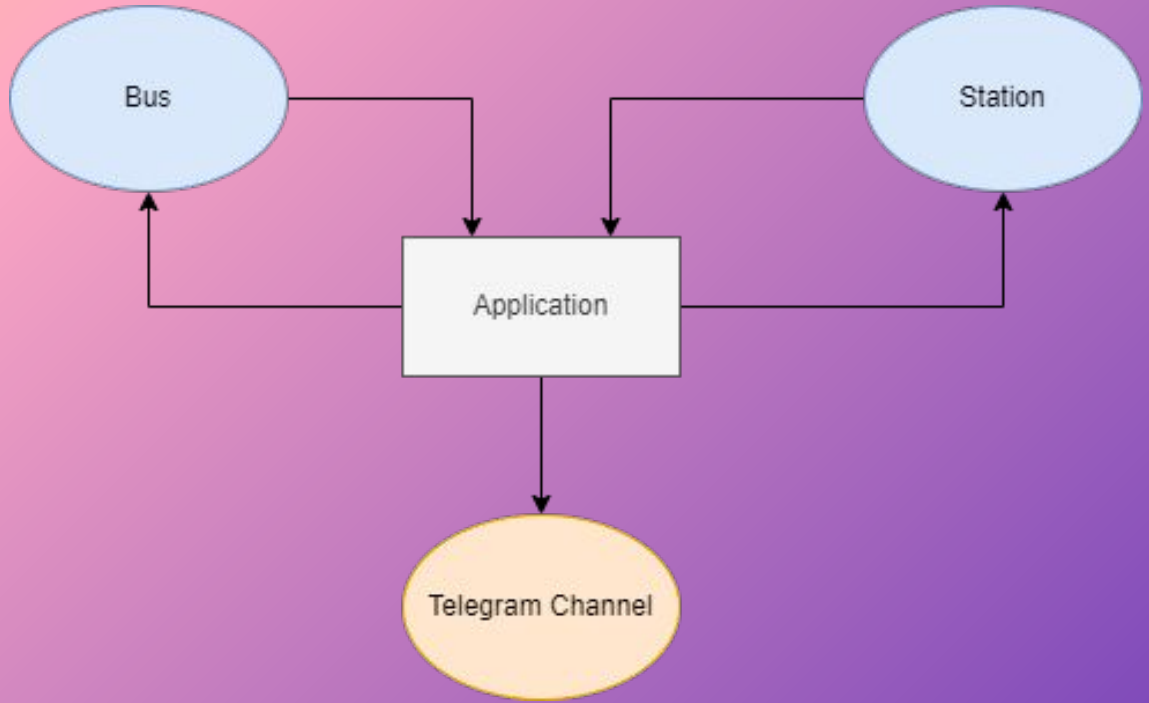
Smart Bus

IoT Project by Gianluca Ronga & Guido Immediata

THE IDEA

The project aims to develop a prototype of a smart bus environment where informations gained by several sensors can be used to improve the quality of the bus service system. The environment is composed of a bus, a station, the application server a telegram channel and a black line path.

