UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

Trabajo Práctico 2 Informe

Sistemas Embebidos

LCC 2025
Lucas Moyano 13446

2. Implementar una aplicación que realice las siguientes tareas usando FreeRTOS:

a) Lea constantemente el valor de la intensidad luminosa.

```
#include <Arduino_FreeRTOS.h>
// Include semaphore supoport
#include <semphr.h>

/*
 * Declaring a global variable of type SemaphoreHandle_t
 *
 */
SemaphoreHandle_t interruptSemaphore;

int sensorValue = -1;
bool isReadOn = true;

// define two tasks for Blink & AnalogRead
void TaskAnalogRead( void *pvParameters );


// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on
LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }


  xTaskCreate(
    TaskPrintAnalogRead,
    "PrintAnalogRead",
    110,
    NULL,
    2,
    NULL
  );
  // Now the task scheduler, which takes over control of scheduling
individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}


/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

// Prints analog read output as a string with a certain format to be
read by backend
void TaskPrintAnalogRead(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    if (isReadOn) {
      // read the input on analog pin 3:
      sensorValue = analogRead(A3);

      char luminosityBuffer[5];
      sprintf(luminosityBuffer, "%04d", sensorValue);
      Serial.println("luminosity: " + String(luminosityBuffer));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three second
    } else {
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // for some reason this
makes isReadOn update
    }
    Serial.println("isReadOn: " + String(isReadOn));
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (analogRead): ");
    Serial.println(stackRemaining);*/
  }
}
```

b) Cada 3 segundos, envíe a través del puerto serial el último valor leído (Nota: la escritura en el monitor serial demora cierto tiempo. No debe ser interrumpida por otra tarea). Muestre el valor leído en una página web y en la aplicación Python.

Codigo Arduino:

```
#include <Arduino_FreeRTOS.h>

int sensorValue = -1;
bool isReadOn = true;

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on
LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  84  // This stack size can be checked & adjusted by reading the
Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the
highest, and 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskPrintAnalogRead,
    "PrintAnalogRead",
    110,
    NULL,
    2,
    NULL
  );

    // Now the task scheduler, which takes over control of scheduling
individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

/*
  Blink
  Turns on LED 11 on for one second, then off for one second,
repeatedly.
  Only if read is on.
*/
void TaskBlink(void *pvParameters)
{
  (void) pvParameters;


  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(11, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    if (isReadOn) {
      digitalWrite(11, HIGH);   // turn the LED on (HIGH is the voltage
level)
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(11, LOW);    // turn the LED off by making the
voltage LOW
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }
    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (blink): ");
    Serial.println(stackRemaining);*/
  }
}


// Prints analog read output as a string with a certain format to be read by backend
void TaskPrintAnalogRead(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    if (isReadOn) {
      // read the input on analog pin 3:
      sensorValue = analogRead(A3);

      char luminosityBuffer[5];
      sprintf(luminosityBuffer, "%04d", sensorValue);
      Serial.println("luminosity: " + String(luminosityBuffer));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three second
    } else {
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // for some reason this makes isReadOn update
    }
    Serial.println("isReadOn: " + String(isReadOn));
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (analogRead): ");
    Serial.println(stackRemaining);*/
  }
}
```

Codigo Backend:

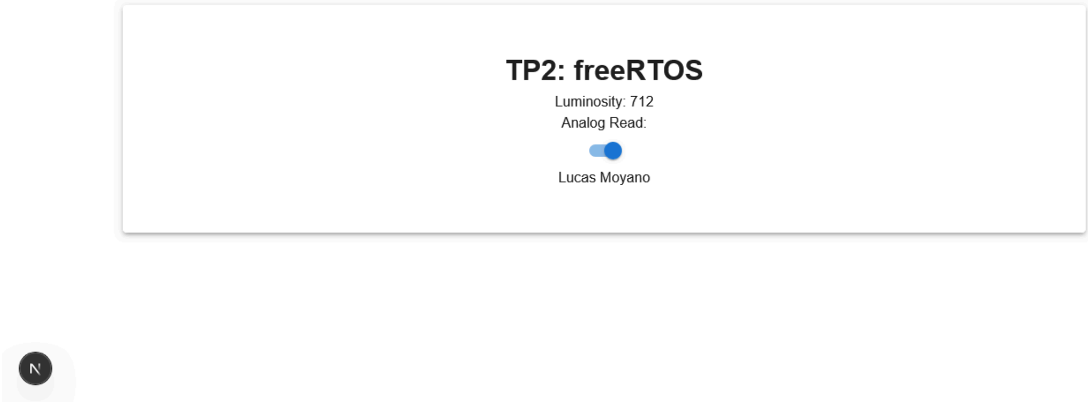
```
from flask import Flask, jsonify, request
from flask_cors import CORS
from flask_socketio import SocketIO
import serial
import threading
import time

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")
```

```
CORS(app, resources={r"/*": {"origins": "*"}},
supports_credentials=True)

ser = serial.Serial(port='/dev/ttyACM0',
                    baudrate=9600,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    timeout=1,
                    xonxoff=False,
                    rtscts=False,
                    dsrdtr=False,
                    inter_byte_timeout=None,
                    exclusive=None)

# Reads arduino serial port output, and sends it to frontend if string
matches case
def read_serial():
    while True:
        if ser.in_waiting > 0:
            try:
                line = ser.readline().decode('utf-8').rstrip()

                if line:
                    if line.startswith("luminosity:"):
                        # Sends luminosity value to frontend
                        # luminosity: 0000
                        # Extract the luminosity value from the line
                        ldrLuminosity = int(line[12:16])
                        print("LDR luminosity: ", ldrLuminosity)

                        socketio.emit("update_arduino_data",
{"ldr_luminosity": ldrLuminosity})
                    elif line.startswith("isReadOn:"):
                        # Sends analog read on/off value to frontend
                        # isReadOn: 0
                        analogReadOn = bool(int(line[10:11]))

                        print("Analog read on: ", analogReadOn)
                        socketio.emit("update_analog_read_on",
{"analog_read_on": analogReadOn})
                    else:
                        print("not a case: ", line)
            except Exception as e:
                print(f"Error reading serial data: {e}")
        time.sleep(1.0)

threading.Thread(target=read_serial, daemon=True).start()

@socketio.on('connect')
def handle_connect():
    print("Client connected")

if __name__ == "__main__":
    socketio.run(app, debug=True, port=8080, host="192.168.100.99")
```

