**Phase 5**

**Smart water fountain**

# PROJECT OBJECTIVES

The objectives of a smart water fountain project are to save water through efficient management, automate its operation using sensors and controls, make it energy-efficient, enhance user experience with features like adjustable displays and music, monitor data on water quality and performance, provide maintenance alerts, prioritize sustainability with eco-friendly practices, create an aesthetically pleasing design, ensure accessibility for all, and manage costs effectively. These goals ensure a sustainable, technologically advanced, and visually appealing water fountain.

# PROJECT DEFINITION

It has been researched and found that there has been the following issues with water fountains. The problems are listed below.

* It has been researched and found that there has been the following issues with water fountains. The problems are listed below.
* Fountains require regular cleaning, upkeep, and repairs. Access to components, pumps, and plumbing can be challenging, and maintenance costs can escalate if not planned for adequately.
* Traditional water fountains often waste water due to continuous flow, leaks, or inefficient cooling systems.
* Compliance with local building codes, zoning regulations, and environmental laws can be complex and may vary depending on the location of the fountain.

# DESIGN THINKING

Having understood the problem statement , we can approach the solution by the designing idealogies.

* Conduct user research to understand the needs, preferences, and pain points of potential users of IoT-based smart water fountains.
* Brainstorm ideas that leverage IoT capabilities, including sensors, connectivity, and data analysis, to enhance water fountains.
* Consider features like real-time water quality monitoring, predictive maintenance, touchless operation, and user app integration.
* Evaluate the impact of IoT technology by analyzing data collected from sensors and user interactions.
* Measure water conservation, maintenance efficiency, and user engagement facilitated.
* If the pilot is successful, plan for the widespread deployment of IoT-based smart water fountains across various locations andConsider scalability, IoT platform integration, and user scalability for larger deployments.

# INNOVATIVE IDEAS

* We will be using the Arduino UNO microcontroller suit the best for our project.
* An innovative idea for smart water fountains could be to incorporate a user-friendly app that allows people to customize their water preferences.
* Users could adjust the water temperature, add flavors or electrolytes, and control the water flow rate, all from their smartphones.
* Additionally, sensors could monitor water quality and send alerts for filter replacements or maintenance, promoting sustainable and clean drinking water.

**SENSORS**

* Water Level Sensor, Temperature Sensor, Water Quality Sensor, Motion Sensor, Ultrasonic Distance Sensor, Light Sensor, Pressure Sensor, Flow Sensor, Humidity Sensor, Sound Sensor. These are the sensor, we will used for our project.

**CONNECTIVITY**

* Wi-Fi: Many smart water fountains connect to your home's Wi-Fi network, allowing you to control them through a mobile app or web interface from anywhere with an internet connection.
* Fountains are connected to the internet for real-time data transmission. Use protocols like MQTT or HTTP for communication.

**PROTOCOL**

* After research, the protocol which we chose for our solution for message queuing and reception is HTTP/HTTPS.
* Because it is simple and perfectly suitable for our innovative solution.

**CLOUD**

* Although many cloud services are available like Google, Amazon and others, it is available in paid version.
* So, we chose BEECEPTOR cloud which supports HTTP/HTTPS protocol for solution.

# **PUBIC PLATFORM**

Creating a public platform for smart water fountains could involve integrating IoT (Internet of Things) technology to monitor and manage water fountains in public spaces.

* Sensor Integration: Install sensors to monitor water quality, temperature, and flow rates. These sensors can transmit data to a central platform.
* Centralized Dashboard: Develop a web-based dashboard for administrators to monitor fountain status, water quality, and maintenance needs.
* Mobile App: Create a user-friendly mobile app that allows the public to locate nearby smart fountains, check water quality information, and receive notifications.
* Water Quality Analysis: Implement algorithms to analyze water quality data and generate alerts if issues are detected.
* Maintenance Scheduling: Set up a maintenance scheduling system based on usage data and sensor readings to ensure fountains are clean and functioning optimally.
* Water Conservation: Integrate features to promote water conservation, such as automatic shut-off during non-peak hours or when water quality is compromised.
* Public Engagement: Encourage public involvement through features like user ratings, feedback submission, and social media integration.
* Security: Implement robust security measures to protect data and prevent tampering with fountain functionality.
* APIs and Open Data: Provide APIs for developers to access the data, allowing them to create third-party apps and services, fostering innovation.

