



# ISA PROJECT

# LOGGUARD

Submitted By:

Prathesh Srinivas  
Malavika Dilu  
Sree Harshitha Kolla  
Ashish Arun Sawant

21ECB0F22  
21ECB0F23  
21ECB0F24  
21ECB0F25

# INTRODUCTION

The project addresses a major problem faced during the usage of public washrooms – the lack of discipline in people to flush after they have used it. Through this project, we have found an innovative, but effective way to inculcate the habit of pressing the flush button once the person has used the toilet.

Our product is a simple smart lock that can be locked manually but will open only on the press of a button – the flush button.

The lock is powered using an Arduino Uno board. This is very convenient for small-scale implementations, but on a larger scale, a central processor would be used to which all the washrooms on 1 floor are connected. The system is easy to install, so it can be added to already existing buildings/washrooms. Further improvements can be done by adding some extra features which are discussed later on in the report.

The prototype implemented by us is a functional model of the product explained above. It additionally closes the door within a specific amount of time to prevent any animals or birds entering an open washroom stall. The door will not be locked in this scenario. This feature further helps improve the hygiene of the toilets.

## ARDUINO CODE:

```
#include <Servo.h>
Servo ServoMotor;
int buttonPin = 2; // Define the pin for the push button
int buttonState = 0; // Variable to store the state of the button
int lastButtonState = 0; // Variable to store the previous state of the button
int RedpinLock = 12;
int GreenpinUnlock = 13;

void setup() {
  ServoMotor.attach(11);
  pinMode(buttonPin, INPUT_PULLUP); // Set the button pin as input with internal pull-up
  resistor
  pinMode(RedpinLock, OUTPUT);
  pinMode(GreenpinUnlock, OUTPUT);
  LockedPosition(true); // Lock the door initially
}
```

```

void loop() {
    buttonState = digitalRead(buttonPin); // Read the state of the button

    // Check if the button state has changed
    if (buttonState != lastButtonState) {
        // If button is pressed
        if (buttonState == LOW) {
            // Toggle the lock state
            LockedPosition(!isLocked());
            delay(200); // Debounce delay
        }
    }

    lastButtonState = buttonState; // Save the current state for the next iteration
}

void LockedPosition(bool locked) {
    if (locked) {

        digitalWrite(RedpinLock, HIGH); // Red LED ON
        digitalWrite(GreenpinUnlock, LOW); // Green LED OFF
        ServoMotor.write(0); // Lock position
        delay(2000);
        ServoMotor.write(90); // Unlock position
    } else {
        digitalWrite(RedpinLock, LOW); // Red LED OFF
        digitalWrite(GreenpinUnlock, HIGH); // Green LED ON
        ServoMotor.write(90); // Unlock position
    }
}

bool isLocked() {
    // Determine the current state of the lock based on servo position
    int servoPosition = ServoMotor.read();
    return (servoPosition == 0); // Assuming 0 degree position indicates locked
}

```

## Line by Line Explanation of the code:

```

```cpp
#include <Servo.h>
```

```

This line includes the Servo library, which is necessary to control servo motors in the Arduino code file.

```
```cpp
Servo ServoMotor;
```
```

It is used to declare a Servo object named `ServoMotor`, which will be used to control the servo motor.

```
```cpp
int buttonPin = 2; // Define the pin for the push button
```
```

It is used to declare an integer variable named `buttonPin` and assign it the value 2, representing the pin connected to the push button.

```
```cpp
int buttonState = 0; // Variable to store the state of the button
```
```

It is used to declare an integer variable named `buttonState` and initialize it to 0. This variable will be used to store the current state of the push button (pressed or not pressed).

```
```cpp
int lastButtonState = 0; // Variable to store the previous state of the button
```
```

It is used to declare an integer variable named `lastButtonState` and initialize it to 0. This variable will be used to store the previous state of the push button, enabling the detection of state changes.

```
```cpp
int RedpinLock = 12;
int GreenpinUnlock = 13;
```
```

These lines are used to declare integer variables `RedpinLock` and `GreenpinUnlock` and assign them the values 12 and 13, respectively. These values represent the digital pins on the board connected to the red LED (for indicating the locked state) and the green LED (for indicating the unlocked state).

```
```cpp
void setup() {
```
```

It is used to mark the beginning of the `setup()` function, which is called once when the Arduino board starts.

```
```cpp
```

```
ServoMotor.attach(11);  
...
```

It is used to attach the servo motor connected to pin 11 to the `ServoMotor` object, enabling control of the servo motor using this object.

```
...cpp  
pinMode(buttonPin, INPUT_PULLUP); // Set the button pin as input with internal pull-up  
resistor  
...
```

It is used to configure the pin defined by `buttonPin` (pin 2) as an input pin with an internal pull-up resistor enabled. This setup is used for reading the state of the push button.