Codigo Frontend:

```
"use client"
import Image from "next/image";
import styles from "./page.module.css";
import Paper from '@mui/material/Paper';
import {Switch } from "@mui/material";
import { useState, useEffect } from "react";
import io from "socket.io-client";

const backendUrl = "http://192.168.100.99:8080";

const socket = io(backendUrl);

export default function Home() {
  const [luminosity, setLuminosity] = useState<number>(0);
  const [analogReadOn, setAnalogReadOn] = useState<boolean>(true);

  useEffect(() => {
    // receives luminosity data from the backend and sets alarm on if >
800
    socket.on("update_arduino_data", (data) => {
      setLuminosity(data.ldr_luminosity);

    });

    // receives analog read on/off data from the backend and sets alarm
off if analog read is off
    socket.on("update_analog_read_on", (data) => {
      if (analogReadOn != data.analog_read_on) {
        setAnalogReadOn(data.analog_read_on);
      }

    });
  }, [socket, luminosity, analogReadOn]);


  return (
    <div className={styles.container}>
      <Paper elevation={3} className={styles.paper}>
        <div className={styles.innerContainer}>

          <h1>TP2: freeRTOS</h1>
          <p>Luminosity: {luminosity}</p>
          <p>Analog Read:</p>
          <Switch checked={analogReadOn}/>

        </div>
      </Paper>

    </div>
  );
}
```

Podemos ver la aplicación web en la [Figura 1].



TP2: freeRTOS

Luminosity: 712

Analog Read:

Lucas Moyano

[Figura 1] Aplicación web desarrollada

c) La lectura puede iniciarse y detenerse a través de los pulsadores de la placa Arduino y desde botones en una página web.

Codigo arduino:

```
#include <Arduino_FreeRTOS.h>
// Include semaphore supoport
#include <semphr.h>

/*
 * Declaring a global variable of type SemaphoreHandle_t
 *
 */
SemaphoreHandle_t interruptSemaphore;

int sensorValue = -1;
bool isReadOn = true;

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

void TaskPressedButtonChecker( void *pvParameters);

void TaskReceiveInstructions( void *pvParameters);

// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on
LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  84  // This stack size can be checked & adjusted by reading the
Stack Highwater
```

```
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the
highest, and 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskPrintAnalogRead,
    "PrintAnalogRead",
    110,
    NULL,
    2,
    NULL
  );

  xTaskCreate(
    TaskPressedButtonChecker,
    "PressedButtonChecker",
    128,
    NULL,
    2,
    NULL
  );


  xTaskCreate(
    TaskReceiveInstructions,
    "ReceiveInstructions",
    126, //20 + 18 + 44 + 44
    NULL,
    2,
    NULL
  );

  /**
   * Create a binary semaphore.
   * https://www.freertos.org/xSemaphoreCreateBinary.html
   */
  interruptSemaphore = xSemaphoreCreateBinary();
  if (interruptSemaphore != NULL) {
    // Attach interrupt for Arduino digital pin
    attachInterrupt(digitalPinToInterrupt(2), interruptHandler,
FALLING);
  }
  // Now the task scheduler, which takes over control of scheduling
individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

void interruptHandler() {
  /**
   * Give semaphore in the interrupt handler
   * https://www.freertos.org/a00124.html
   */

  xSemaphoreGiveFromISR(interruptSemaphore, NULL);
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

/*
  Blink
  Turns on LED 11 on for one second, then off for one second,
repeatedly.
  Only if read is on.
*/
void TaskBlink(void *pvParameters)
{
  (void) pvParameters;


  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(11, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    if (isReadOn) {
      digitalWrite(11, HIGH);   // turn the LED on (HIGH is the voltage
level)
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(11, LOW);    // turn the LED off by making the
voltage LOW
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (blink): ");
    Serial.println(stackRemaining);*/
  }
}

// Prints analog read output as a string with a certain format to be
read by backend
void TaskPrintAnalogRead(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    if (isReadOn) {
      // read the input on analog pin 3:
      sensorValue = analogRead(A3);

      char luminosityBuffer[5];
      sprintf(luminosityBuffer, "%04d", sensorValue);
      Serial.println("luminosity: " + String(luminosityBuffer));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three second
    } else {
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // for some reason this
makes isReadOn update
    }
    Serial.println("isReadOn: " + String(isReadOn));
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (analogRead): ");
    Serial.println(stackRemaining);*/
  }
}

// Task that activates on interruption (button 2 pressed), it turns off
read
void TaskPressedButtonChecker(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    /**
    * Take the semaphore.
    * https://www.freertos.org/a00122.html
    */
    if (xSemaphoreTake(interruptSemaphore, portMAX_DELAY) == pdPASS) {


      isReadOn = !isReadOn;
      //Serial.println("semaphore taken, isReadOn: " +
String(isReadOn));
      Serial.println("isReadOn: " + String(isReadOn));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three
seconds
    } else {
      vTaskDelay(10);
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (button pressed): ");
    Serial.println(stackRemaining);*/
  }
}

// Reads from serial and if it reads a t turns on read, if it reads an
f it turns it off
void TaskReceiveInstructions( void *pvParameters) {
  (void) pvParameters;

  char buffer;


  for (;;) {
    if (Serial.available()>0) {

      buffer = (char)Serial.read();
      Serial.print("Buffer: ");
      Serial.println(buffer);

      if (buffer == 'f') {
        isReadOn = false;
      } else if (buffer == 't') {
        isReadOn = true;
      }
    }
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (receiveInstructions): ");
    Serial.println(stackRemaining);*/
    vTaskDelay(10);
  }
}
```