THE ARCHITECTURE



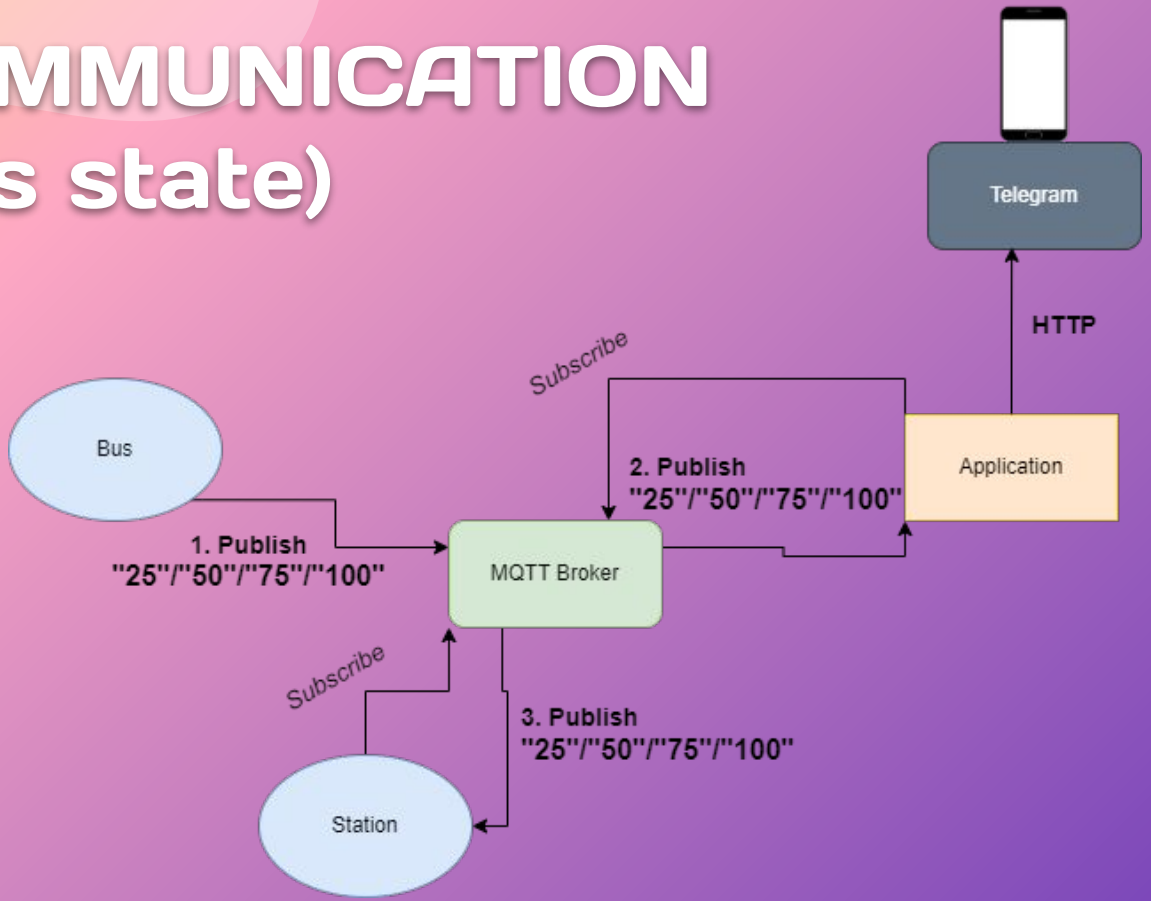
COMMUNICATION


```
// MQTT Broker Configuration:
const char *mqtt_broker = "broker.emqx.io";
const int mqtt_port = 1883;
```

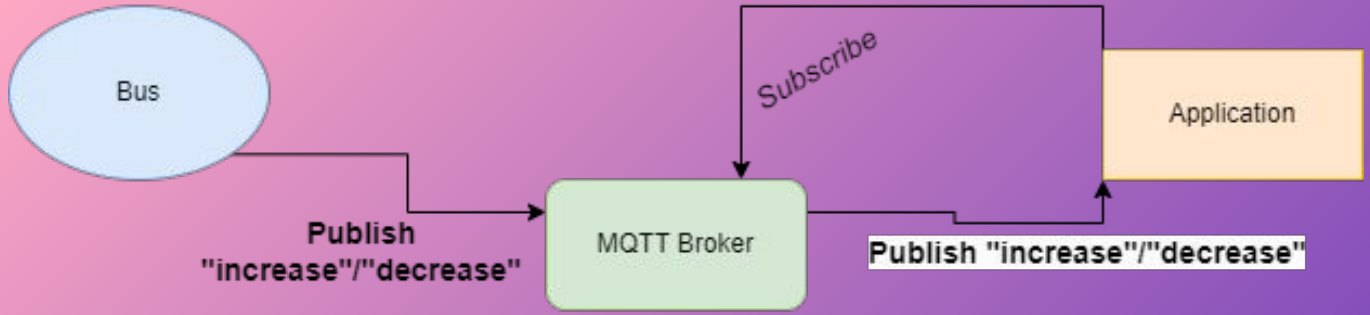
```
// Connessione MQTT
client.setServer(mqtt_broker, mqtt_port);
client.subscribe(start_topic);
client.setCallback(callback);
```

```
client.publish(pos_topic, "25");
```

