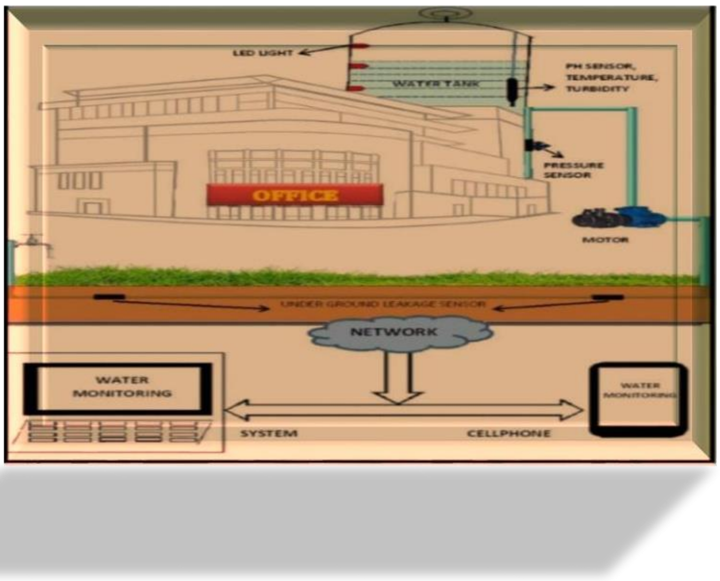
# Phase 5 : Project Documentation & Submission

| | |
|---|---|
| **PROJECT TITLE** | **IOT SMART WATER SYSTEM** |
| **NAME** | **V.Kirubagaran** |
| **TEAM ID** | **5254** |
| **TEAM NAME** | **PROJ_204182_TEAM_2** |
| **COLLEGE CODE - NAME** | **9238-MANGAYARKARASI COLLEGE OF ENGINEERING PARAVAI, MADURAI** |
| **GITHUB REPOSITORY LINK** | https://github.com/kirubagaran-v/IBM-NAANMUDHALVAN.git |