Codigo backend:

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from flask_socketio import SocketIO
import serial
import threading
import time

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")
CORS(app, resources={r"/*": {"origins": "*"}},
supports_credentials=True)

ser = serial.Serial(port='/dev/ttyACM0',
                    baudrate=9600,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    timeout=1,
                    xonxoff=False,
                    rtscts=False,
                    dsrdtr=False,
                    inter_byte_timeout=None,
                    exclusive=None)

# Reads arduino serial port output, and sends it to frontend if string
matches case
def read_serial():
    while True:
        if ser.in_waiting > 0:
            try:
                line = ser.readline().decode('utf-8').rstrip()
                if line:
                    if line.startswith("luminosity:"):
                        # Sends luminosity value to frontend
                        # luminosity: 0000
                        # Extract the luminosity value from the line
                        ldrLuminosity = int(line[12:16])
                        print("LDR luminosity: ", ldrLuminosity)

                        socketio.emit("update_arduino_data",
{"ldr_luminosity": ldrLuminosity})
                    elif line.startswith("isReadOn:"):
                        # Sends analog read on/off value to frontend
                        # isReadOn: 0
                        analogReadOn = bool(int(line[10:11]))

                        print("Analog read on: ", analogReadOn)
                        socketio.emit("update_analog_read_on",
{"analog_read_on": analogReadOn})
                    else:
                        print("not a case: ", line)
            except Exception as e:
                print(f"Error reading serial data: {e}")
        time.sleep(1.0)

threading.Thread(target=read_serial, daemon=True).start()

@socketio.on('connect')
def handle_connect():
    print("Client connected")

# Sends on/off signal to analog read task in arduino
@app.post('/api/switch_analog_read')
def switch_analog_read():
    data = request.get_json()

    analogReadOn = data["analogReadOn"]
    # Send the command to the Arduino
    cadena = 't' if analogReadOn == True else 'f'
    print(cadena)
    ser.write(cadena.encode('utf-8'))
    return jsonify({"status": "success", "message": "Analog read
updated"}), 200
```

```python
if __name__ == "__main__":
    socketio.run(app, debug=True, port=8080, host="192.168.100.99")
```

Codigo frontend:

```tsx
"use client"
import Image from "next/image";
import styles from "./page.module.css";
import Paper from '@mui/material/Paper';
import { Switch } from "@mui/material";
import { useState, useEffect } from "react";
import io from "socket.io-client";

const backendUrl = "http://192.168.100.99:8080";

const socket = io(backendUrl);

export default function Home() {
  const [luminosity, setLuminosity] = useState<number>(0);
  const [analogReadOn, setAnalogReadOn] = useState<boolean>(true);

  useEffect(() => {
    // receives luminosity data from the backend and sets alarm on if > 800
    socket.on("update_arduino_data", (data) => {
      setLuminosity(data.ldr_luminosity);

    });

    // receives analog read on/off data from the backend and sets alarm off if analog read is off
    socket.on("update_analog_read_on", (data) => {
      if (analogReadOn != data.analog_read_on) {
        setAnalogReadOn(data.analog_read_on);
      }

    });
  }, [socket, luminosity, analogReadOn]);