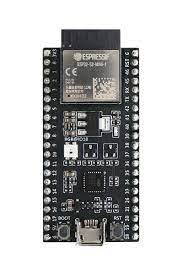
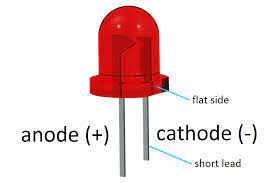
# DESCRIPTION

This document is describing the process of developing the project with the mentioned components in phase 2 innovation segment. For our project we are going to use ultrasonic Sensor and ESP32 module for smart water fountain.

# COMPONENTS REQUIRED

* ESP32 module.
* Stepper motor.
* Water pump.
* Relay module.
* Ultrasonic sensor.
* Wokwi virtual components.

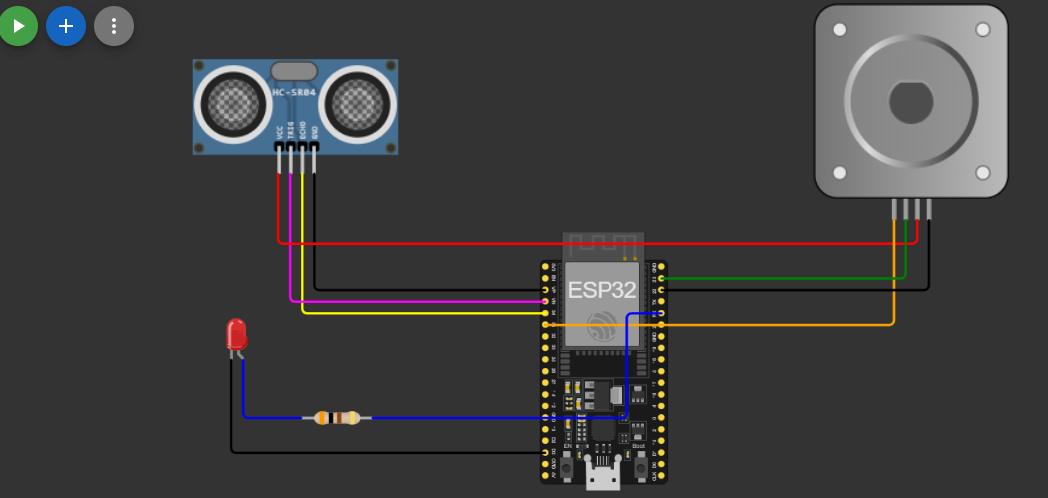
# IOT DEVICES



# WORKING DESCRRIPTION

Smart water fountains are innovative devices designed to enhance the traditional concept of public water dispensers. They incorporate advanced technology to provide a more efficient and user-friendly experience. These fountains are equipped with sensors and filtration systems that ensure the water is clean and safe to drink. They often feature touchless operation, allowing users to fill their bottles or cups without physical contact, promoting hygiene. Additionally, smart water fountains can monitor water usage, track water quality, and even offer real-time data through mobile apps, contributing to water conservation efforts and promoting sustainability. These fountains represent a fusion of convenience, sustainability, and technology, making them a valuable addition to modern urban environments.

# CIRCUIT DIGRAM

****

Here is the .json program for the above diagram as follows:

{

"version": 1,

"author": "Anonymous maker",

"editor": "wokwi",

"parts": [

{ "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 9.6, "left": 187.24, "attrs": {} },

{

"type": "wokwi-stepper-motor",

"id": "stepper1",

"top": -178.79,

"left": 413.23,

"attrs": { "size": "17" }

},

{ "type": "wokwi-hc-sr04", "id": "ultrasonic1", "top": -132.9, "left": -100.1, "attrs": {} },