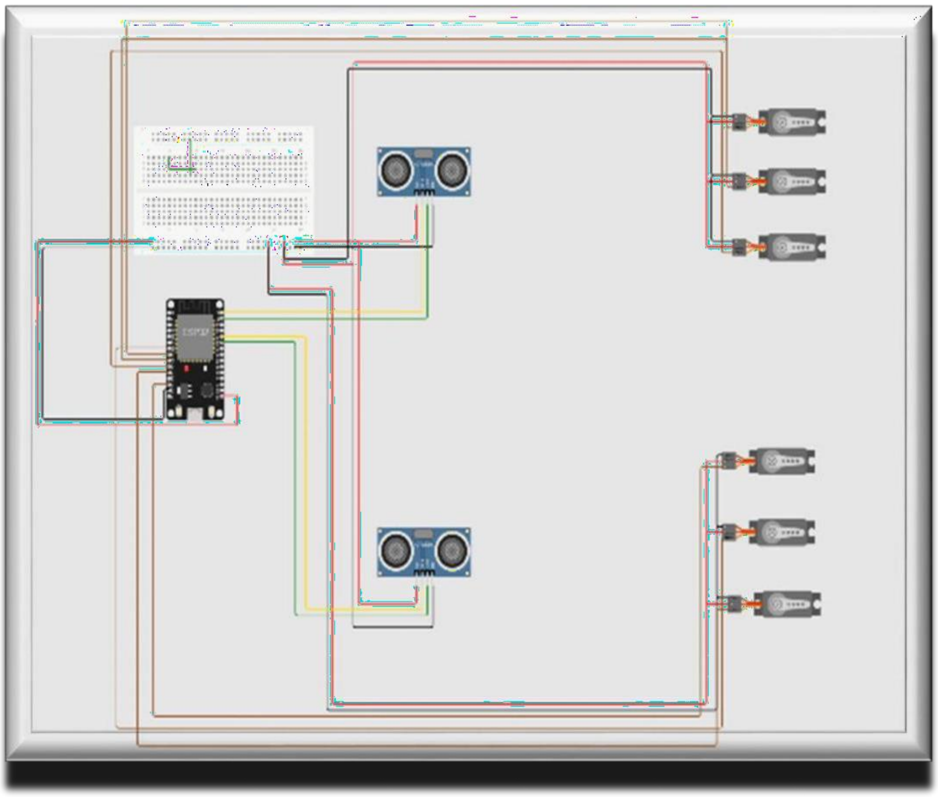


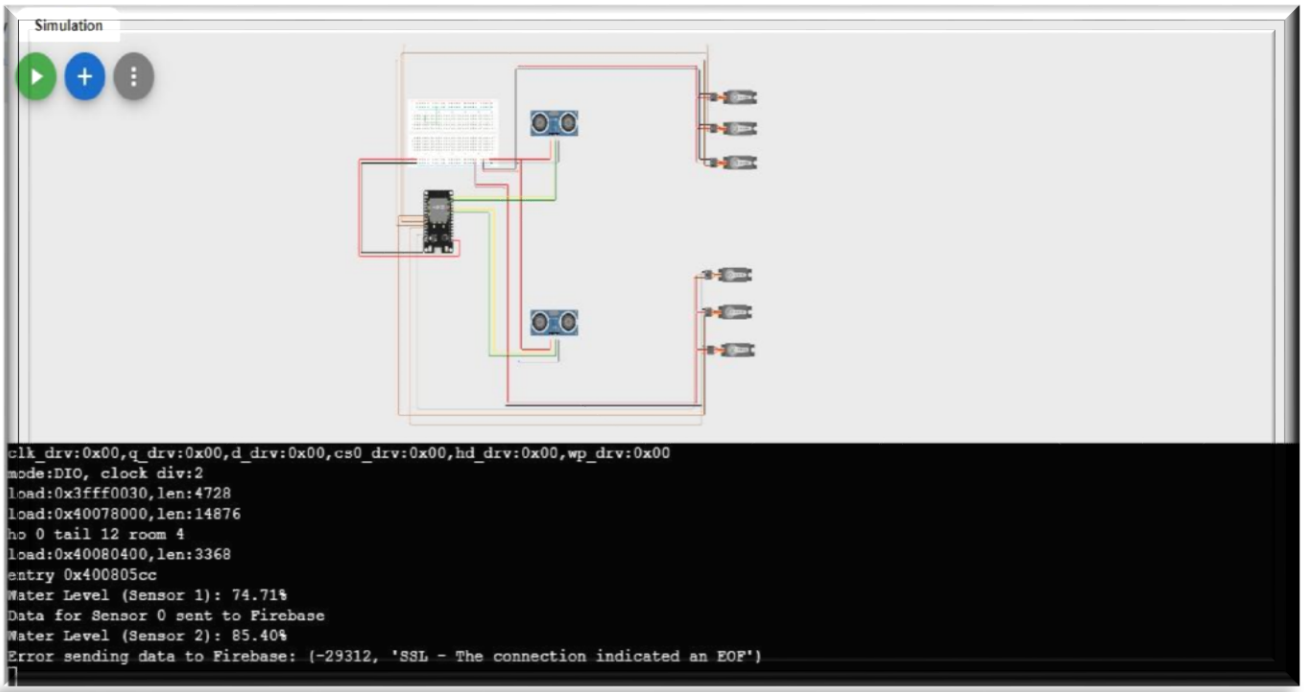
**Abstract:**
Water is essential for living things. So, we have to know about how to manage the water usage . where the water is being wasted ? find the places and we have to prevent this situation by SMART WATER SYSTEM . And we ensure the quality of water that is also important . so, monitor the pH value of the water by this plan .

**Project objective** :

A Wi-Fi module is connected to themicrocontrollerdevice which help to transfer the data to the cloud over internet. The ultrasonic sensor helps to measure the water level when the water flow reach certain level then the water flow can be stopped automatically by turning the motor off or close the water flow in pipe by the help of micro controller. The water flow sensor measure the quantity of water flow through the pipe in a given time, this data will be sent to cloud for storage and analysis purposes. The other sensor like ultra sonicsensor measure the water quality and help to determine whether the water is useful for drinking or any agricultural purposes.

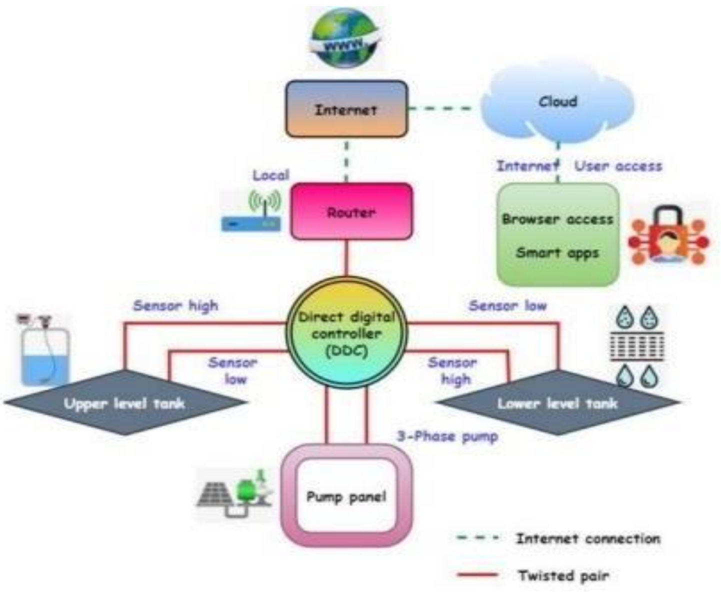
Process Diagram:





```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14876
ho 0 tail 12 room 4
load:0x40080400,len:3368
entry 0x400805cc
Water Level (Sensor 1): 74.71%
Data for Sensor 0 sent to Firebase
Water Level (Sensor 2): 85.40%
Error sending data to Firebase: (-29312, 'SSL - The connection indicated an EOF')
```

Process step Diagram:



**IOT Device setup:**

**Hardware Components :**

1.**ESP32** : The ESP32 serves the main control and runs the python program. It provide the necessary GPIO pins for connecting sensors

**2. Ultrasonic sensor :**
Ultrasonic sensor (HC-SR04) are used to measure the water level in the tank and You have configured two ultrasonic sensor in the project.