  // Sends post request to the backend to switch the analog read on or off and
  // turns alarm off if analog read is off
  async function switchRead() {
    setAnalogReadOn(!analogReadOn);


    const response = await
fetch(`${backendUrl}/api/switch_analog_read`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ analogReadOn: !analogReadOn }),
    });
    const data = await response.json();


  }

  return (
    <div className={styles.container}>
      <Paper elevation={3} className={styles.paper}>
        <div className={styles.innerContainer}>

          <h1>TP2: freeRTOS</h1>
          <p>Luminosity: {luminosity}</p>
          <p>Analog Read:</p>
          <Switch checked={analogReadOn} onChange={switchRead}/>

        </div>
      </Paper>

    </div>
  );
}
```

Ver [Figura 1] para ver los botones de la pagina web.

d) Parpadee el led 11 cada un segundo si la lectura está activada. No parpadee el led 11 si la lectura está desactivada.

```
#include <Arduino_FreeRTOS.h>

int sensorValue = -1;
```

```
bool isReadOn = true;

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );

// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on
LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  84  // This stack size can be checked & adjusted by reading the
Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the
highest, and 0 being the lowest.
    ,  NULL );


  // Now the task scheduler, which takes over control of scheduling
individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

/*
  Blink
  Turns on LED 11 on for one second, then off for one second, repeatedly.
  Only if read is on.
*/
void TaskBlink(void *pvParameters)
{
  (void) pvParameters;


  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(11, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    if (isReadOn) {
      digitalWrite(11, HIGH);   // turn the LED on (HIGH is the voltage level)
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(11, LOW);    // turn the LED off by making the voltage LOW
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (blink): ");
    Serial.println(stackRemaining);*/
  }
}
```

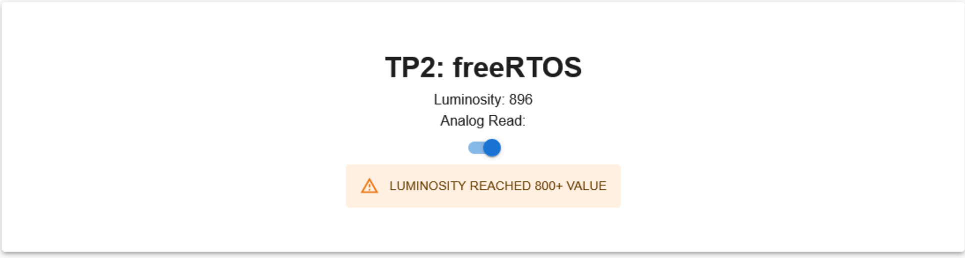
e) Si detecta que el valor de intensidad luminosa supera 800, encienda una alarma que:

Haga parpadear el led 12 con periodo de 0.1 segundo

```
// Blinks led 12 very fast if luminosity exceeds 800 value and read is on
void TaskAlarm(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  pinMode(12, OUTPUT);

  bool isAlarmActivated = false;

  for (;;) {
    if (sensorValue > 800 && isReadOn == true && isAlarmActivated == false) {
      isAlarmActivated = true;
    }

    if (isAlarmActivated == true) {
      digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage level)
      vTaskDelay( 100 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(12, LOW);    // turn the LED off by making the voltage LOW
      vTaskDelay( 100 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }

    if (isReadOn == false && isAlarmActivated == true){
      isAlarmActivated = false;
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (alarm): ");
    Serial.println(stackRemaining);*/
  }
}
```

Indique en la aplicación web la situación.

Podemos ver como se indica la situación en la [Figura 2].



TP2: freeRTOS

Luminosity: 896

Analog Read:

[Figura 2] Aplicación con alarma activada

La alarma se desactiva solo si la lectura se desactiva (la alarma no debe desactivarse si la lectura vuelve a estar por debajo de 800).

Codigo Arduino:

```
#include <Arduino_FreeRTOS.h>
// Include semaphore supoport
#include <semphr.h>

/*
 * Declaring a global variable of type SemaphoreHandle_t
 *
 */
SemaphoreHandle_t interruptSemaphore;

int sensorValue = -1;
bool isReadOn = true;

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

void TaskPressedButtonChecker( void *pvParameters);

void TaskAlarm( void *pvParameters);