{

"type": "wokwi-resistor",

"id": "r1",

"top": 157.55,

"left": -9.6,

"attrs": { "value": "300" }

},

{ "type": "wokwi-led", "id": "led1", "top": 73.2, "left": -82.6, "attrs": { "color": "red" } }

],

"connections": [

[ "esp:TX", "$serialMonitor:RX", "", [] ],

[ "esp:RX", "$serialMonitor:TX", "", [] ],

[ "ultrasonic1:GND", "esp:VP", "black", [ "v0" ] ],

[ "ultrasonic1:ECHO", "esp:34", "yellow", [ "v0" ] ],

[ "ultrasonic1:TRIG", "esp:VN", "magenta", [ "v0" ] ],

[ "ultrasonic1:VCC", "stepper1:B+", "red", [ "v57.6", "h528" ] ],

[ "esp:35", "stepper1:A-", "orange", [ "h0" ] ],

[ "esp:23", "stepper1:A+", "green", [ "h0" ] ],

[ "stepper1:B-", "esp:22", "black", [ "v0" ] ],

[ "r1:2", "esp:RX", "blue", [ "v0", "h210", "v-86.4" ] ],

[ "led1:A", "r1:1", "blue", [ "v0" ] ],

[ "led1:C", "esp:D3", "black", [ "v0" ] ]

],

"dependencies": {}

}

# PROGRM FOR SIMULATION

As our project is based on IOT, we need to code the Instructions to the controller for performing our desired function. Here is the code for simulating our project. Before that, we have to ensure the required libraries are installed in wowki platform .

#include <WiFi.h>

#include <Stepper.h>

#include <NewPing.h>

const char\* ssid = "your\_wifi\_ssid";

const char\* password = "your\_wifi\_password";

const int stepsPerRevolution = 2048; // Adjust this value based on your stepper motor

Stepper myStepper(stepsPerRevolution, 16, 17, 18, 19);

const int motorSpeed = 5; // Adjust this value based on your motor's speed requirements

const int waterPumpPin = 23;

const int ultrasonicTriggerPin = 24;

const int ultrasonicEchoPin = 25;

const int ledPin = 26;

NewPing sonar(ultrasonicTriggerPin, ultrasonicEchoPin, 200); // Adjust the max distance (200cm) as needed

void setup() {

Serial.begin(115200);

