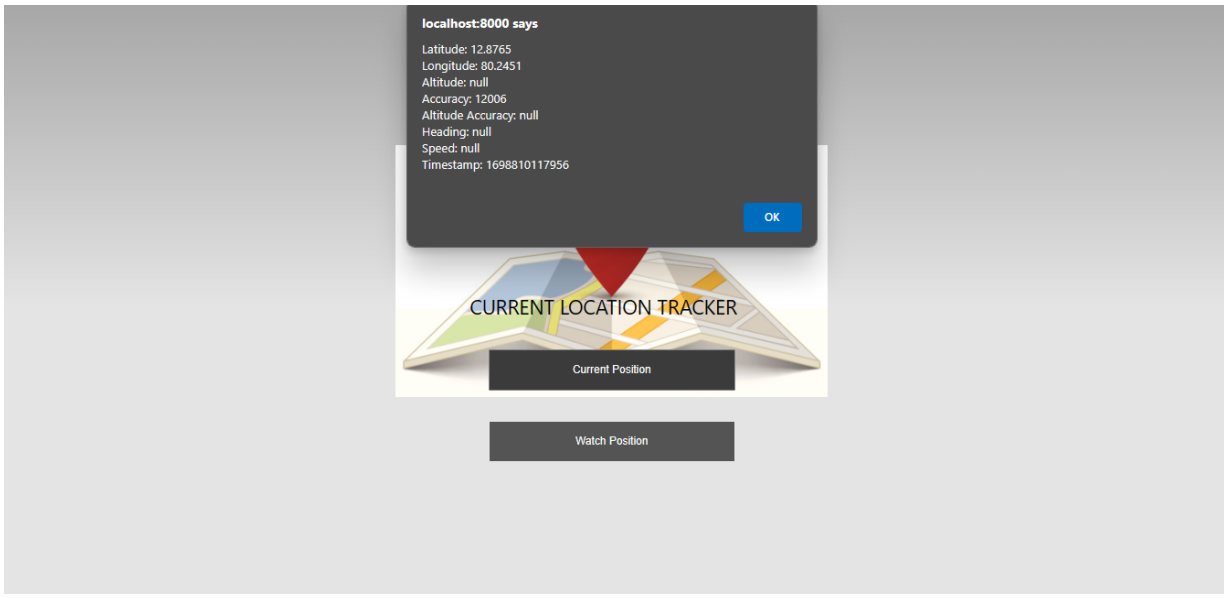
```
document.getElementById("getPosition").addEventListener("click", getPosition);
document.getElementById("watchPosition").addEventListener("click", watchPosition);

    function getPosition() {
        var options = {
          enableHighAccuracy: true,
          maximumAge: 3600000
        }
        var watchID = navigator.geolocation.getCurrentPosition(onSuccess, onError, options);

        function onSuccess(position) {
          alert('Latitude: '          + position.coords.latitude          + '\n' +
            'Longitude: '         + position.coords.longitude         + '\n' +
```

```
            'Altitude: '          + position.coords.altitude          + '\n' +
            'Accuracy: '          + position.coords.accuracy          + '\n' +
            'Altitude Accuracy: ' + position.coords.altitudeAccuracy  + '\n' +
            'Heading: '           + position.coords.heading           + '\n' +
            'Speed: '             + position.coords.speed             + '\n' +
            'Timestamp: '         + position.timestamp                + '\n');
        };

        function onError(error) {
          alert('code: '    + error.code    + '\n' + 'message: ' + error.message + '\n');
        }
     }
     function watchPosition() {
    var options = {
      maximumAge: 3600000,
      timeout: 3000,
      enableHighAccuracy: true,
    }
    var watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);

    function onSuccess(position) {
      alert('Latitude: '          + position.coords.latitude          + '\n' +
        'Longitude: '         + position.coords.longitude         + '\n' +
        'Altitude: '          + position.coords.altitude          + '\n' +
        'Accuracy: '          + position.coords.accuracy          + '\n' +
        'Altitude Accuracy: ' + position.coords.altitudeAccuracy  + '\n' +
        'Heading: '           + position.coords.heading           + '\n' +
        'Speed: '             + position.coords.speed             + '\n' +
        'Timestamp: '         + position.timestamp                + '\n');
    };

    function onError(error) {
      alert('code: '    + error.code    + '\n' +'message: ' + error.message + '\n');
    }
}
function watchPosition() {
     var options = {
        maximumAge: 3600000,
        timeout: 3000,
        enableHighAccuracy: true,
     }
     var watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);

     function onSuccess(position) {
        alert('Latitude: '          + position.coords.latitude          + '\n' +
          'Longitude: '         + position.coords.longitude         + '\n' +
          'Altitude: '          + position.coords.altitude          + '\n' +
          'Accuracy: '          + position.coords.accuracy          + '\n' +
          'Altitude Accuracy: ' + position.coords.altitudeAccuracy  + '\n' +
          'Heading: '           + position.coords.heading           + '\n' +
          'Speed: '             + position.coords.speed             + '\n' +
```

```
        'Timestamp: '          + position.timestamp                + '\n');
    };

    function onError(error) {
      alert('code: '    + error.code    + '\n' +'message: ' + error.message + '\n');
    }
}
```

Output :





Result:

Thus the application to find a user's current location is built and output is verified.