void TaskReceiveInstructions( void *pvParameters);
```

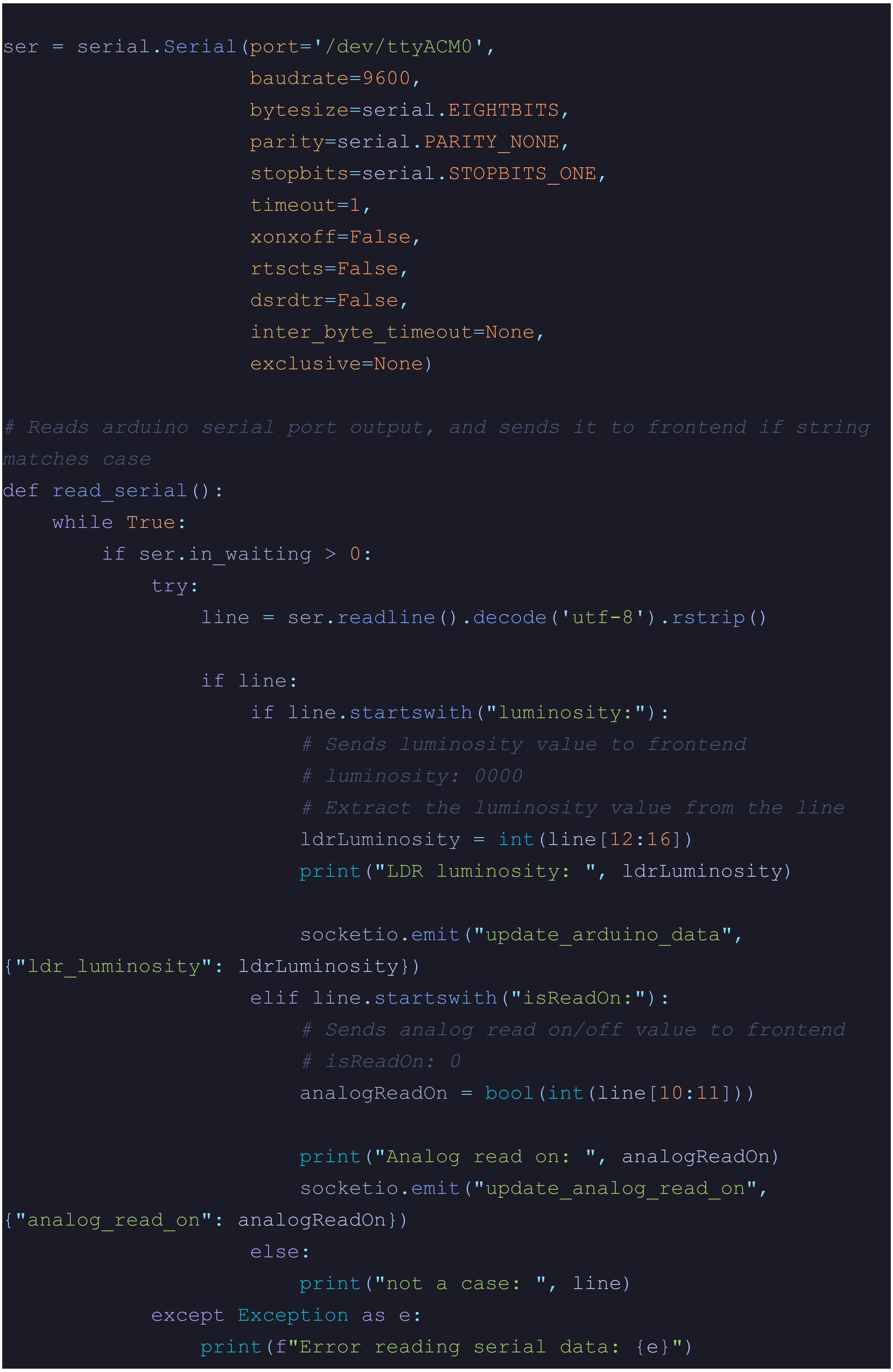
```
// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on
LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  84  // This stack size can be checked & adjusted by reading the
Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the
highest, and 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskPrintAnalogRead,
    "PrintAnalogRead",
    110,
    NULL,
    2,
    NULL
  );

  xTaskCreate(
    TaskPressedButtonChecker,
    "PressedButtonChecker",
    128,
    NULL,
    2,
    NULL
  );

  xTaskCreate(
    TaskAlarm,
    "Alarm",
    84,
    NULL,
    2,
    NULL
  );

  xTaskCreate(
    TaskReceiveInstructions,
    "ReceiveInstructions",
    126, //20 + 18 + 44 + 44
    NULL,
    2,
    NULL
  );

  /**
   * Create a binary semaphore.
   * https://www.freertos.org/xSemaphoreCreateBinary.html
   */
  interruptSemaphore = xSemaphoreCreateBinary();
  if (interruptSemaphore != NULL) {
    // Attach interrupt for Arduino digital pin
    attachInterrupt(digitalPinToInterrupt(2), interruptHandler,
FALLING);
  }

  // Now the task scheduler, which takes over control of scheduling
individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

void interruptHandler() {
  /**
   * Give semaphore in the interrupt handler
   * https://www.freertos.org/a00124.html
   */

  xSemaphoreGiveFromISR(interruptSemaphore, NULL);
```

```
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

/*
  Blink
  Turns on LED 11 on for one second, then off for one second,
repeatedly.
  Only if read is on.
*/
void TaskBlink(void *pvParameters)
{
  (void) pvParameters;


  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(11, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    if (isReadOn) {
      digitalWrite(11, HIGH);   // turn the LED on (HIGH is the voltage
level)
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(11, LOW);    // turn the LED off by making the
voltage LOW
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (blink): ");
    Serial.println(stackRemaining);*/
  }
}