**3. Servo motors :**
Servo motors are act as valves in this project. Water can be distributed by this Servo motors. There are 6 servo motors used. Each ultrasonic sensor have three servo motors.

**4. Bread board :**
A breadboard (sometimes called a plugblock) is used for building temporary circuits. It is useful to designers because it allows components to be removed and replaced easily. It is useful to the person who wants to build a circuit todemonstrate its action, then to reuse the components in another circuit.

5. **Jumper wires** :
Jumper wires are used to establish connections between the ESP32's GPIO pins and the sensors.

**Wi-Fi Module**:
You should ensure that your ESP32 is connected to a Wi-Fi network, either using• built-in Wi-Fi or an external Wi-Fi module. The program relies on this network connection to send data to Firebase

**Power Supply**:
The ESP32 and sensors should be powered appropriately. The ESP32 can be powered through a USB power supply, and sensors may need a separate 5V supply. Ensure that all components share a common ground .

**Platform Devlopment**:

**1. Import Libraries:**
•machine: Provides access to the hardware components of the microcontroller.
•time: Allows you to work with time delays.

•urequests: Enables HTTP requests to be made to send data to Firebase.
•network: Used for connecting to Wi-Fi.

**2. Wi-Fi Setup:**
• You define the Wi-Fi credentials (SSID and password) and connect to the network.

**3. Pin Definitions:**
• You define GPIO pins for ultrasonic sensors (trig and echo), as well as for servo motors.

**4. Firebase Credentials:**
• You provide the URL of your Firebase Realtime Database and the secret for authentication.
**5. Functions:**
•measure_distance(trig_pin, echo_pin): Measures the distance using an ultrasonic sensor. This function triggers the sensor and calculates the distance based on the time taken for the echo pulse to return.
•calculate_water_level_percentage(current_distance, min_distance, max_distance): Calculates the water level percentage based on the current distance, minimum distance, and maximum distance.
•send_water_level_to_firebase(water_level_percentage, sensor_number): Sends the water level percentage to Firebase for a specific sensor. It uses a PATCH request to update the data in Firebase.

**6. Main Loop:**
• It's an infinite loop (while True) where the following steps occur repeatedly:
• For each sensor, it measures the water level, calculates the percentage, sends the data to Firebase, and controls the servo valve based on the water level.
• The loop also includes a sleep period of 1 second to control the data transmission frequency.

**Code implementation:**

```
import machine
import time
import urequests
import network
importujson
```

```
# Define your Wi-Fi credentials
wifi_ssid = 'Wokwi-GUEST'
wifi_password = ''

# Connect to Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(wifi_ssid, wifi_password)

# Wait for Wi-Fi connection
while not wifi.isconnected():pass

# Define GPIO pins for ultrasonic sensors
trig_pins = [21, 23] # Example pins, use the appropriate pins for your setup
echo_pins = [19, 22] # Example pins, use the appropriate pins for your setup

# Define GPIO pins for servo motors (6 servos)
servo_pins = [13, 14, 25, 26, 27, 33] # Example pins, use the appropriate pins foryour setup
servos = [machine.PWM(machine.Pin(pin), freq=50, duty=0) for pin in servo_pins]

# Firebase Realtime Database URL and secret
firebase_url = 'https://iot-smart-water-manageme-6f2dc-default-rtdb.asia-southeast1.firebasedatabase.app'
firebase_secret = 'PuZIQmVtENsSNLybJ4wDwEHzXUZiiKxsCgh7j6SS'



# Function to measure distance using ultrasonic sensor
defmeasure_distance(trig_pin, echo_pin):
trig = machine.Pin(trig_pin, machine.Pin.OUT)
echo = machine.Pin(echo_pin, machine.Pin.IN)

# Ensure the trigger pin is low
trig.off()
time.sleep_us(2)

# Generate a 10us pulse on the trigger pin
```

```
trig.on()
time.sleep_us(10)
trig.off()

# Measure the duration of the pulse on the echo pin
whileecho.value() == 0:
pulse_start = time.ticks_us()
whileecho.value() == 1:
pulse_end = time.ticks_us()
# Calculate the duration of the pulse
pulse_duration = time.ticks_diff(pulse_end, pulse_start)

# Calculate the distance based on the speed of sound
distance = (pulse_duration / 2) / 29.1 # Speed of sound in air is approximately
343 m/s
return distance

# Function to calculate the water level percentage
defcalculate_water_level_percentage(current_distance, min_distance,
max_distance):
ifcurrent_distance<min_distance:
return 0
elifcurrent_distance>max_distance:
return 100
else:
return ((current_distance - min_distance) / (max_distance - min_distance)) *
100
# Define the minimum and maximum distances your sensor can detect
min_distance = 2 # Example minimum distance (in cm)
max_distance = 400 # Example maximum distance (in cm)




# Function to send water level to Firebase
defsend_water_level_to_firebase(water_level_percentage, sensor_number):
data = {'WaterLevel': water_level_percentage}
url =  f'{firebase_url}/sensor_{sensor_number}.json?auth={firebase_secret}'

try:
```

```
response = urequests.patch(url, json=data)
if response.status_code == 200:
print(f"Data for Sensor {sensor_number} sent to Firebase")
else:
print(f"Failed to send data to Firebase. Status code:
{response.status_code}")
except Exception as e:
print(f"Error sending data to Firebase: {str(e)}")
while True:


# Measure water level using ultrasonic sensors
for sensor_number in range(len(trig_pins)):
ultrasonic_distance = measure_distance(trig_pins[sensor_number],
echo_pins[sensor_number])

# Calculate and print water level percentage
water_level_percentage =
calculate_water_level_percentage(ultrasonic_distance, min_distance, max_distance)
print(f"Water Level (Sensor {sensor_number + 1}):
{water_level_percentage:.2f}%")

# Send data to Firebase
send_water_level_to_firebase(water_level_percentage, sensor_number)

# Control the servo valve based on the water level percentage
if water_level_percentage < 50:
# Adjust servo control logic based on the water level percentage
servos[sensor_number].duty(512) # Set servo duty cycle to a value to control it
else:
servos[sensor_number].duty(0) # Set servo duty cycle to 0 to stop it
send_water_level_to_firebase(water_level_percentage, sensor_number)
time.sleep(1)
 # Adjust the sleep duration as needed
```

