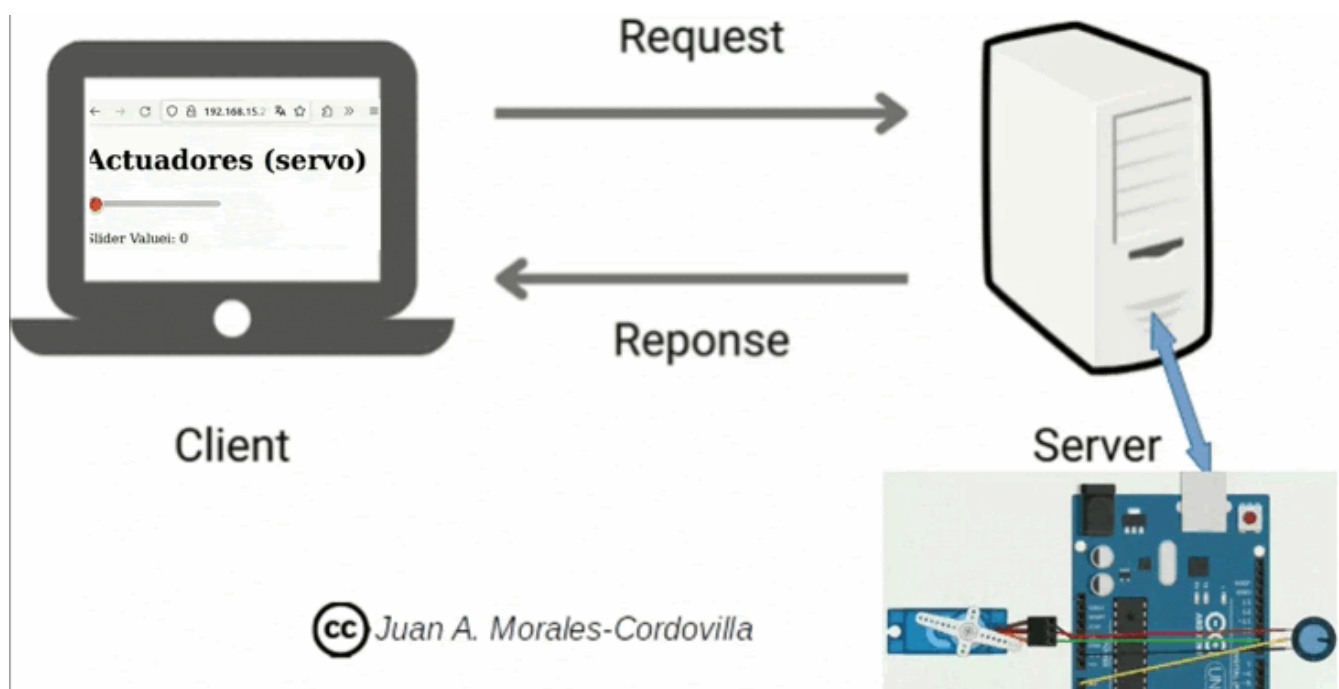


Control en tiempo real de un robot IoT via web

Proyecto: Control en tiempo real de un robot IoT via web

Presentación

Bienvenidos profesores y alumnos a este documento el cual es un proyecto para aprender algunos principios básicos del [Internet de las Cosas \(IoT\)](#), y construir al menos un sistema como el del siguiente GIF:



Este documento se estructura como sigue: la Sec. Guía Didáctica es para ser leída por el profesor, detalla la forma de trabajar el proyecto que es partir el producto final en 8 tareas. A continuación vienen las Sec. con explicaciones para que el alumnado vaya realizando estas 8 tareas. Finalmente concluimos el proyecto con un resumen y reflexión sobre el futuro del IoT.

Sin más, espero que disfruten de este proyecto.

Juan Andrés Morales Cordovilla

Guía Didáctica

Guía Didáctica (orientada al profesor)

A quién va dirigido: este proyecto titulado "*Control en tiempo real de un robot IoT via web*" es una práctica guiada que puede servir a cualquier persona interesada en el control remoto de robots, es decir en el Internet de las Cosas (IoT) en Tiempo Real (RT). Sin embargo se ha diseñado como Situación de Aprendizaje (SdA) para el alumnado de 2º de Bachillerato (materias de "*Programación y Computación*" (PYC) y "*Tecnologías de la Información y la Comunicación*" (TIC) II) o para el de ciclo de FP de 2º de SMR (módulo de "*Aplicaciones Webs*") de España (Andalucía).