// Blinks led 12 very fast if luminosity exceeds 800 value and read is
on
void TaskAlarm(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  pinMode(12, OUTPUT);

  bool isAlarmActivated = false;

  for (;;) {
    if (sensorValue > 800 && isReadOn == true && isAlarmActivated ==
false) {
      isAlarmActivated = true;
    }

    if (isAlarmActivated == true) {
      digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage
level)
      vTaskDelay( 100 / portTICK_PERIOD_MS ); // wait for one second
      digitalWrite(12, LOW);    // turn the LED off by making the
voltage LOW
      vTaskDelay( 100 / portTICK_PERIOD_MS ); // wait for one second
    } else {
      vTaskDelay(10);
    }

    if (isReadOn == false && isAlarmActivated == true){
      isAlarmActivated = false;
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (alarm): ");
    Serial.println(stackRemaining);*/
  }
}

// Prints analog read output as a string with a certain format to be
read by backend
void TaskPrintAnalogRead(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    if (isReadOn) {
      // read the input on analog pin 3:
      sensorValue = analogRead(A3);
      char luminosityBuffer[5];
      sprintf(luminosityBuffer, "%04d", sensorValue);
      Serial.println("luminosity: " + String(luminosityBuffer));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three second
    } else {
      vTaskDelay( 1000 / portTICK_PERIOD_MS ); // for some reason this
makes isReadOn update
    }
    Serial.println("isReadOn: " + String(isReadOn));
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (analogRead): ");
    Serial.println(stackRemaining);*/
  }
}

// Task that activates on interruption (button 2 pressed), it turns off
read
void TaskPressedButtonChecker(void *pvParameters) {
  (void) pvParameters;

  for (;;) {
    /**
    * Take the semaphore.
    * https://www.freertos.org/a00122.html
    */
    if (xSemaphoreTake(interruptSemaphore, portMAX_DELAY) == pdPASS) {


      isReadOn = !isReadOn;
      //Serial.println("semaphore taken, isReadOn: " +
String(isReadOn));
      Serial.println("isReadOn: " + String(isReadOn));
      vTaskDelay( 3000 / portTICK_PERIOD_MS ); // wait for three
seconds
    } else {
      vTaskDelay(10);
    }

    /*UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (button pressed): ");
    Serial.println(stackRemaining);*/
  }
```

```
}

// Reads from serial and if it reads a t turns on read, if it reads an
f it turns it off
void TaskReceiveInstructions( void *pvParameters) {
  (void) pvParameters;

  char buffer;


  for (;;) {
    if (Serial.available()>0) {

      buffer = (char)Serial.read();
      Serial.print("Buffer: ");
      Serial.println(buffer);

      if (buffer == 'f') {
        isReadOn = false;
      } else if (buffer == 't') {
        isReadOn = true;
      }
    }
    /*
    UBaseType_t stackRemaining = uxTaskGetStackHighWaterMark(NULL);
    Serial.print("stackRemaining (receiveInstructions): ");
    Serial.println(stackRemaining);*/
    vTaskDelay(10);
  }
}
```

Codigo Backend:

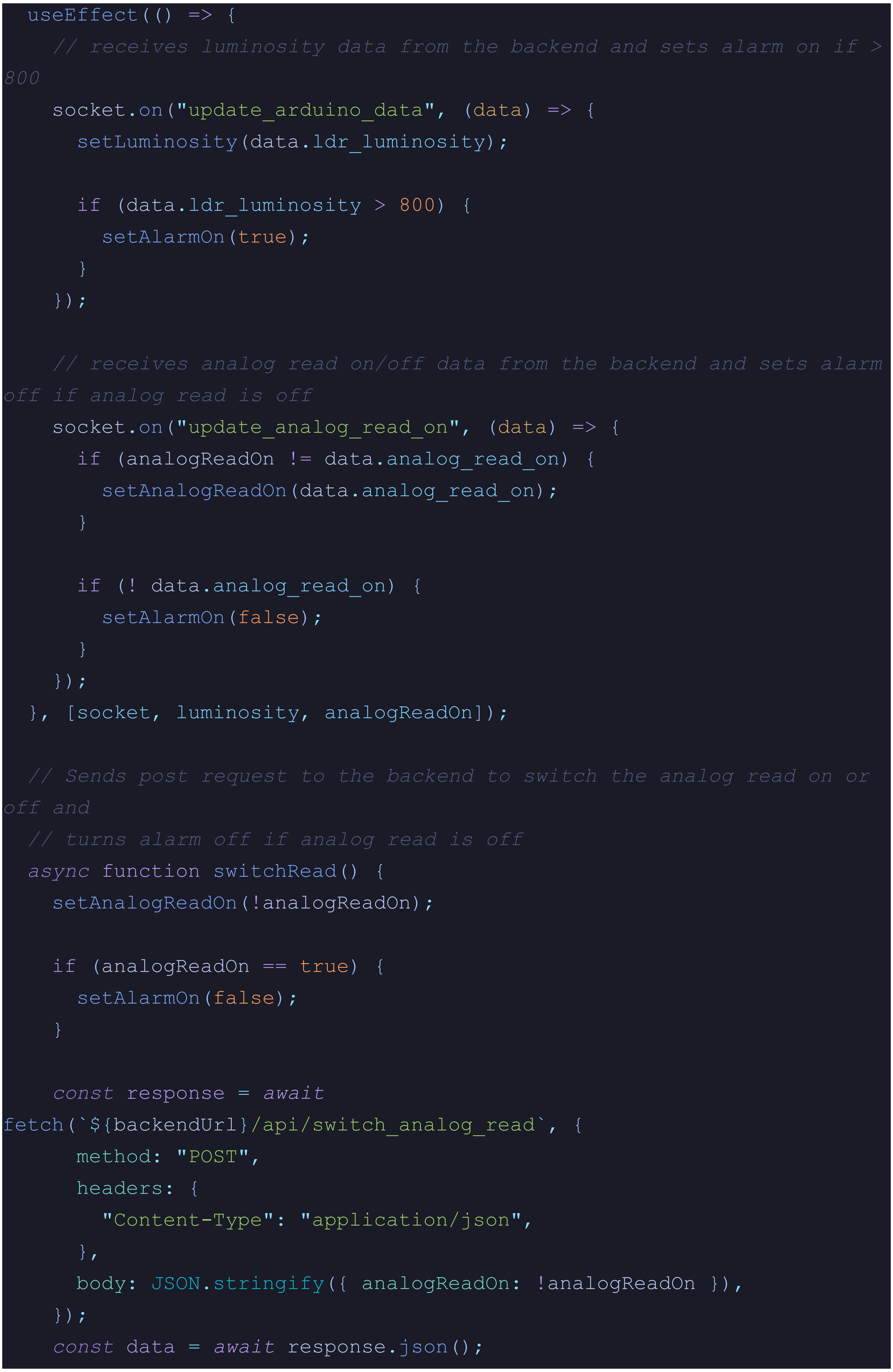
```
from flask import Flask, jsonify, request
from flask_cors import CORS
from flask_socketio import SocketIO
import serial
import threading
import time