ITEM MANAGEMENT

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Item Management</title>
</head>
<body>
  <h1>Item Management</h1>

  <form id="itemForm">
    <input type="text" id="itemName" placeholder="Item name" required>
    <input type="number" id="itemPrice" placeholder="Price" required>
    <button type="submit" id="addItemBtn">Add Item</button>
  </form>

  <table id="itemTable" border="1">
    <thead>
      <tr>
        <th>Item Name</th>
        <th>Price</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody id="itemList">
      <!-- Items will be dynamically added here -->
    </tbody>
  </table>

  <script src="script.js"></script>
</body>
</html>
```

INDEX.JS

```
// Sample items to start with
let items = [
  { name: 'Item 1', price: 10 },
  { name: 'Item 2', price: 20 },
];

// Function to display items in the table
function displayItems() {
```

```
  const itemList = document.getElementById('itemList');
  itemList.innerHTML = '';

  items.forEach((item, index) => {
    const row = itemList.insertRow();
    row.innerHTML = `
      <td>${item.name}</td>
      <td>$${item.price}</td>
      <td>
        <button onclick="editItem(${index})">Edit</button>
        <button onclick="deleteItem(${index})">Delete</button>
        <button onclick="buyItem(${index})">Buy</button>
      </td>
    `;
  });
}

// Function to add a new item
function addItem(event) {
  event.preventDefault();
  const itemName = document.getElementById('itemName').value;
  const itemPrice = document.getElementById('itemPrice').value;

  if (itemName && itemPrice) {
    items.push({ name: itemName, price: parseFloat(itemPrice) });
    displayItems();
    document.getElementById('itemForm').reset();
  } else {
    alert('Please enter item name and price.');
  }
}

// Function to edit an existing item
function editItem(index) {
  const newName = prompt('Enter new name for the item:');
  const newPrice = parseFloat(prompt('Enter new price for the item:'));

  if (newName && !isNaN(newPrice)) {
    items[index].name = newName;
    items[index].price = newPrice;
    displayItems();
  } else {
    alert('Invalid input. Name and price must be provided.');
  }
```

```
}

// Function to delete an item
function deleteItem(index) {
  items.splice(index, 1);
  displayItems();
}

// Function to simulate buying an item
function buyItem(index) {
  const selectedItem = items[index];
  // Add logic for purchasing
  alert(`You purchased ${selectedItem.name} for $${selectedItem.price}`);
}

// Initial display of items
displayItems();

// Event listener for adding items
document.getElementById('itemForm').addEventListener('submit', addItem);
```

DOCTOR MANAGEMENT

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Doctor Appointment Booking</title>
</head>
<body>
  <h1>Book an Appointment</h1>

  <label for="doctorSelect">Select a Doctor:</label>
  <select id="doctorSelect">
    <option value="1">Dr. Smith</option>
    <option value="2">Dr. Johnson</option>
    <!-- Add more doctors as needed -->
  </select>

  <label for="dateInput">Select Date:</label>
  <input type="date" id="dateInput" required>

  <label for="timeInput">Select Time:</label>
  <input type="time" id="timeInput" required>

  <button onclick="bookAppointment()">Book Appointment</button>

  <script src="script.js"></script>
</body>
</html>
```

INDEX.JS

```
// Doctor schedule data (in a real application, this might be fetched from a server)
const doctorSchedules = {
  1: {
    // Doctor 1 schedule
    '2023-12-01': ['09:00', '10:00', '14:00'],
    // Add more dates and times as needed for Doctor 1
  },
  2: {
    // Doctor 2 schedule
    '2023-12-01': ['11:00', '15:00'],
    // Add more dates and times as needed for Doctor 2
```

```
  }
  // Add schedules for more doctors if required
};

// Function to check if the selected date and time are available for booking
function checkAvailability(doctorId, selectedDate, selectedTime) {
  const doctorSchedule = doctorSchedules[doctorId];

  if (doctorSchedule && doctorSchedule[selectedDate]) {
    const availableTimes = doctorSchedule[selectedDate];
    return availableTimes.includes(selectedTime);
  }

  return false;
}

// Function to book an appointment
function bookAppointment() {
  const doctorSelect = document.getElementById('doctorSelect');
  const selectedDoctorId = doctorSelect.value;

  const dateInput = document.getElementById('dateInput');
  const selectedDate = dateInput.value;

  const timeInput = document.getElementById('timeInput');
  const selectedTime = timeInput.value;

  const isAvailable = checkAvailability(selectedDoctorId, selectedDate, selectedTime);

  if (isAvailable) {
    alert(`Appointment booked with Doctor ${selectedDoctorId} on ${selectedDate} at
${selectedTime}`);
    // You might want to add logic here to handle the booking in a real application
  } else {
    alert('This time slot is not available. Please choose another time.');
  }
}
```