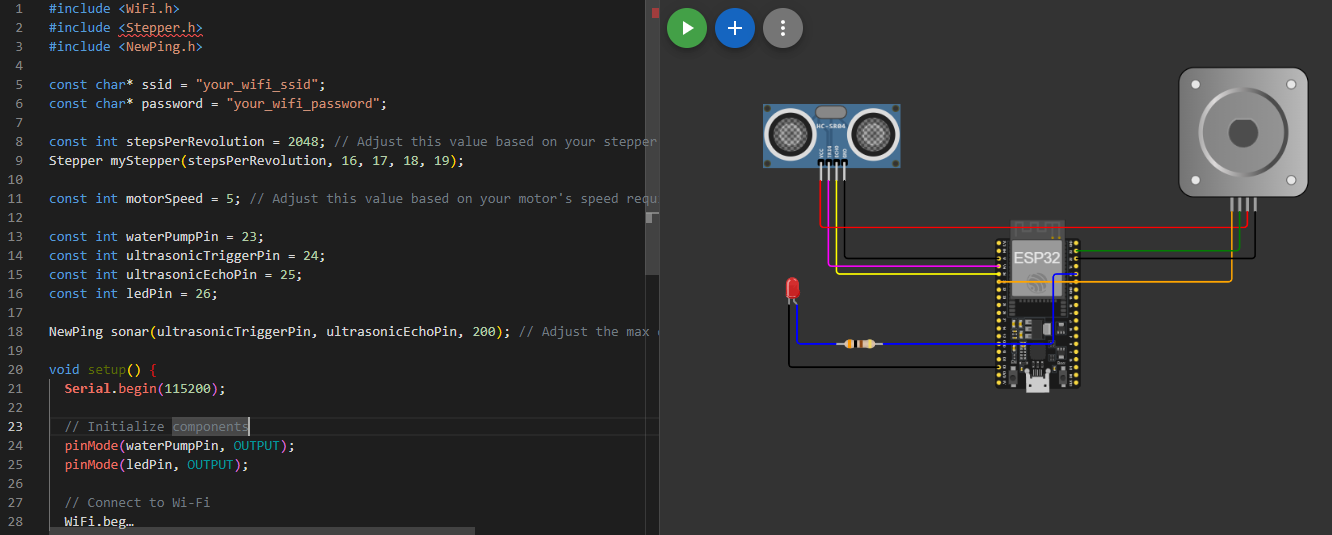
// Initialize components

pinMode(waterPumpPin, OUTPUT);

pinMode(ledPin, OUTPUT);

// Connect to Wi-Fi

WiFi.beg…



It includes the necessary libraries for WiFi, stepper motor control, and ultrasonic sensor.

It defines the pins for the components and sets up the Stepper motor.

In the setup() function, it initializes the components, sets up the Wi-Fi connection, and waits for a successful connection.

In the loop() function, it checks the water level using the ultrasonic sensor.

If the water level is below 10 cm (you can adjust this threshold), it turns on the LED and water pump.

It then uses the stepper motor to dispense water (adjust the steps and delay as needed).

After dispensing, it turns off the water pump and LED.

Remember to replace "your\_wifi\_ssid" and "your\_wifi\_password" with your actual Wi-Fi credentials.

# CODE FOR AUTOMATIC FOUNTAIN

To make the water fountain work automatically in the morning and evening, you'll need to implement a time-based scheduling system. This can be achieved using the built-in clock of the ESP32 and the Time.h library. Below is the modified code.

#include<wifi.h>

#include <Stepper.h>

#include <NewPing.h>

#include <TimeLib.h>

const char\* ssid = "your\_wifi\_ssid";

const char\* password = "your\_wifi\_password";

const int stepsPerRevolution = 2048; // Adjust this value based on your stepper motor

Stepper myStepper(stepsPerRevolution, 16, 17, 18, 19);

const int motorSpeed = 5; // Adjust this value based on your motor's speed requirements

const int waterPumpPin = 23;

const int ultrasonicTriggerPin = 24;

const int ultrasonicEchoPin = 25;

const int ledPin = 26;

NewPing sonar(ultrasonicTriggerPin, ultrasonicEchoPin, 200); // Adjust the max distance (200cm) as needed

const int morningHour = 7; // Set the hour for the morning activation

const int eveningHour = 19; // Set the hour for the evening activation

void setup() {

Serial.begin(115200);

// Initialize components

pinMode(waterPumpPin, OUTPUT);

pinMode(ledPin, OUTPUT);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

// Initialize Time library

configTime(0, 0, "pool.ntp.org", "time.nist.gov"); // Configure NTP servers

}

void loop() {

// Update time

struct tm timeinfo;

if(!getLocalTime(&timeinfo));

Serial.println("Failed to obtain time");

return;

}

int currentHour = timeinfo.tm\_hour;

if ((currentHour == morningHour || currentHour == eveningHour) && timeinfo.tm\_min == 0 && timeinfo.tm\_sec == 0) {

// It's the specified activation hour and minute

// Check water level

int distance = sonar.ping\_cm();

Serial.print("Distance: ");

Serial.print(distance);

Serial.println(" cm");

if (distance < 10) {

// Water level is low, turn on the LED and water pump

digitalWrite(ledPin, HIGH);

digitalWrite(waterPumpPin, HIGH);

// Dispense water using the stepper motor

myStepper.setSpeed(motorSpeed);

myStepper.step(stepsPerRevolution);

// Wait for dispensing to complete

delay(5000); // Adjust as needed

// Turn off water pump and LED

digitalWrite(waterPumpPin, LOW);