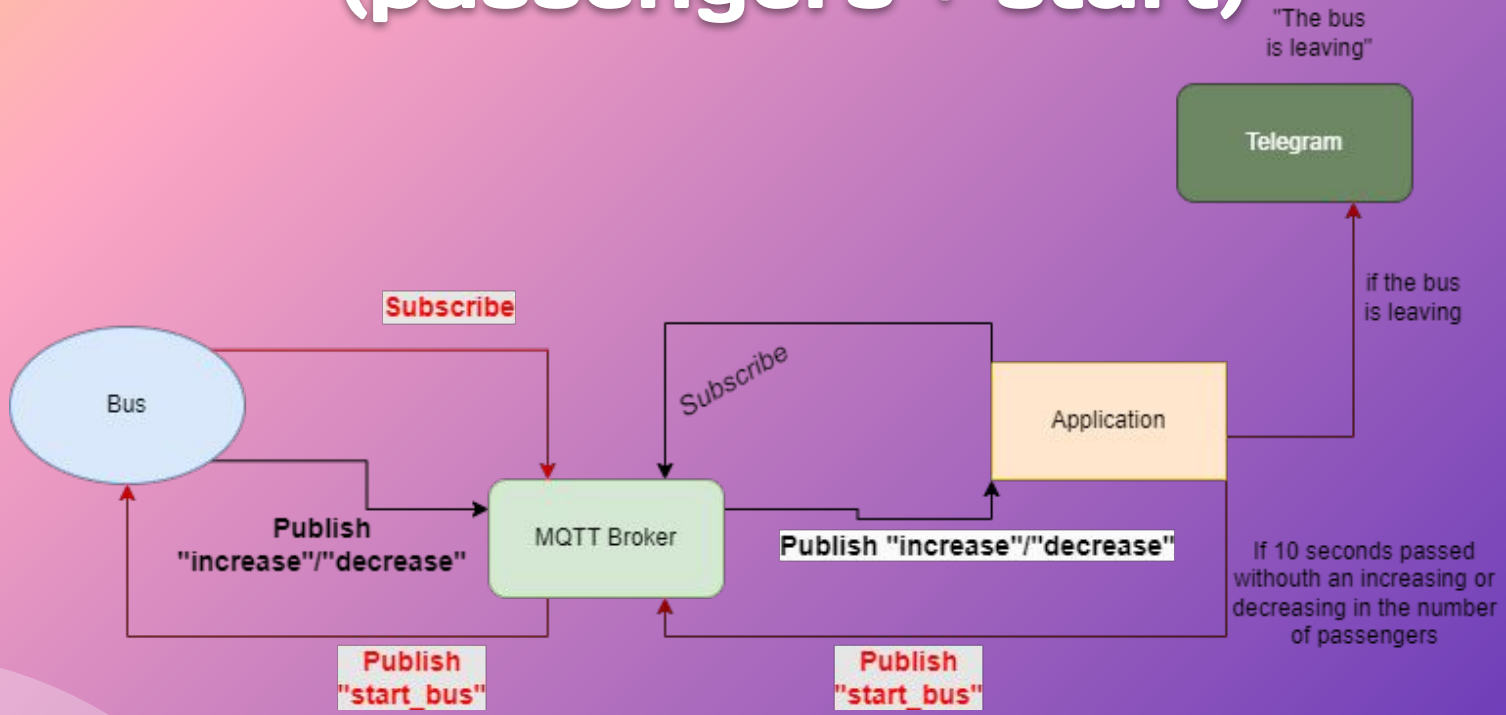
COMMUNICATION (bus state)



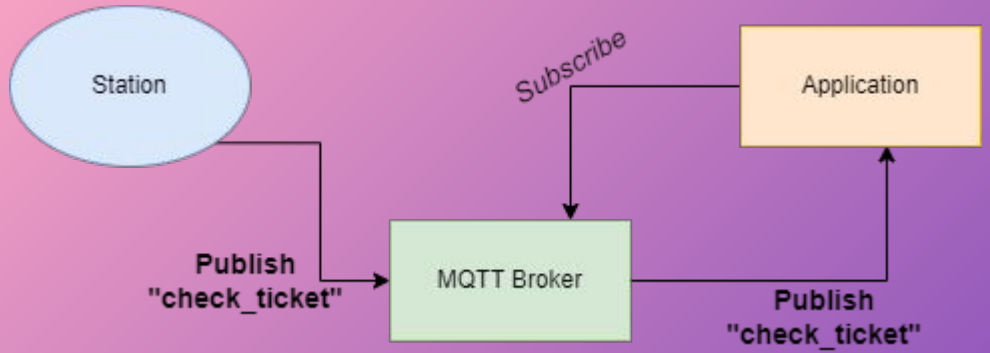
COMMUNICATION (passengers)



COMMUNICATION (passengers + start)



COMMUNICATION (num_ticket)



02

SENSORS

What is the purpose of every sensor in the project

PASSENGER MANAGEMENT

Two ultrasonic sensors were utilized to track passengers entering and exiting the Smart Bus, with the data managed through an integer variable named passengerCount.

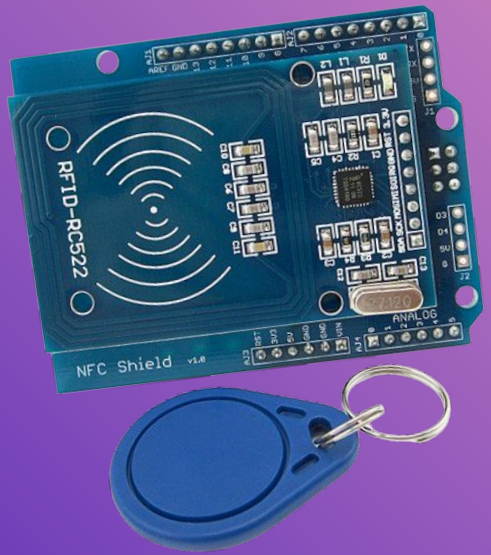
Several checks were implemented to simulate a real-life scenario effectively:

1) A boolean was used to block communication while the Smart Bus is in motion.
2) A cooldown period was added to the sensors between detections to prevent the counter from increasing too rapidly.
3) The passengerCount variable was ensured to never hold a negative value.