```
...cpp  
pinMode(RedpinLock, OUTPUT);  
pinMode(GreenpinUnlock, OUTPUT);  
...
```

It is used to configure the pins connected to the red and green LEDs as output pins, allowing the Arduino to control them.

```
...cpp  
LockedPosition(true); // Lock the door initially  
...
```

It is used to call the `LockedPosition` function with the argument `true`, which locks the door initially. This function is responsible for setting the initial state of the lock and controlling the LEDs and servo motor accordingly.

```
...cpp  
void loop() {  
...
```

It is used to mark the beginning of the `loop()` function, which runs continuously after the `setup()` function has completed.

```
...cpp  
buttonState = digitalRead(buttonPin); // Read the state of the button  
...
```

It is used to read the current state of the push button connected to the pin defined by `buttonPin` and stores it in the `buttonState` variable.

```
...cpp  
if (buttonState != lastButtonState) {  
...
```

It is used to check if the current state of the push button is different from the previous state, indicating a state change.

```
```cpp
    if (buttonState == LOW) {
...

```

It is used to check if the push button is pressed, as `LOW` indicates a pressed state due to the pull-up resistor configuration.

```
```cpp
    LockedPosition(!isLocked());
...

```

It is used to call the `LockedPosition` function with the argument `!isLocked()`, which toggles the lock state based on the current lock state.

```
```cpp
    delay(200); // Debounce delay
...

```

It is used to add a brief delay to debounce the push button, preventing rapid toggling of the lock state due to mechanical bouncing.

```
```cpp
    lastButtonState = buttonState; // Save the current state for the next iteration
...

```

It is used to update the `lastButtonState` variable with the current state of the push button for comparison in the next iteration of the loop.

```
```cpp
void LockedPosition(bool locked) {
...

```

This line marks the beginning of the `LockedPosition` function, which is responsible for controlling the lock state, LEDs, and servo motor based on the `locked` parameter.

```
```cpp
    if (locked) {
...

```

It is used to check if the `locked` parameter is `true`, indicating that the door should be locked.

```
```cpp
    digitalWrite(RedpinLock, HIGH); // Red LED ON
    digitalWrite(GreenpinUnlock, LOW); // Green LED OFF
...

```

These lines turn on the red LED (indicating the locked state) and turn off the green LED (indicating the unlocked state).

```
```cpp
    ServoMotor.write(0); // Lock position

```

```

    delay(2000);
    ServoMotor.write(90); // Unlock position
...

```

It is used to move the servo motor to the locked position (0 degrees) and then to the unlocked position (90 degrees) to simulate locking and unlocking the door.

```

...cpp
    } else {
...

```

It marks the beginning of the `else` block, which executes if the `locked` parameter is `false`, indicating that the door should be unlocked.

```

...cpp
    digitalWrite(RedpinLock, LOW); // Red LED OFF
    digitalWrite(GreenpinUnlock, HIGH); // Green LED ON
...

```

It is used to turn off the red LED and turn on the green LED to indicate the unlocked state.

```

...cpp
    ServoMotor.write(90); // Unlock position
...

```

It is used to move the servo motor to the unlocked position (90 degrees).

```

...cpp
    }
}
...

```

It marks the end of the `LockedPosition` function.

```

...cpp
bool isLocked() {
...

```

It marks the beginning of the `isLocked` function, which checks if the door is locked based on the position of the servo motor.

```

...cpp
    int servoPosition = ServoMotor.read();
...

```

It is used to read the current position of the servo motor and stores it in the `servoPosition` variable.

```

...cpp
    return (servoPosition == 0); // Assuming 0 degree position indicates locked
...

```

It returns `true` if the `servoPosition` is 0 (indicating the locked position) and `false` otherwise, assuming that 0 degrees represents the locked position of the servo motor.