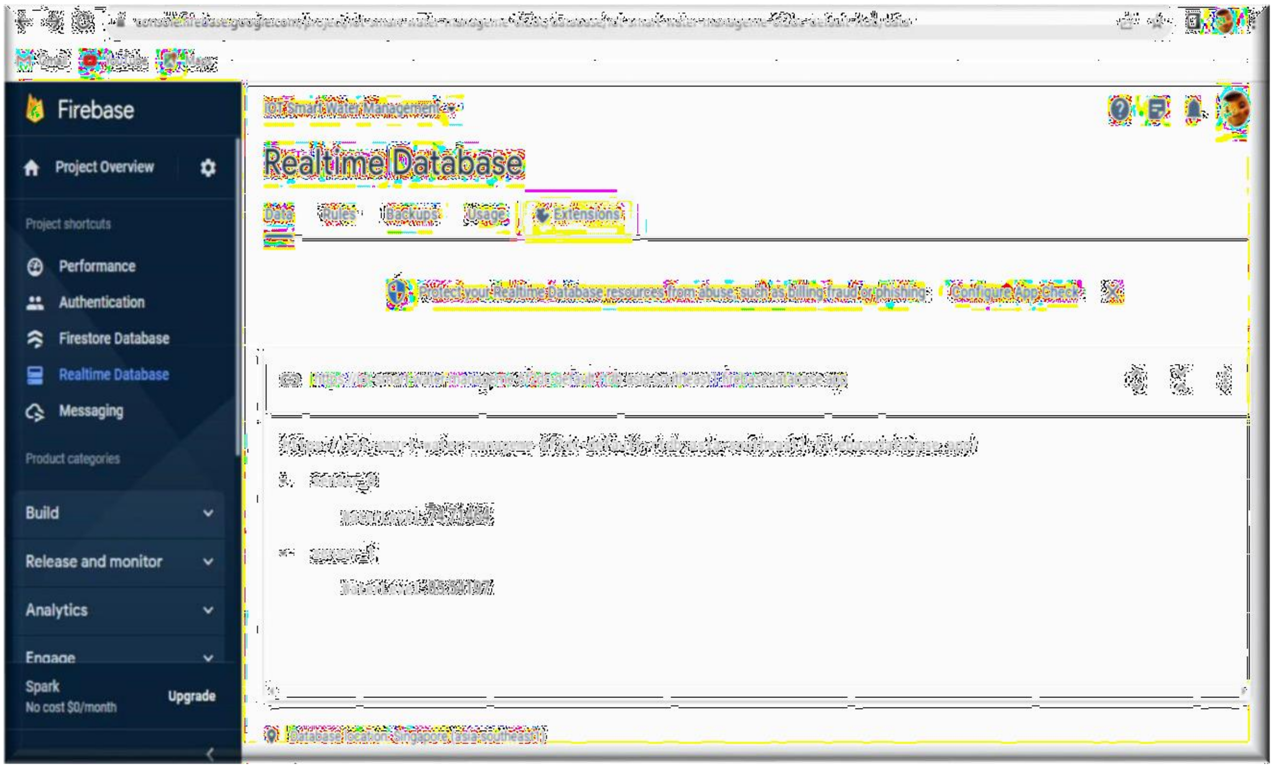
- This is our Python program which is used to design for a smartwater consumption monitoring system using ultrasonic and othersensors to monitor the water consumption and send this information to a Firebase Real-time Database

**Firebase Database:**

Real time Database: Firebase RealtimeDatabase is a cloud-hosted NoSQL database provided by Firebase,□ a mobile and web application development platform that is now part of Google's cloud offerings. Firebase Realtime Database is designed for real-time data synchronization and is commonly used in

**Mobile application Development:**

## Prerequisites:



- Android studio  installed
- A firebase account setup for the project
- An ESP32 microcontroller programmed with Micro python
- A basic understanding of python programming

<u>Step 1: Setting Up the Development Environment</u>

<u>Install Android Studio:</u>

- If not already installed, download and install Android Studio from the official website: [https://developer.android.com/studio](https://developer.android.com/studio).