Ultrasonic Sensor Code

```
int calculateDistanceSalita() {
  digitalWrite(trigPin1, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin1, LOW);

  duration = pulseIn(echoPin1, HIGH);
  return duration * 0.034 / 2;
}

int calculateDistanceDiscesa() {
  digitalWrite(trigPin2, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin2, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin2, LOW);

  duration = pulseIn(echoPin2, HIGH);
  return duration * 0.034 / 2;
}
```

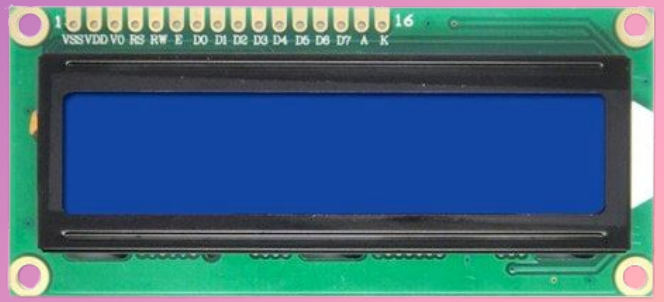
TICKET MANAGEMENT



A RFID sensor was implemented to simulate the ticketing system of the Smart Bus. This system required fewer controls compared to passenger management, as users can validate their tickets while the bus is in motion, and a ticket validation can’t result in a decrement.

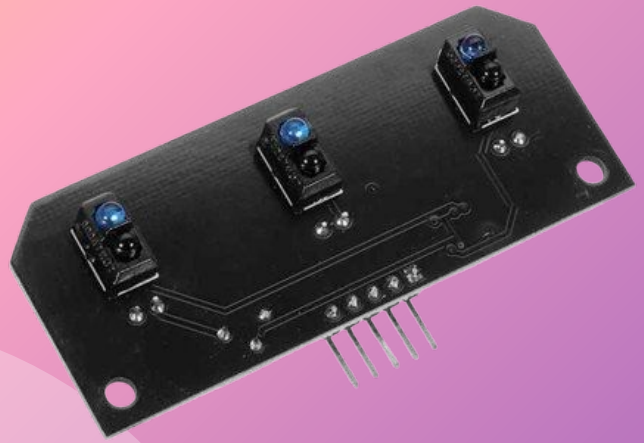
BUS STOP INFOBOX

When a user arrives at the bus stop, they can check the location of the Smart Bus on an LCD display, which shows a progress bar indicating the bus's position along its route. The progress bar is updated by subscribing to the “arrivobus” topic, which provides information about the current position of the Smart Bus.



LCD Display Code

```
void updateProgressBar(unsigned long count, unsigned long totalCount, int lineToPrintOn){
    double factor = totalCount/80.0;
    int percent = (count+1)/factor;
    int number = percent/4;
    int remainder = percent%4;
    if(number > 0)
    {
      lcd.setCursor(number-1,lineToPrintOn);
      lcd.write(5);
    }

      lcd.setCursor(number,lineToPrintOn);
      lcd.write(remainder);

}

void clearRow(byte row){
    lcd.setCursor(0,row);
    lcd.print("                    ");
}
```



INFRARED SENSORS

The Smart Bus was designed with automation in mind, incorporating a line-following mechanism to navigate its route.

The infrared sensors provide four possible outputs:

1) If the left sensor is activated the bus turns left.

2) If the right sensor is activated the bus turns right.

3) If neither the left nor the right sensor is activated, the bus moves forward.

4) If both sensors are activated simultaneously, the bus publishes an MQTT message to the posbus topic. After repeating this four times, it stops and waits for a message on the startbus topic to resume operation.