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")
CORS(app, resources={r"/*": {"origins": "*"}},
supports_credentials=True)
```



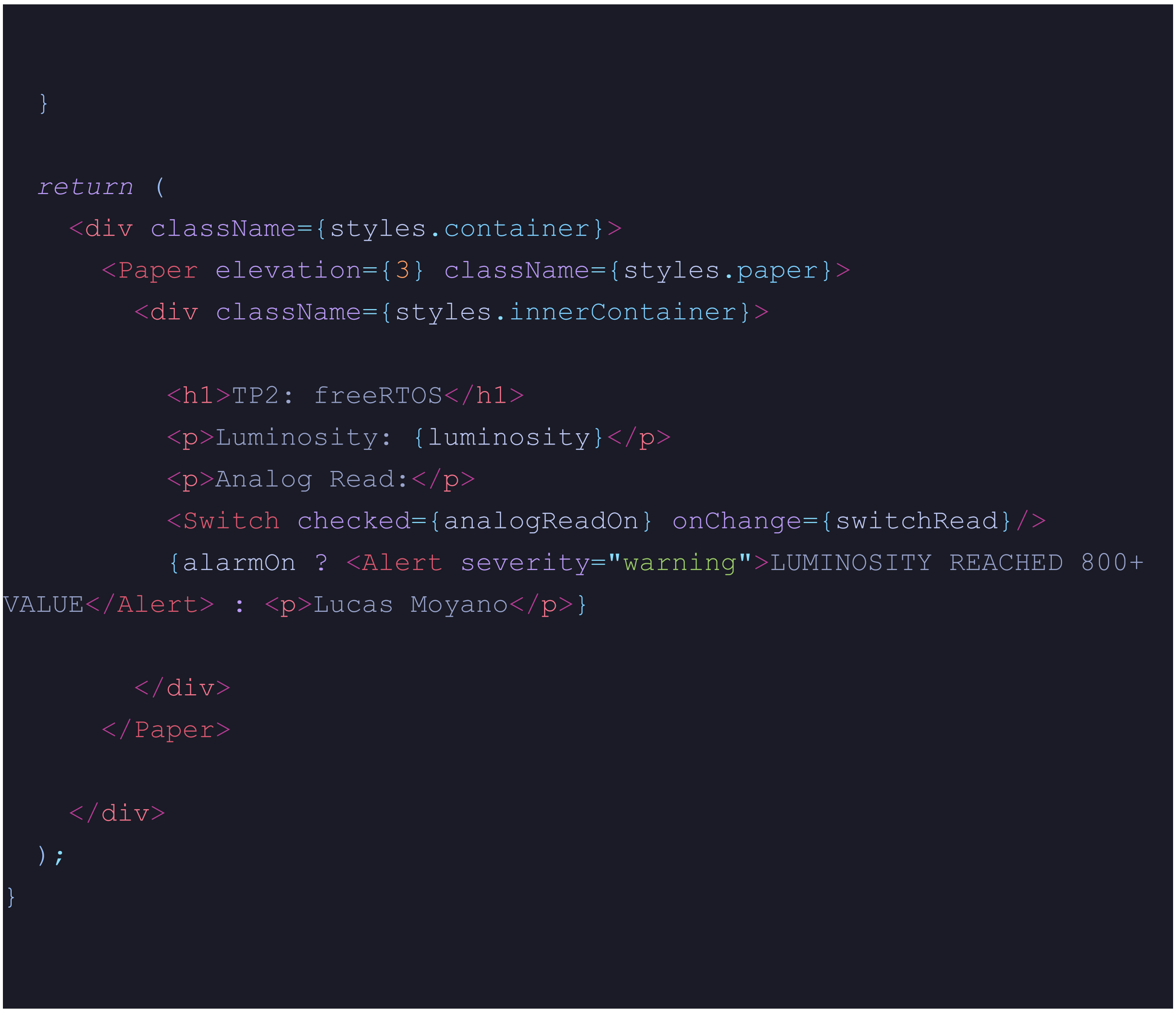
```
ser = serial.Serial(port='/dev/ttyACM0',
                    baudrate=9600,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    timeout=1,
                    xonxoff=False,
                    rtscts=False,
                    dsrdtr=False,
                    inter_byte_timeout=None,
                    exclusive=None)