```
```cpp  
}  
```
```

It marks the end of the `isLocked` function.

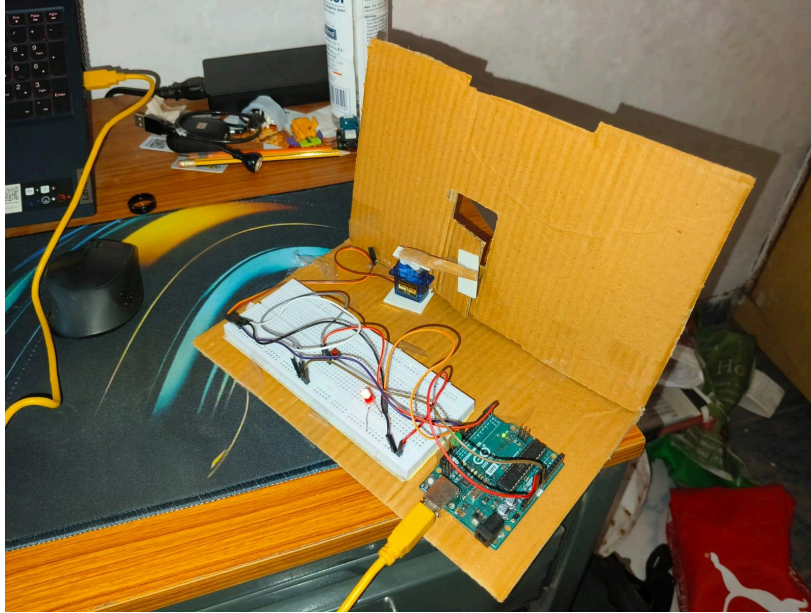
## PHOTO AND VIDEO OF THE PROTOTYPE:

Case 1: The door is open when the switch(flush) is pressed and the LED turns green.




Case 2: When the person leaves the washroom, the door is closed after a fixed time interval(here we gave 2 seconds to explain the prototype and it can be extended) and the light turns red once again.





Video of the Working Prototype(Link):

 [ISA\\_CRSquad\\_Project\\_video.mp4](#)

## FUTURE SCOPE

The product, Looguard, once implemented, has a wide range of future applications and benefits.

### 1. Integration with IoT:

- Remote Monitoring and Management: By integrating with IoT, facility managers can remotely monitor the status of each bathroom, receiving real-time updates on usage, flush counts, and door operations. This can streamline maintenance operations and ensure that restrooms are always functioning properly.
- Predictive Maintenance: IoT sensors can predict when a component is likely to fail or need servicing, enabling proactive maintenance before issues arise, thus reducing downtime and repair costs.

### 2. Advanced Hygiene Features:

- Automatic Sanitization: Incorporating automatic sanitization sprays that activate after each use can ensure a consistently clean environment, reducing the spread of germs and maintaining high hygiene standards.
- Air Fresheners: Integrating automatic air fresheners that release a pleasant scent after each use can improve the overall user experience by ensuring the bathroom always smells fresh.

- Real-Time Cleanliness Monitoring: Sensors can detect the cleanliness level of the restroom, alerting cleaning staff when attention is needed, thus maintaining a pristine environment at all times.

### 3. Smart Display:

- User Alerts and Feedback: A smart display outside the washroom can allow users to report maintenance issues or cleanliness concerns directly to facility managers. This feedback loop ensures that problems are addressed promptly.
- Usage Statistics and Notifications: The smart display can show real-time occupancy, estimated wait times, and notifications about maintenance activities, enhancing user experience and satisfaction.

### 4. Enhanced Security Features:

- User Authentication: Implementing user authentication through RFID, QR codes, or biometric systems can ensure that only authorized users access certain restrooms, enhancing security in sensitive areas such as employee-only facilities.
- Incident Reporting: Integrated security cameras and sensors can detect and report vandalism or other incidents in real-time, ensuring a safer and more secure environment.

### 5. Environmental Impact Monitoring:

- Carbon Footprint Analysis: Monitoring and analyzing the carbon footprint of restroom usage can help in developing strategies to reduce environmental impact, such as optimizing energy use for lighting and ventilation.
- Sustainable Materials: Using data to evaluate the effectiveness of sustainable cleaning materials and practices can lead to more eco-friendly operations.

### 6. User Experience Enhancements:

- Personalization: The system can be designed to recognize regular users and personalize settings such as preferred air freshener scents or automatic seat adjustments, enhancing user comfort.
- Voice Assistance: Integrating voice control features for flushing and reporting issues can provide a touchless experience, further improving hygiene and convenience.

### 7. Integration with Building Management Systems (BMS):

- Centralized Control: Looguard can be integrated with the overall building management system to allow centralized control and monitoring of all restroom facilities, contributing to more efficient building operations.
- Energy Efficiency: By integrating with BMS, restroom lighting and ventilation can be optimized based on occupancy data, reducing energy consumption and operational costs.

By exploring these future scopes, Looguard can significantly enhance restroom management, hygiene, user satisfaction, and environmental sustainability, making it a versatile and innovative solution for a variety of settings.

## CONCLUSION

In conclusion, our project tackles the problem of improper and unhygienic flushing habits in public washrooms. The smart lock system encourages users to develop better habits by integrating the automatic lock with the flush mechanism, ensuring the door unlocks only after flushing. This promotes personal hygiene effectively. Using Arduino UNO, we demonstrate this concept on a small scale, showcasing its feasibility and potential impact.