Importancia: sobre la trascendencia del IoT en RT basta mencionar que es la base del control remoto de dispositivos como robots que trabajan en zonas donde el hombre no puede estar (ej. zona radioactiva). La importancia educativa de este proyecto reside en los 3 puntos siguientes:

1. Unificar muchos de los saberes y competencias que el alumnado debe adquirir en las etapas correspondientes de los cursos mencionados como Hardware, Programación y Redes.
2. Intentar que el alumno aprenda a su ritmo y sea evaluado automáticamente dejando al profesor como organizador del trabajo en grupo o de la reflexión del impacto de las TIC.
3. Aportar material al escaso existente sobre IoT en RT en castellano, de hecho es una mejora del proyecto titulado "[*Web Invernadero*](#)" del mismo autor como explica este [pdf](#).

Saberes y competencias a adquirir: a nivel técnico se espera mejorar en lo siguiente:

- Arduino: instalación y control de actuadores (servomotor, led..) y sensores (potenciómetro, botones) con C/C++
- HTML y JavaScript: modificación de elementos en web en RT
- Node.js: envío de mensajes con identificador entre cliente y servidor mediante sockets webs de RT

Estos saberes contribuirán a adquirir las competencias mostradas en este [pdf](#) y tomadas de la [Orden 30 de Mayo de 2023](#)

Duración y conocimientos previos: el proyecto está previsto para unas 8 horas de clase, para cumplirlo el alumnado de haber adquirido previamente los conocimientos de HTML, JavaScript y Arduino C/C++ que se suelen enseñar en dichas materias. Es por ello que se recomienda hacerlo en los últimos meses del curso.

Recursos necesarios: para realizar este proyecto se requiere:

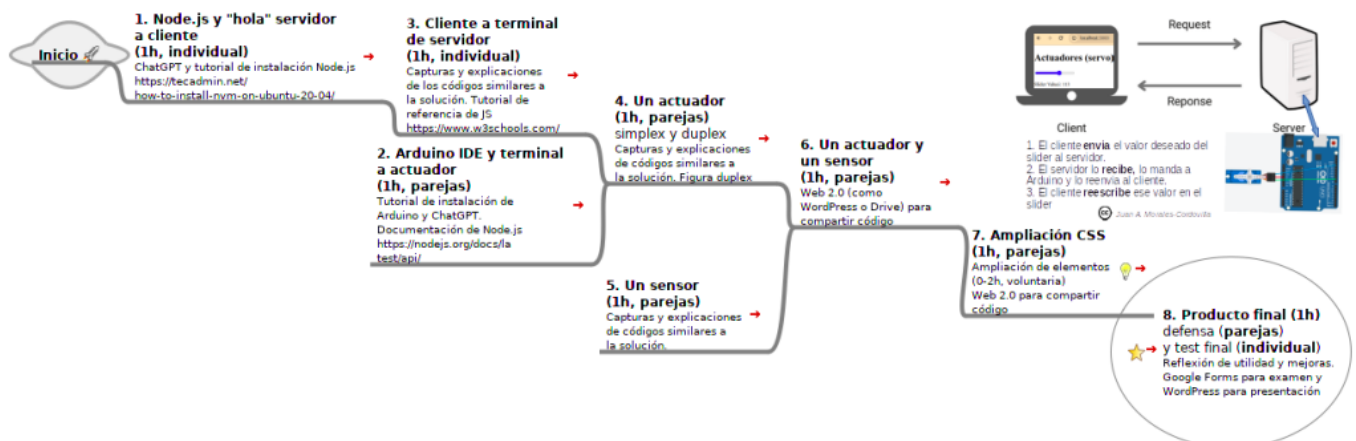
- Un computador (a ser posible con Linux Ubuntu 22.04) que pueda hacer de cliente y servidor a la vez (mediante un navegador y localhost).
- Un Arduino (a ser posible UNO) con, cables, servomotor y potenciómetro (y a ser posible otros componentes clásicos como botones y leds).
- Conexión a Internet: para descargar el software libre Node.js y Arduino IDE.
- Opcionalmente: otro computador (móvil, PC, tablet...) conectado a la misma red que el primero (como Andared) para hacer pruebas remotas.