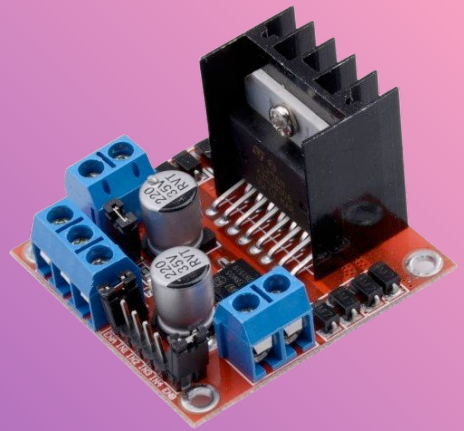
Code

```
//If none of the sensors detects black line, then go straight
if (rightIRSensorValue == LOW && leftIRSensorValue == LOW)
{

  Serial.println("Avanti");
  moveForward(vspeed);
}
//If right sensor detects black line, then turn left
else if (rightIRSensorValue == HIGH && leftIRSensorValue == LOW )
{
    Serial.println("Destra");
    turnLeft(tspeed);
}
//If left sensor detects black line, then turn left
else if (rightIRSensorValue == LOW && leftIRSensorValue == HIGH )
{
    Serial.println("Sinistra");
    turnRight(tspeed);
}
```

```
else
{
  percentuale += 1;
  if(percentuale == 1){
    client.publish(pos_topic, "25");
    delay(70);
  }
  if(percentuale == 2){
    client.publish(pos_topic, "50");
    delay(80);
  }
  if(percentuale == 3){
    client.publish(pos_topic, "75");
    delay(80);
  }
  if(percentuale == 4){
    client.publish(pos_topic, "100");
    Serial.println("STOP");
    pullmanArrived = 1;
    client.publish(partenza_topic, "start");
    stopMotors();
    delay(20);
    percentuale = 0;
  }
```

MOTOR DRIVER AND MOTORS

The Smart Bus was designed with a structure that includes four driving wheels. To control their rotation and speed effectively, a motor driver was required.



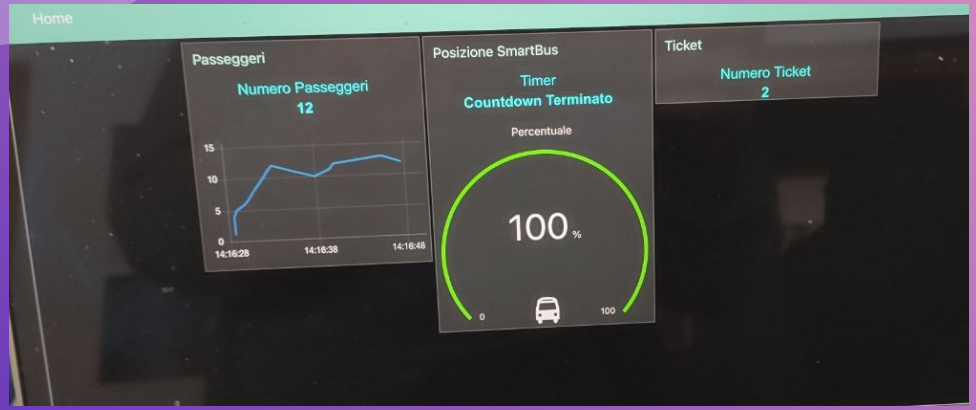
All the 4 motors were connected to a L298N motor driver and were managed by the following code:

```
void moveForward(int speed) {
    analogWrite(enableRightMotor, speed);
    analogWrite(enableLeftMotor, speed);

    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
    digitalWrite(leftMotorPin1, HIGH);
    digitalWrite(leftMotorPin2, LOW);
}

void turnLeft(int speed) {
    analogWrite(enableRightMotor, speed);
    analogWrite(enableLeftMotor, speed / 3);

    digitalWrite(rightMotorPin1,HIGH);
    digitalWrite(rightMotorPin2,LOW);
    digitalWrite(leftMotorPin1,HIGH);
    digitalWrite(leftMotorPin2,LOW);
}
```

```
void turnRight(int speed) {
    analogWrite(enableRightMotor, speed / 3);
    analogWrite(enableLeftMotor, speed);

    digitalWrite(rightMotorPin1,LOW);
    digitalWrite(rightMotorPin2,HIGH);
    digitalWrite(leftMotorPin1,HIGH);
    digitalWrite(leftMotorPin2,LOW);

}

void stopMotors() {
    analogWrite(enableRightMotor, 0);
    analogWrite(enableLeftMotor, 0);

    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, LOW);
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, LOW);
}
```