Configure Android Studio:

- Ensure you have the necessary SDKs and tools installed for Android app development.

Firebase Setup:

- If not done already, create a Firebase project at https://console.firebase.google.com/ and configure it for your Android app.

## Step 2: Designing the App Interface

Implement the UI:

- Use Android Studio's Layout Editor to create the app's user interface.

1. **Open Android Studio:** Launch Android Studio and open your Android app project.
2. **Navigate to XML Layout File:** In the project explorer, navigate to the "res" folder, then "layout," and find the XML layout file where you want to design your user interface. Double-click the XML file to open it.
3. **Open Layout Editor:** Once you've opened the XML layout file, you'll see two tabs at the bottom of the XML editor: "Text" and "Design." Click on the "Design" tab to open the Layout Editor.
4. **Palette:** On the left side of the Layout Editor, you'll find the "Palette" panel.It contains various UI components such as buttons, text views, image views, and more. You can drag and drop these components onto the layout canvas tobuild your interface

**Component Tree:**

On the right side of the Layout Editor, you'll find the "Component Tree" panel. It displays the hierarchy of UI components on yourlayout. You can select and manipulate components in this panel.

5. **Attributes Panel:** Below the "Component Tree" panel, you'll find the "Attributes" panel. This panel allows you to customize the properties of selected UI components. You can change attributes like text, color, size, andpositioning.
6. **Layout Canvas:** The central area of the Layout Editor is the layout canvas. This is where you visually arrange and design your app's user interface. You can drag and drop components onto the canvas, adjust their positions, and seea real-time preview of how your layout will appear in the app.
7. **Preview:** Above the layout canvas, there's a "Preview" panel that shows a livepreview of how your layout will look on different devices and orientations. You can switch between various screen sizes and orientations to ensure your layout is responsive.
8. **Zoom and Pan:** You can zoom in and out of the layout canvas by using the zoom slider in the bottom right corner. You can also pan around the canvas towork on different parts of your layout.
9. **Design Toolbar:** At the top of the Layout Editor, you'll find the design toolbar. It contains options for adding constraints, aligning components, andcustomizing the layout.
10. **Adding Constraints:** Android Studio uses a constraint-based layout system (Constraint Layout) by default. To position UI components, you can add constraints that specify how they relate to other components or the parent layout. Constraints help your layout adapt to different screen sizes.

11. **Preview Your Layout:** As you design your user interface, use the "Preview" panel to see how your layout will appear on different devices and orientations.Make adjustments as needed to ensure a responsive design.
12. **Save Your Layout:** Don't forget to save your layout by clicking the "Save"button in the top-left corner.
13. **XML Code View:** If you need to make fine-grained adjustments or add complex attributes, you can switch to the "Text" tab to edit the XML codedirectly.