En el siguiente [Symbaloo](#) se da una lista de otros recursos, software de apoyo como ChatGPT o la documentación de Node.js:

https://www.symbaloo.com/embed/shared/AAAABFi737UAA41_linnOQ==

Metodología y tareas: usamos la metodología activa de Aprendizaje Basado en Proyectos (ABP). Para ello se van proponiendo tareas (individuales o en parejas) que crecen en dificultad hasta alcanzar el producto final de nuestra SdA como muestra el siguiente mapa mental:

Tareas del proyecto "Control en tiempo real de un robot IoT via web"



Algunas de estas tareas como la última intentan transmitir valores trasnversales.

Evaluación: el alumnado empleará una entrada de su Blog (tipo WordPress y que luego defenderá) para ir resumiendo lo aprendido e ir recopilando las capturas de pantalla firmadas de las tareas. Firmar querrá decir: poner su nombre en algún comentario, usar variables con su nombre o traducir los comentarios del profesor al castellano. Siempre respetando las [licencias de uso](#).

Los Criterios de Evaluación de 2º Bach. asociados a cada tarea se detallan en este [otro pdf](#).

The screenshot shows a PDF viewer interface with the title 'TareasCE.pdf' and page number '1 / 5'. The zoom level is '78%'. The document content includes the 'aulavirtual' logo and the title 'Relacionamos tareas con criterios de evaluación'. Below the title, there is a table with two columns: 'TAREA' and 'CRITERIO EVALUACIÓN'. The first row of the table contains the text 'Instalación de Node.JS y “hola mundo” web (al compañero)' in the 'TAREA' column and 'PRYC.3.2_Proc TIYC.2.3.1_HT' in the 'CRITERIO EVALUACIÓN' column.

TAREA	CRITERIO EVALUACIÓN
Instalación de Node.JS y “hola mundo” web (al compañero)	PRYC.3.2_Proc TIYC.2.3.1_HT

Finalmente añadir que algunas tareas son voluntarias pero las 6+1 más importantes aparecerán en una sección aparte tipo tarea. Estas se evaluarán usando esta [diana de evaluación](#) interactiva que automáticamente computa también la calificación por CE:

Diana de evaluación de las principales tareas del proyecto IoT

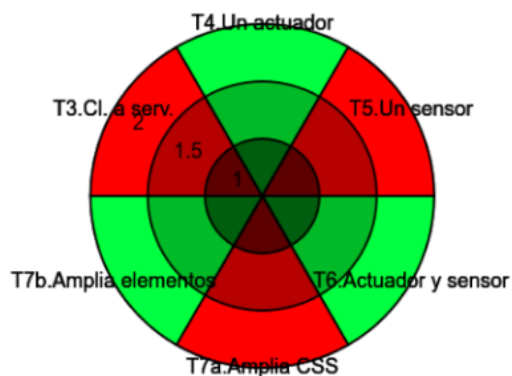
Nombre del alumno/a:

Calificaciones de las tareas: [T3,T4,T5,T6,T7a,T7b] NOTA: La calificación de tarea 8 sale de [este Test](#) y no se incluye.

Suma de las calificaciones de las tareas: aprueba si > 6

Calificaciones por CE [asociados a las tareas](#): [TIYC.2.1.1 _Impacto, TIYC.2.3.1 _HTML, TIYC.2.5.2 _colaborar, TIYC.2.5.3 _subprobl]

Marque en la diana la puntuación en las [seis tareas](#) más relevantes del proyecto y se recalcularán las calificaciones automáticamente:



Puntos según nivel competencial:

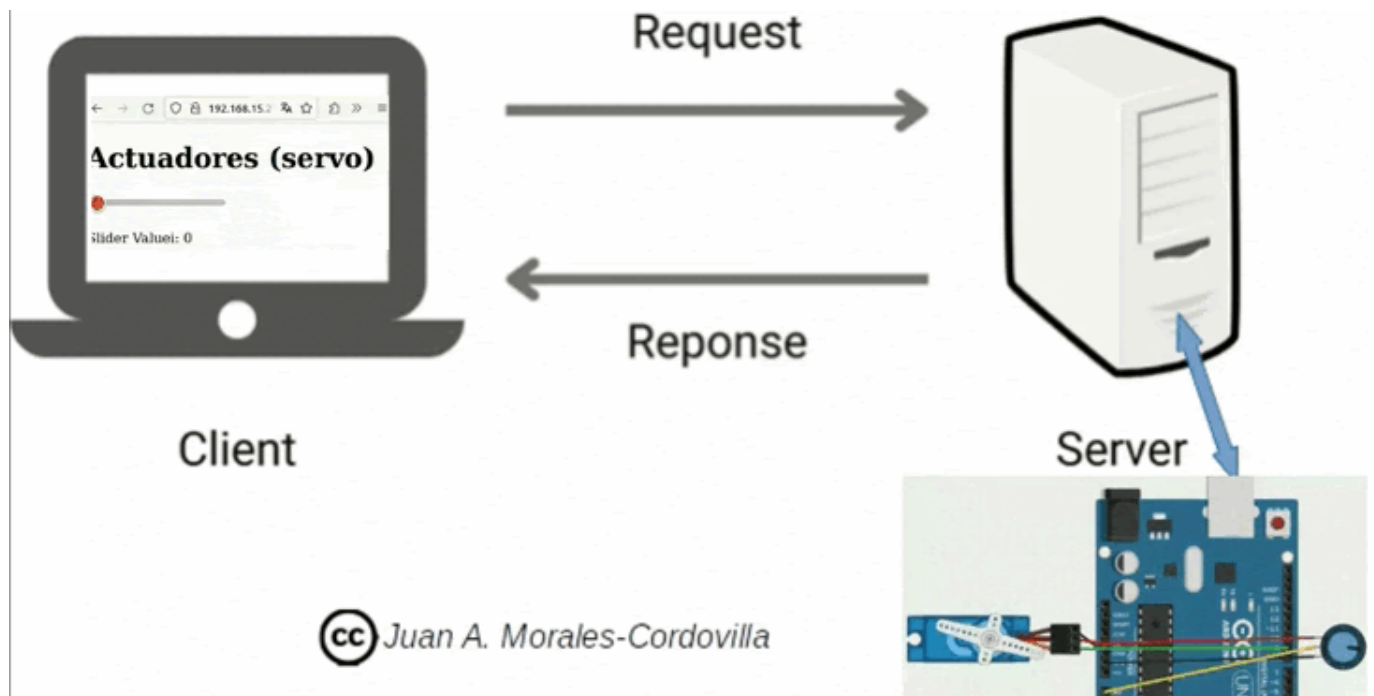
- 0: no logrado (no marcado)
- 1: logrado con mucha ayuda del profesor
- 1.5: logrado casi independientemente
- 2: logrado y mejorado independientemente

CC Juan A. M. Cordovilla

1. Instalación de Node.js y hola mundo (1h, individual)

Definición de IoT e instalación de Node.js

Qué pretendemos con este proyecto: aunque en la Presentación ya lo hemos mencionado, te lo recordamos en simples palabras: que aprendas a controlar remotamente y en tiempo real al menos los dos elementos básicos de cualquier robot: un actuador y un sensor como muestra el GIF de abajo:



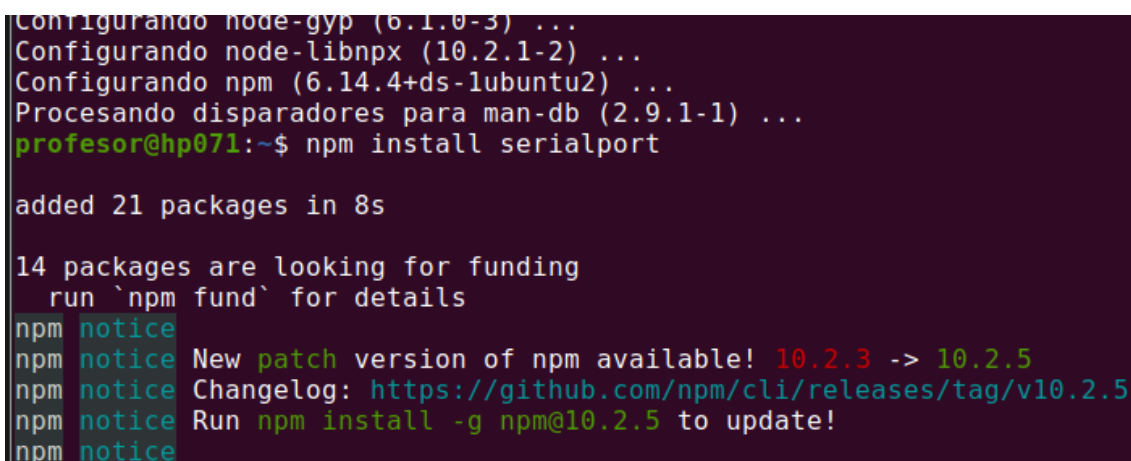
Para ello te iremos explicando unos cuantos conceptos clave (como IoT, RT,..) y partiremos el problema de llegar al producto final en subproblemas o tareas según este [mapa mental](#) y que te evaluaremos con esta [diana](#).