# Reads arduino serial port output, and sends it to frontend if string
matches case
def read_serial():
    while True:
        if ser.in_waiting > 0:
            try:
                line = ser.readline().decode('utf-8').rstrip()

                if line:
                    if line.startswith("luminosity:"):
                        # Sends luminosity value to frontend
                        # luminosity: 0000
                        # Extract the luminosity value from the line
                        ldrLuminosity = int(line[12:16])
                        print("LDR luminosity: ", ldrLuminosity)

                        socketio.emit("update_arduino_data",
{"ldr_luminosity": ldrLuminosity})
                    elif line.startswith("isReadOn:"):
                        # Sends analog read on/off value to frontend
                        # isReadOn: 0
                        analogReadOn = bool(int(line[10:11]))

                        print("Analog read on: ", analogReadOn)
                        socketio.emit("update_analog_read_on",
{"analog_read_on": analogReadOn})
                    else:
                        print("not a case: ", line)
            except Exception as e:
                print(f"Error reading serial data: {e}")
        time.sleep(1.0)

threading.Thread(target=read_serial, daemon=True).start()

@socketio.on('connect')
def handle_connect():
    print("Client connected")

# Sends on/off signal to analog read task in arduino
@app.post('/api/switch_analog_read')
def switch_analog_read():
    data = request.get_json()

    analogReadOn = data["analogReadOn"]
    # Send the command to the Arduino
    cadena = 't' if analogReadOn == True else 'f'
    print(cadena)
    ser.write(cadena.encode('utf-8'))
    return jsonify({"status": "success", "message": "Analog read 
updated"}), 200


if __name__ == "__main__":
    socketio.run(app, debug=True, port=8080, host="192.168.100.99")
```

Codigo Frontend:

```
"use client"
import Image from "next/image";
import styles from "./page.module.css";
import Paper from '@mui/material/Paper';
import { Alert, Switch } from "@mui/material";
import { useState, useEffect } from "react";
import io from "socket.io-client";

const backendUrl = "http://192.168.100.99:8080";

const socket = io(backendUrl);

export default function Home() {
  const [alarmOn, setAlarmOn] = useState<boolean>(false);
  const [luminosity, setLuminosity] = useState<number>(0);
  const [analogReadOn, setAnalogReadOn] = useState<boolean>(true);

```



```
  useEffect(() => {
    // receives luminosity data from the backend and sets alarm on if >
800
    socket.on("update_arduino_data", (data) => {
      setLuminosity(data.ldr_luminosity);

      if (data.ldr_luminosity > 800) {
        setAlarmOn(true);
      }
    });

    // receives analog read on/off data from the backend and sets alarm
off if analog read is off
    socket.on("update_analog_read_on", (data) => {
      if (analogReadOn != data.analog_read_on) {
        setAnalogReadOn(data.analog_read_on);
      }

      if (! data.analog_read_on) {
        setAlarmOn(false);
      }
    });
  }, [socket, luminosity, analogReadOn]);

  // Sends post request to the backend to switch the analog read on or
off and
  // turns alarm off if analog read is off
  async function switchRead() {
    setAnalogReadOn(!analogReadOn);

    if (analogReadOn == true) {
      setAlarmOn(false);
    }

    const response = await
fetch(`${backendUrl}/api/switch_analog_read`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ analogReadOn: !analogReadOn }),
    });
    const data = await response.json();
```



```
  }

  return (
    <div className={styles.container}>
      <Paper elevation={3} className={styles.paper}>
        <div className={styles.innerContainer}>

          <h1>TP2: freeRTOS</h1>
          <p>Luminosity: {luminosity}</p>
          <p>Analog Read:</p>
          <Switch checked={analogReadOn} onChange={switchRead}/>
          {alarmOn ? <Alert severity="warning">LUMINOSITY REACHED 800+
VALUE</Alert> : <p>Lucas Moyano</p>}

        </div>
      </Paper>

    </div>
  );
}
```