03
WEB PAGE
&
TELEGRAM

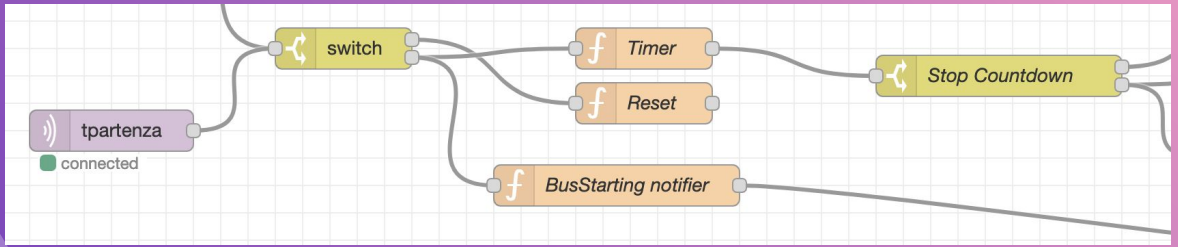
How a user interacts with the system



User Visualization

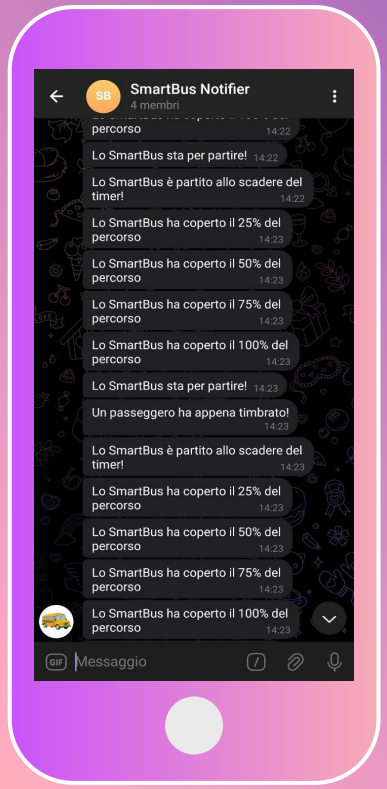
On the web page, users can easily view key system information, including the total number of tickets, the current number of passengers, a chart showing passenger trends, the Smart Bus current location on its route, and the remaining time until departure.

The only aspect not prominently displayed is that the remaining time before departure resets whenever a passenger boards or exits the bus: a reflection of the kindness of our considerate bus driver.



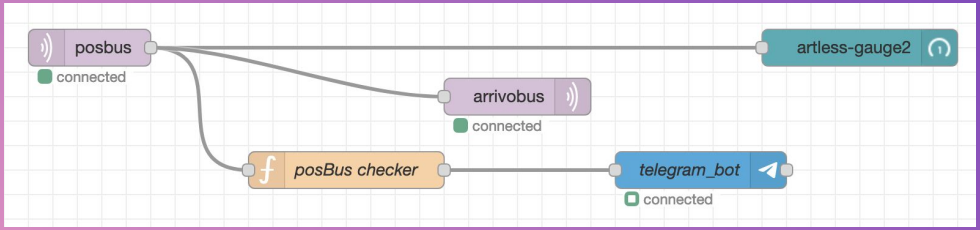
JS Code

```
let timer = flow.get('timer') || null;
let countdown = flow.get('countdown') || 0;


if (msg.payload === "start" && !timer) {
    countdown = 10;
    flow.set('countdown', countdown);


    timer = setInterval(() => {
        countdown = flow.get('countdown') || 0;
        countdown -= 1;

        if (countdown > 0) {
            flow.set('countdown', countdown);
            node.send({ payload: countdown });
        } else {
            clearInterval(timer);
            flow.set('timer', null);
            flow.set('countdown', 0);
            node.send({ payload: countdown });
            node.send({ payload: "Countdown Terminato" });
        }
    }, 1000);

    flow.set('timer', timer);
}

return null;
```