## Layout program:

## Login page :

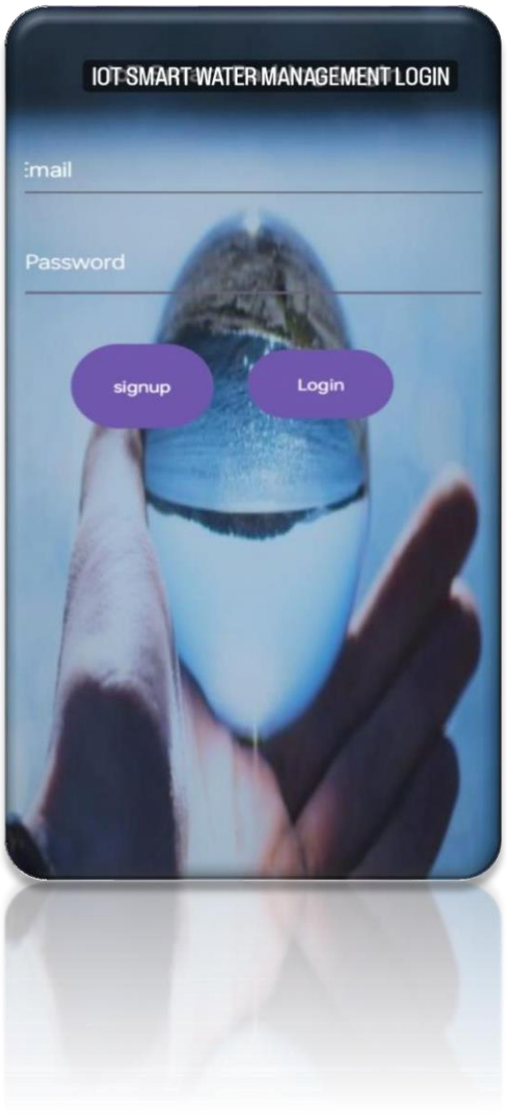
## Activity_main.xml

```
<?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"
    android:background="@drawable/image21"
    android:padding="16dp"
    tools:ignore="ExtraText">
```

```
<EditText
    android:id="@+id/emailEditText"
    android:layout_width="349dp"
    android:layout_height="54dp"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:textColor="#FAFAFA"
    android:textColorHint="#FFFFFF"
    app:layout_constraintBottom_toTopOf="@+id/passwordEditText"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.865" />

<EditText
    android:id="@+id/passwordEditText"
    android:layout_width="335dp"
    android:layout_height="67dp"
    android:hint="Password"
    android:inputType="textPassword"
    android:textColor="#F1EEEE"
    android:textColorHighlight="#F8F5F5"
    android:textColorHint="#F3F0F0"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.592"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.257" />

<Button
    android:id="@+id/loginButton"
    android:layout_width="105dp"
    android:layout_height="64dp"
    android:layout_marginEnd="64dp"
    android:text="Login"
    android:textColorHighlight="#CD7C7C"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.419"
    app:rippleColor="#E14545"
    app:strokeColor="#E85656" />

<Button
    android:id="@+id/signup"
    android:layout_width="102dp"
    android:layout_height="77dp"
    android:layout_marginEnd="24dp"
    android:text="signup"
    android:textColorHighlight="#DC4040"
    android:textColorLink="#D84577"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/loginButton"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.42" />

<TextView
    android:id="@+id/textView"
    android:layout_width="251dp"
    android:layout_height="36dp"
    android:layout_marginBottom="24dp"
    android:fontFamily="sans-serif-black"
    android:text="IoT Smart water management"
    android:textAlignment="center"
    android:textColor="#F8F5F5"
    android:textColorHighlight="#FAF9F9"
    android:textColorHint="#FFFFFF"
    android:textColorLink="#FBF9F9"
```

```
android:textSize="20sp"
android:textStyle="bold"
app:layout_constraintBottom_toTopOf="@+id/emailEditText"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.506"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.859" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig: LOGIN PAGE

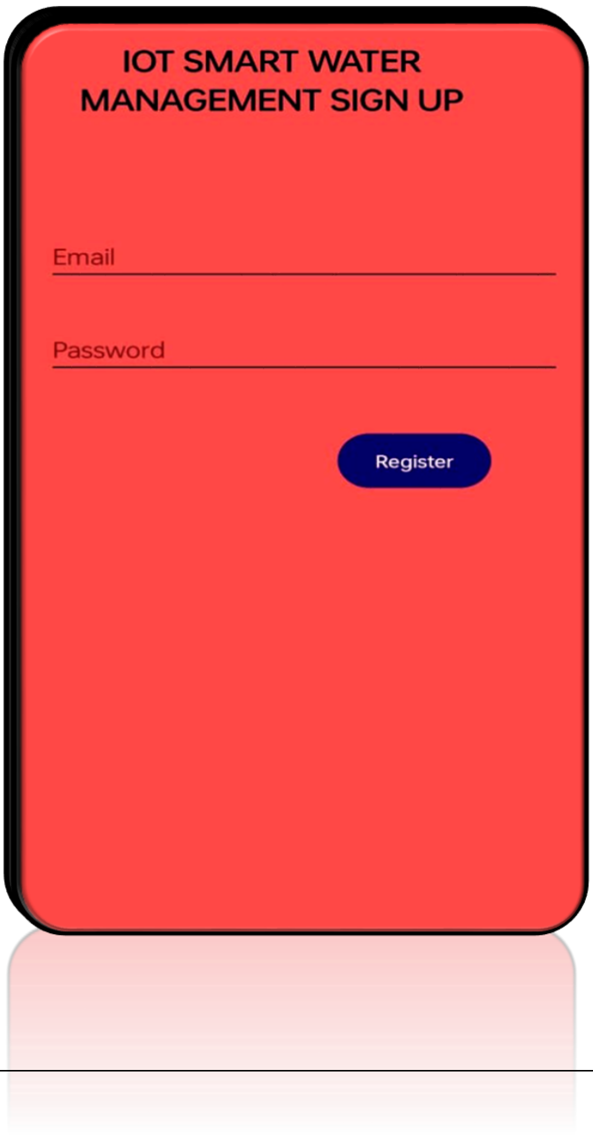


REGISTRATION PAGE :
Activity_login.xml

```
 <?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"

    android:background="#FF9191"

    android:padding="16dp">


    <EditText

        android:id="@+id/emailEditText"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:hint="Email"

        android:inputType="textEmailAddress"

        app:layout_constraintBottom_toTopOf="@+id/passwordEditText"

        app:layout_constraintTop_toTopOf="parent"

        app:layout_constraintVertical_bias="0.844"

        tools:layout_editor_absoluteX="16dp" />


    <EditText
    android:id="@+id/passwordEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="436dp"
    android:hint="Password"
    android:inputType="textPassword"
    app:layout_constraintBottom_toBottomOf="parent"
    tools:layout_editor_absoluteX="16dp" />

<Button
    android:id="@+id/registerButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Register"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.803"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/passwordEditText"
    app:layout_constraintVertical_bias="0.094" />