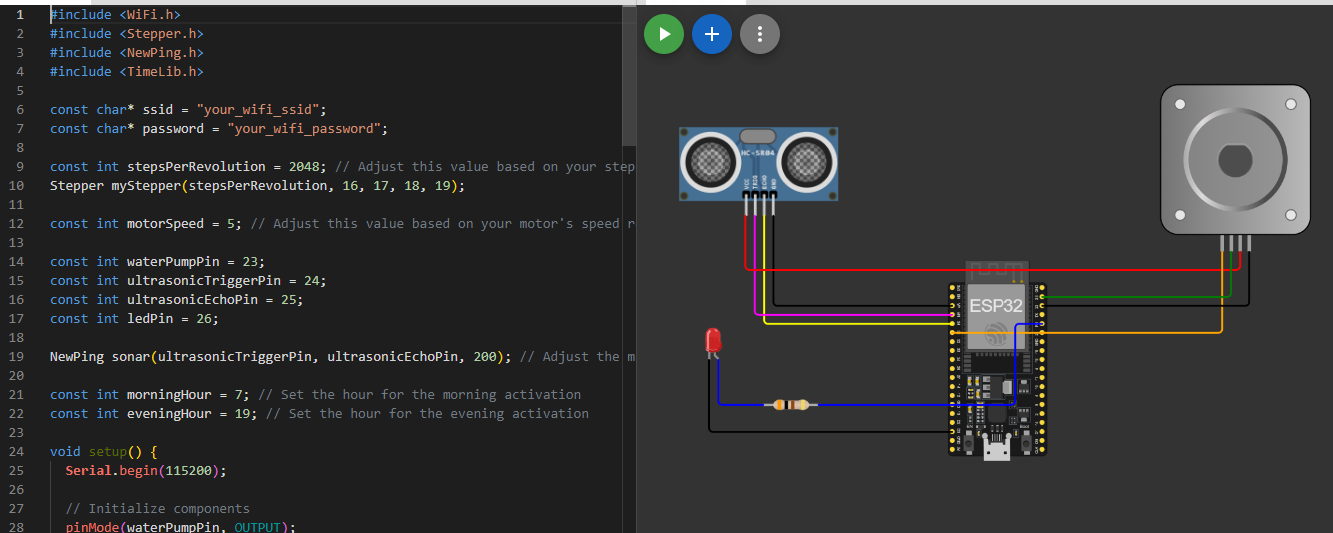
digitalWrite(ledPin, LOW);

}

}

delay(1000); // Adjust delay as needed for your application

}



In this code, I've added a morningHour and eveningHour variable, which you can set to the desired activation times. The code will now check the current time, and if it matches the specified activation hours and it's the start of a new hour (minutes and seconds are both zero), it will proceed with the fountain operation.This code will automatically activate the water fountain at the specified times in the morning and evening.

# Code for indication and notification

To implement water level indication and send notifications to the user, you'll need to integrate a method for sending messages. Below is the code with water level indication and sending a notification to the user.

#include <WiFi.h>

#include <Stepper.h>

#include <NewPing.h>

#include <TimeLib.h>

#include <Pushbullet.h> // Include the Pushbullet library

const char\* ssid = "your\_wifi\_ssid";

const char\* password = "your\_wifi\_password";

const int stepsPerRevolution = 2048; // Adjust this value based on your stepper motor

Stepper myStepper(stepsPerRevolution, 16, 17, 18, 19);

const int motorSpeed = 5; // Adjust this value based on your motor's speed requirements

const int waterPumpPin = 23;

const int ultrasonicTriggerPin = 24;

const int ultrasonicEchoPin = 25;

const int ledPin = 26;

NewPing sonar(ultrasonicTriggerPin, ultrasonicEchoPin, 200); // Adjust the max distance (200cm) as needed

const int morningHour = 7; // Set the hour for the morning activation

const int eveningHour = 19; // Set the hour for the evening activation

// Pushbullet API credentials

const char\* pushbulletToken = "your\_pushbullet\_api\_token";

Pushbullet pushbullet(pushbulletToken);

void setup() {

Serial.begin(115200);

// Initialize components

pinMode(waterPumpPin, OUTPUT);

pinMode(ledPin, OUTPUT);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

// Initialize Time library

configTime(0, 0, "pool.ntp.org", "time.nist.gov"); // Configure NTP servers

}

void loop() {

// Update time

struct tm timeinfo;

if(!getLocalTime(&timeinfo)){

Serial.println("Failed to obtain time");

return;