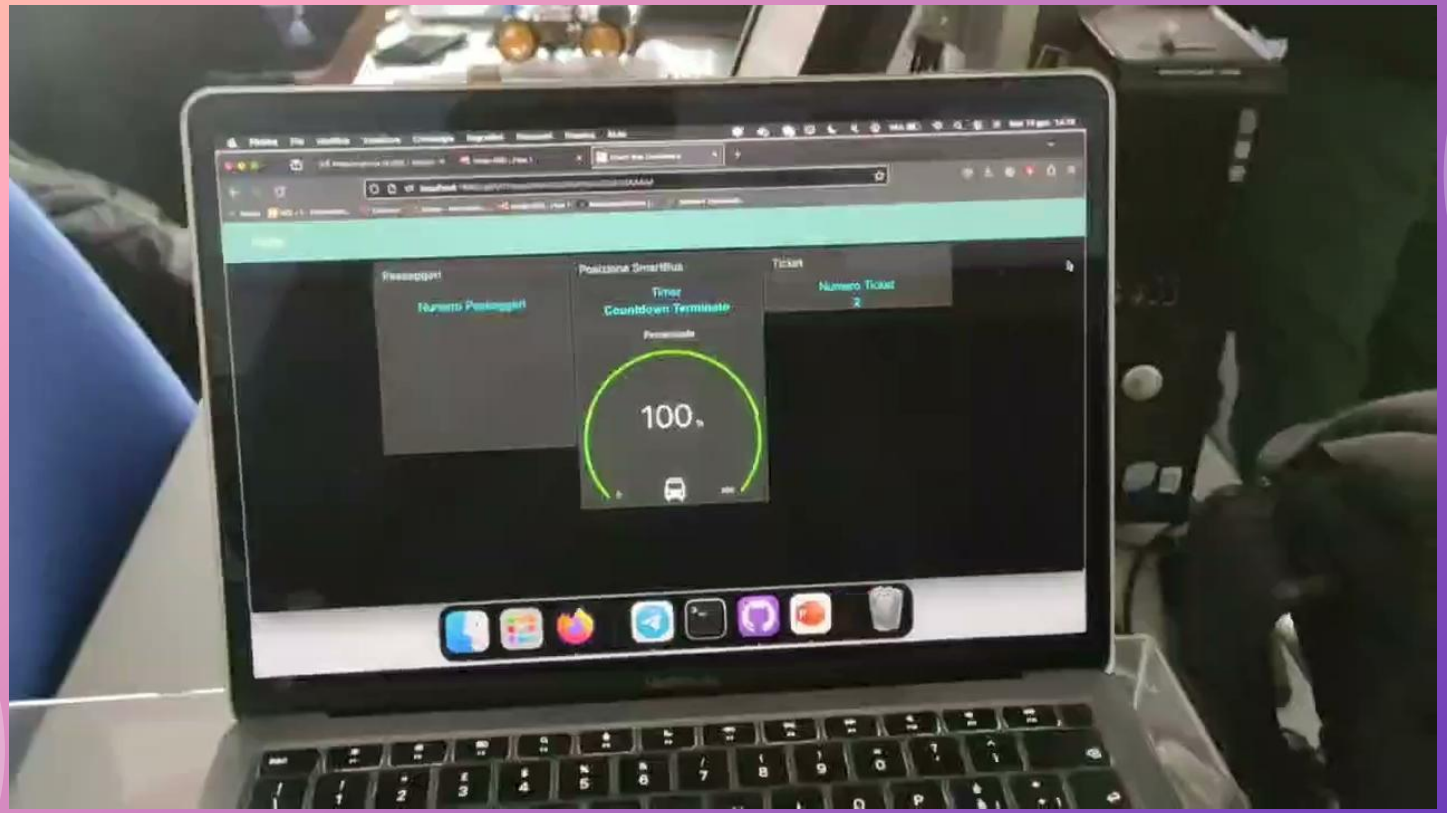
Telegram Bot

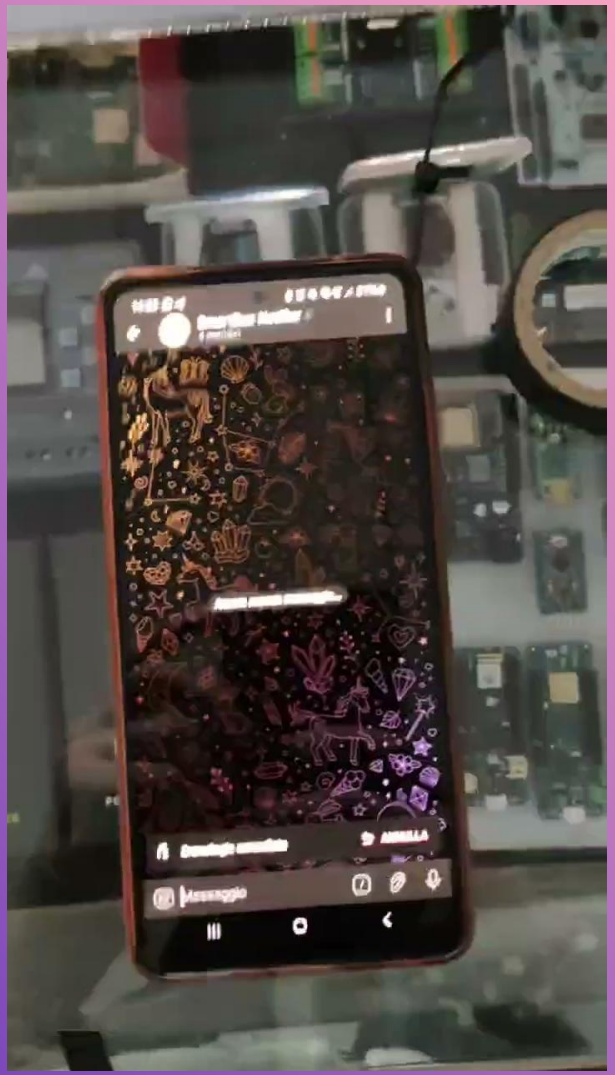
A Telegram bot named SmartBus was created to assist users by providing notifications about ticket validation, the bus's route status and departure times.