<TextView
    android:id="@+id/textView2"
```

```
        android:layout_width="286dp"

        android:layout_height="69dp"

        android:text="IoT Smart water management Sign Up"

        android:textAlignment="center"

        android:textAllCaps="true"

        android:textSize="24sp"

        android:textStyle="bold"

        tools:layout_editor_absoluteX="48dp"

        tools:layout_editor_absoluteY="39dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig: REGISTRATION PAGE



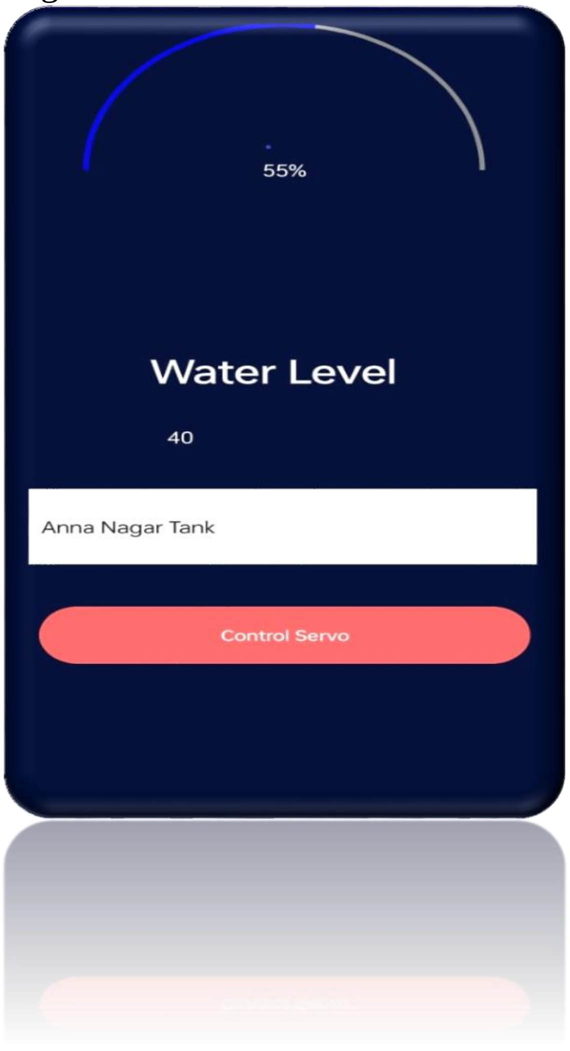
Activity_firebase:

```
<?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:id="@+id/relativeLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000C33"
    tools:context=".MainActivity">

    <!-- Custom Meter Gauge -->


    <com.example.smartwatermanagement.GaugeView

        android:id="@+id/gaugeView"
        android:layout_width="328dp"
        android:layout_height="254dp"
        android:layout_centerInParent="true"
    android:rotation="0"
    android:rotationY="0"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.493"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.033" />

<TextView
    android:id="@+id/waterFlowTextView"
    android:layout_width="150dp"
    android:layout_height="32dp"
    android:text="Water Flow: 0 L/min"
    android:textColor="#FFFFFF"
    android:textSize="16sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.533" />
```

```
<Spinner
    android:id="@+id/locationSpinner"
    android:layout_width="326dp"
    android:layout_height="66dp"
    android:background="#FDFDFD"
    android:backgroundTint="#FFFFFF"
    android:entries="@array/location_entries"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.471"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.645" />

<Button
    android:id="@+id/controlButton"
    android:layout_width="315dp"
    android:layout_height="58dp"
    android:backgroundTint="#FB6565"
    android:text="Control Servo"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.787" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="243dp"
        android:layout_height="39dp"
        android:text="Water Level"
        android:textAlignment="center"
        android:textColor="#FDFDFD"
        android:textSize="32sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.44"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.439" />



</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig: WATER LEVEL INDICATION



## Firebase.java

```
package com.example.smartwatermanagement;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class Firebase extends AppCompatActivity {

    private GaugeView distanceGauge;
    private TextView humidityGauge;
    private Spinner locationSpinner;
    private DatabaseReference sensorDataRef;

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

        locationSpinner = findViewById(R.id.locationSpinner);

        // Initialize Firebase
        FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();

        // Set up initial DatabaseReference based on the default spinner selection
        updateSensorDataRef(locationSpinner.getSelectedItemPosition(),
firebaseDatabase);

        // Get references to GaugeView widgets in your layout
        distanceGauge = findViewById(R.id.gaugeView);
        humidityGauge = findViewById(R.id.waterFlowTextView);

        // Set up a listener to retrieve data from Firebase based on the selected
sensor
        sensorDataRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Retrieve values from the dataSnapshot
                    DataSnapshot waterLevelSnapshot =
dataSnapshot.child("WaterLevel");

                    if (waterLevelSnapshot.exists()) {
                        Integer distance = waterLevelSnapshot.getValue(Integer.class);