}

int currentHour = timeinfo.tm\_hour;

if ((currentHour == morningHour || currentHour == eveningHour) && timeinfo.tm\_min == 0 && timeinfo.tm\_sec == 0) {

// It's the specified activation hour and minute

// Check water level

int distance = sonar.ping\_cm();

Serial.print("Distance: ");

Serial.print(distance);

Serial.println(" cm");

if (distance < 10) {

// Water level is low, turn on the LED and water pump

digitalWrite(ledPin, HIGH);

digitalWrite(waterPumpPin, HIGH);

// Dispense water using the stepper motor

myStepper.setSpeed(motorSpeed);

myStepper.step(stepsPerRevolution);

// Wait for dispensing to complete

delay(5000); // Adjust as needed

// Turn off water pump and LED

digitalWrite(waterPumpPin, LOW);

digitalWrite(ledPin, LOW);

// Send notification to user about low water level

pushbullet.sendNote("Water Level Alert", "The water level is low. Please refill the fountain.");

}

}

// Indicate water level using the LED (for example, turn on if level is below 10 cm)

int distance = sonar.ping\_cm();

if (distance < 10) {

digitalWrite(ledPin, HIGH);

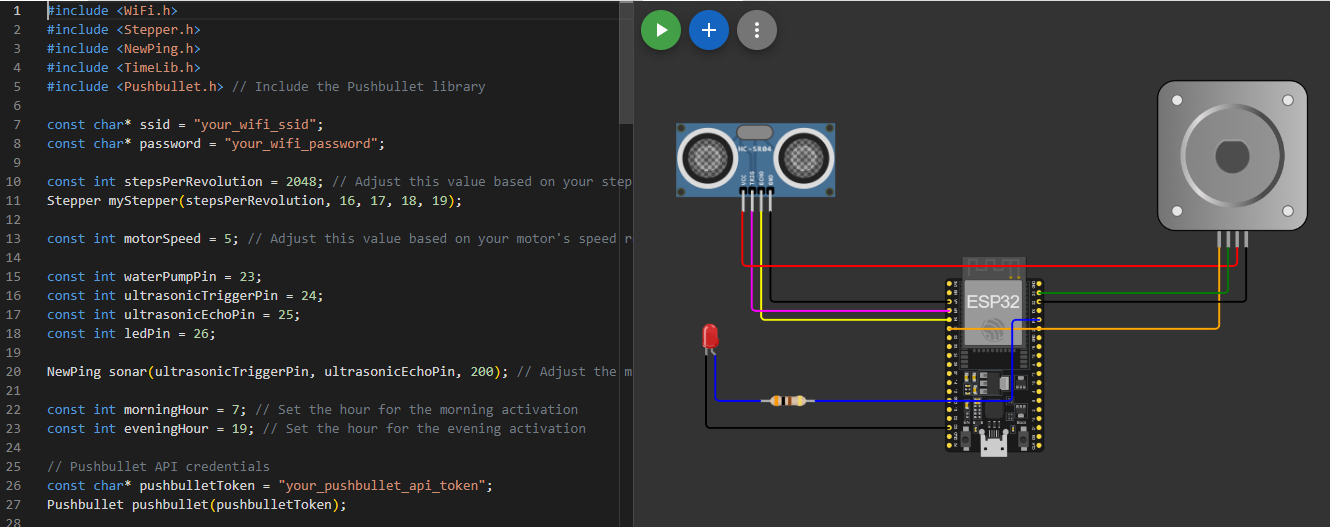
} else {

digitalWrite(ledPin, LOW);

}

delay(1000); // Adjust delay as needed for your application

}



After checking the water level, if it's below 10 cm, the LED is turned on, the water pump and stepper motor are activated to dispense water, and a message is constructed with the water level data.

The Pushbullet library is used to send a notification to the user with the water level information.

A message is created with the current water level, and if the notification is sent successfully, it prints "Notification sent successfully" to the Serial Monitor. If there is a failure, it prints "Failed to send notification."