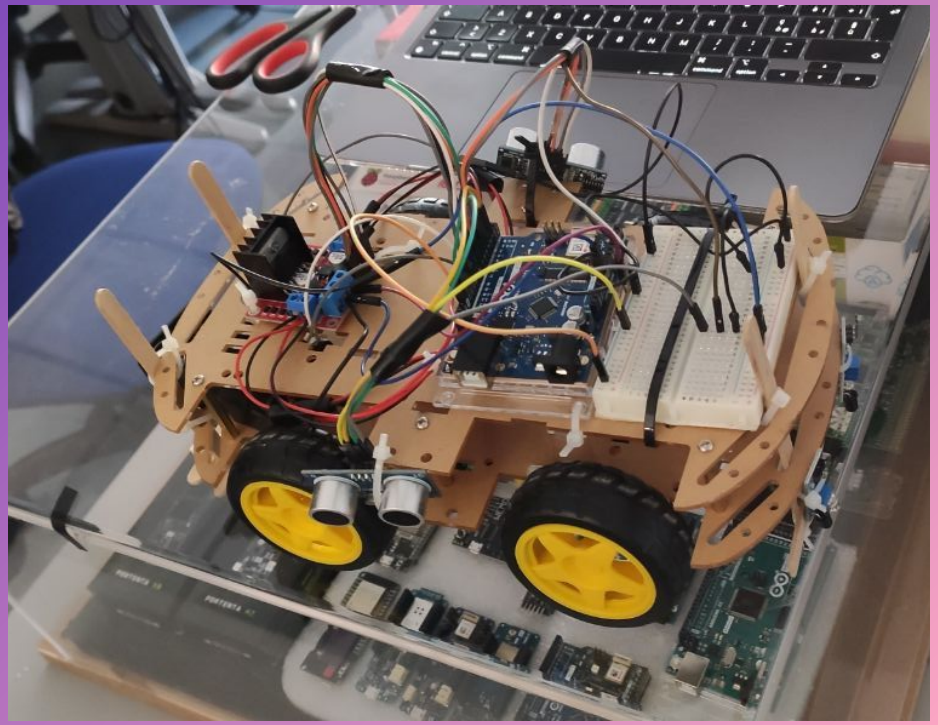


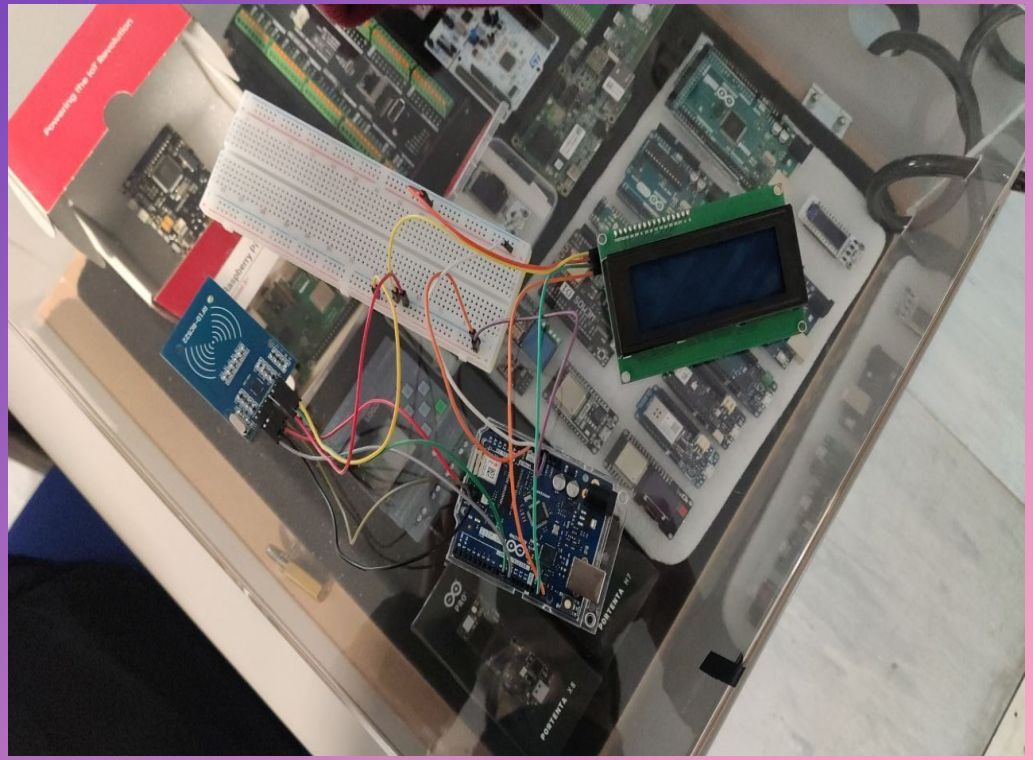
Communication with Telegram is handled through Node-RED nodes, using a predefined bot and channel ID along with a dynamic payload to generate the message content.

04 PROJECT VIDEO(s)


Passeggeri
Numero Passeggeri
Posizione Smartbus
Timer
Countdown Terminato
100
Ticket
Numero Ticket
2






Powering the IoT Revolution

THANKS!

For your attention

https://github.com/GameWame/SmartBus