                        if (distance != null) {
                            int distanceValue = distance;
                            // Now you can safely use distanceValue
                            distanceGauge.setValue(distanceValue);
                            humidityGauge.setText("40"); // Assuming you want to set a
string, not an integer
                            Toast.makeText(Firebase.this, "Waterlevel: " + distanceValue,
Toast.LENGTH_SHORT).show();
                        } else {
                            Toast.makeText(Firebase.this, "WaterLevel itself exists but is
null", Toast.LENGTH_SHORT).show();
                        }
                    } else {
                        // Handle the case where "WaterLevel" does not exist in the
dataSnapshot
                        Toast.makeText(Firebase.this, "WaterLevel itself doesn't exist",
Toast.LENGTH_SHORT).show();
                    }
                } else {
                    // Handle the case where the dataSnapshot itself doesn't exist
                    Toast.makeText(Firebase.this, "dataSnapshot itself doesn't exist",
Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                // Handle database error
            }
        });
        // Add a listener to the spinner to change the selected sensor
        locationSpinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parentView, View
selectedItemView, int position, long id) {
                // When the spinner selection changes, update the DatabaseReference
                updateSensorDataRef(position, firebaseDatabase);
            }

            @Override
            public void onNothingSelected(AdapterView<?> parentView) {
                // Do nothing
            }
        });
    }

    // Method to update the DatabaseReference based on spinner selection
    private void updateSensorDataRef(int selectedSensor, FirebaseDatabase
firebaseDatabase) {
        if (selectedSensor == 0) {
            sensorDataRef = firebaseDatabase.getReference("sensor_0");
        } else if (selectedSensor == 1) {
            sensorDataRef = firebaseDatabase.getReference("sensor_1");
        }
    }
}
```

## Program Description:

Overview of what the code does:

1. **Imports**: The necessary libraries are imported, including Firebase and Android UI components.

2. **Class Definition**: The class **Firebase** extends **AppCompatActivity**, indicating that it represents an Android activity.

3. **Member Variables**:

- **distanceGauge**, **humidityGauge**, **locationSpinner**: These are references to UI elements (GaugeView, TextView, and Spinner) in the layout XML file.
- **sensorDataRef**: This is a reference to the Firebase database.

4. **onCreate Method**:
    - This is called when the activity is created.
    - It sets the content view to the layout defined in **activity_firebase.xml**.
    - It initializes the member variables and gets references to UI elements.
    - It sets up the initial DatabaseReference based on the default spinner selection.
    - It sets up listeners for the sensor data updates and spinner item selections.
5. **updateSensorDataRef Method**:
    - This method takes an integer **selectedSensor** and a **FirebaseDatabase** instance as arguments.
    - Depending on the value of **selectedSensor**, it updates the **sensorDataRef** with the appropriate reference from the Firebase database.
6. **onDataChange Method**:
    - This method is a callback that is triggered whenever data in the Firebase database changes.

- It retrieves the water level data from the database and updates the UI elements (distanceGauge and humidityGauge) accordingly.

7. **onCancelled Method**:

   - This method is a callback that is triggered if there's an error when trying to read from the database.

8. **onItemSelected Method**:

   - This method is a callback that is triggered when an item in the spinner is selected.
   - It calls **updateSensorDataRef** to update the DatabaseReference based on the selected sensor.

This Android application is designed for smart water management. It integrates with Firebase, a real-time database service, to retrieve sensor data. The app provides a user interface to view water level information from different sensors.

**Key Features:**

1. **Firebase Integration:**

   - The app leverages Firebase, a cloud-based platform, for real-time database functionality.
   - It establishes a connection to Firebase to retrieve sensor data.

2. **User Interface:**

   - The app's UI includes elements like Spinner, GaugeView, and TextView to display sensor data.
   - The Spinner allows the user to select different sensor locations.

3. **Sensor Data Display:**
    - The GaugeView visually represents water level data, providing an intuitive display for the user.
    - The TextView shows additional information, such as humidity or flow data.
4. **Dynamic Data Retrieval:**
    - The app dynamically updates sensor data based on the user's selection in the Spinner.
    - It listens for changes in the Firebase database and updates the UI accordingly.
5. **Error Handling:**
    - The app incorporates error handling, including cases where data may be missing or null.

**Workflow:**

1. **App Initialization:**
    - Upon launching, the app sets up the initial UI components and establishes a connection to Firebase.
2. **Sensor Selection:**
    - The Spinner allows the user to choose a specific sensor location.
3. **Firebase Data Retrieval:**
    - The app listens for changes in the selected sensor's data within Firebase.
4. **Data Update:**
    - When sensor data changes, the app retrieves the new values and updates the UI elements.

5. **Error Handling:**
    - The app handles potential errors, providing user-friendly toast messages to indicate any issues.

**Usage Scenario:**

- An end-user, such as a water management professional, opens the app on their Android device.
- They select a specific sensor location from the Spinner.
- The app fetches real-time water level data from Firebase and displays it using the GaugeView and TextView.
- If there are any issues with data retrieval, the app notifies the user with a toast message.

**Potential Extensions:**

- The app's functionality could be expanded to include additional sensors or parameters.
- It could incorporate features like historical data tracking, notifications, or user authentication for enhanced functionality.

## Conclusion:

In this paper, a prototype water monitoring system using IOT is presented. For this some sensors are used. The collected data from the all the sensors are used for analysis purpose for better solution of water problems. The data is sends to the cloud server via Wi-Fi module ESP8266. So this application will be the best challenger in real time monitoring & control system and use to solve all the water related problems.