- + [Blog y licencias de uso](#)
- + [Internet de las Cosas \(IoT\) en Tiempo Real \(RT\)](#)
- + [Node.js](#)

Tarea: instalar curl y Node.js usando el bash script de abajo. Pega captura firmada en tu Blog de la instalación y completa los # (que son comentarios) para mostrar que la comprendes.

```
#Instalar Node por defecto
sudo apt update
sudo apt install nodejs
#Sal de superusuario y continua como usuario
node -v #node v10
# Si te falta instala curl.
sudo apt install curl
# Instalar una NVM version como este enlace: https://tecadmin.net/how-to-install-nv
#Instala los "esenciales"
sudo apt-get install build-essential
#Checkea los esenciales
checkinstall libssl-dev
# Instala NVM
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.32.1/install.sh | bash
# Reinicia terminal
# Version
nvm --version
#Ultimo node 20
nvm install 20
#Ultimo npm 10 (con ultimo nvm)
sudo apt install npm
#Ultimo serialport y express
npm install serialport express
```

Ej. de captura



```
Configurando node-gyp (6.1.0-3) ...
Configurando node-libnpa (10.2.1-2) ...
Configurando npm (6.14.4+ds-1ubuntu2) ...
Procesando disparadores para man-db (2.9.1-1) ...
profesor@hp071:~$ npm install serialport

added 21 packages in 8s

14 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New patch version of npm available! 10.2.3 -> 10.2.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.5
npm notice Run npm install -g npm@10.2.5 to update!
npm notice
```

Tarea: haz un «hola mundo» en Node.js para ver que todo está OK usando ChatGPT. Sube captura similar a esta:

The screenshot shows a code editor with a file named `app.js` containing two lines of JavaScript code:

```
1 console.log("Hello, World!");
2
```

Below the code editor, a terminal window shows the command `node app.js` being executed, resulting in the output `¡Hola, mundo!`.

Observa que:

- `console.log(«¡Hola, mundo!»);`: imprime por consola texto
- `node app.js` lanza `app.js` en modo servidor, se recomienda no lanzarlo como superusuario

Hola mundo web

Tarea: haz un «hola mundo» node.js que salga en el navegador. Suba una captura con del código comentado, la ejecución y el resultado

The screenshot shows a web browser at `localhost:3000` displaying the text `¡Hola, mundo Juan Andrés!`. In the background, a code editor (Kate) shows the `app.js` file with the following code:

```
1 // app.js
2 const express = require('express');
3 const app = express();
4 const port = 3000;
5
6 // Ruta para manejar las solicitudes GET
7 app.get('/', (req, res) => {
8   // Envía una respuesta con el mensaje personalizado
9   res.send(`¡Hola, mundo Juan Andrés!`);
10 });
11
12 // Inicia el servidor en el puerto especificado
13 app.listen(port, () => {
14   console.log(`Servidor escuchando en http://localhost:${port}`);
15 });
```

Observe que:

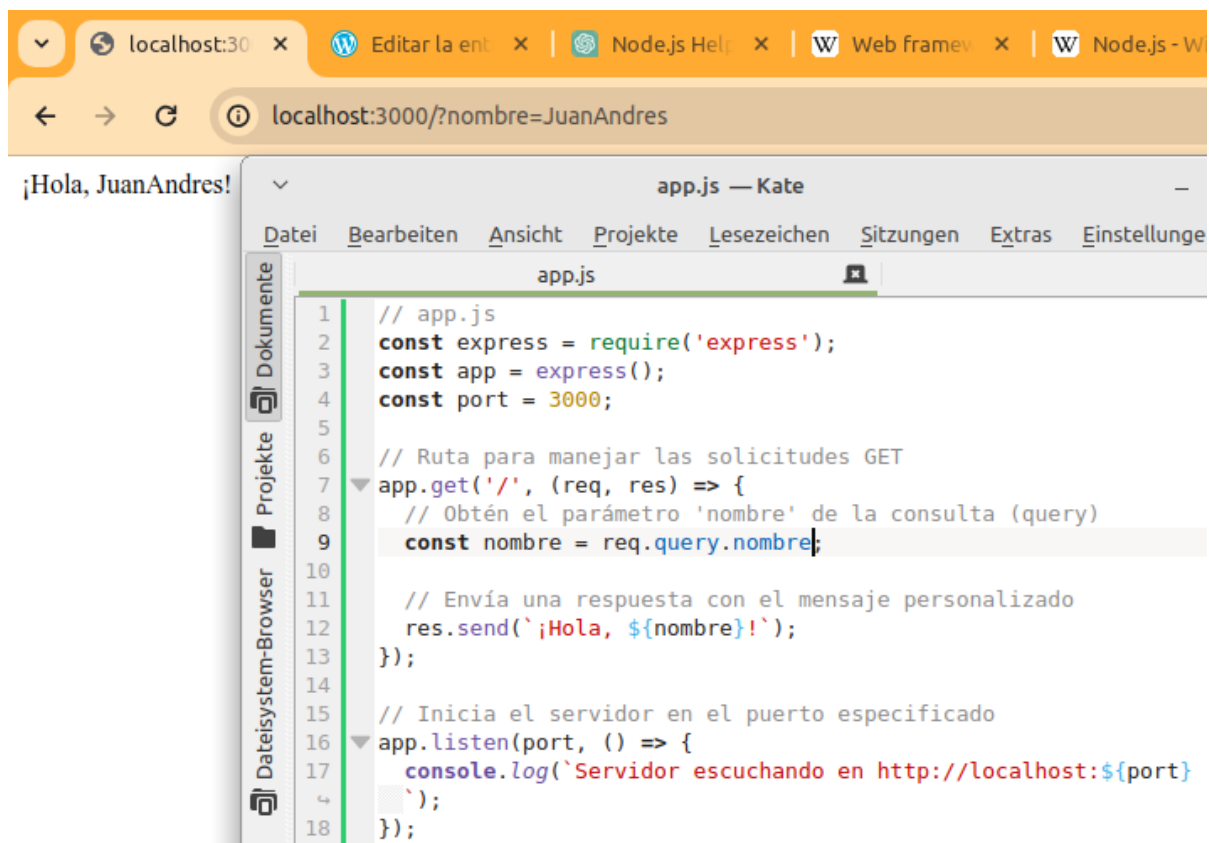
- Express.js: es el «estándar de facto» frameworks o librería de los servidores Node.js.
`const express=require('express');` importa dicha librería en objeto `express`

- `const app = express();` crea el objeto que va a representar toda nuestra aplicación Node.js. App tiene muchos métodos para configurar app, definir rutas, manejar solicitudes y respuestas...
- `app.get()`: función asociada a peticiones GET (por URL aunque aquí no se usa)
- `app.listen(port)`: inicia tu PC como servidor escuchando en puerto 3000
- Para lanzarlo escribimos en el navegador la URL: IP+:puerto del servidor Ej. si el servidor eres tu mismo se pone localhost o 127.0.0.1 <http://localhost:3000/>

Tarea: conéctate al servidor de tu compañero de clase y pon captura

- NOTA: es importante que el cliente y servidor estén dentro de la misma red (Ej: Andared_Cooperativo donde las IP puede ser 192.168.53.87)

Tarea-voluntaria: cree un «hola mundo» que reciba por GET un nombre como muestra la Fig.:



Observe que:

- El envío del nombre (Juan) se hace por la URL del cliente: <http://localhost:3000/?nombre=JuanAndres>
- `req.query.nombre`: saca el nombre (Juan) del envío recibido en el servidor
- `res.send(`¡Hola, ${nombre}!`);`: junta la cadena 'Hola' con la variable nombre

2. Instalación de Arduino IDE y control por terminal de actuador (1h, en parejas)

Instalación de Arduino IDE

+ Arduino IDE

Tarea: instala Arduino IDE en Linux

1. Ir a la web Arduino IDE



2. Descarga Linux App Image 64 bit (a la derecha). Llevarlo a una carpeta segura (ej. /home/usuario/Escritorio/Juan), darle permiso de ejecución (chmod u+x arduino...) y lanzarlo

Tarea-voluntaria: enciende y apaga el pin-13, 2 veces rápido y 2 lento y "flashealo" sobre tu Arduino para ver que funciona. Si no sabes pregunta a ChatGPT como hacerlo.

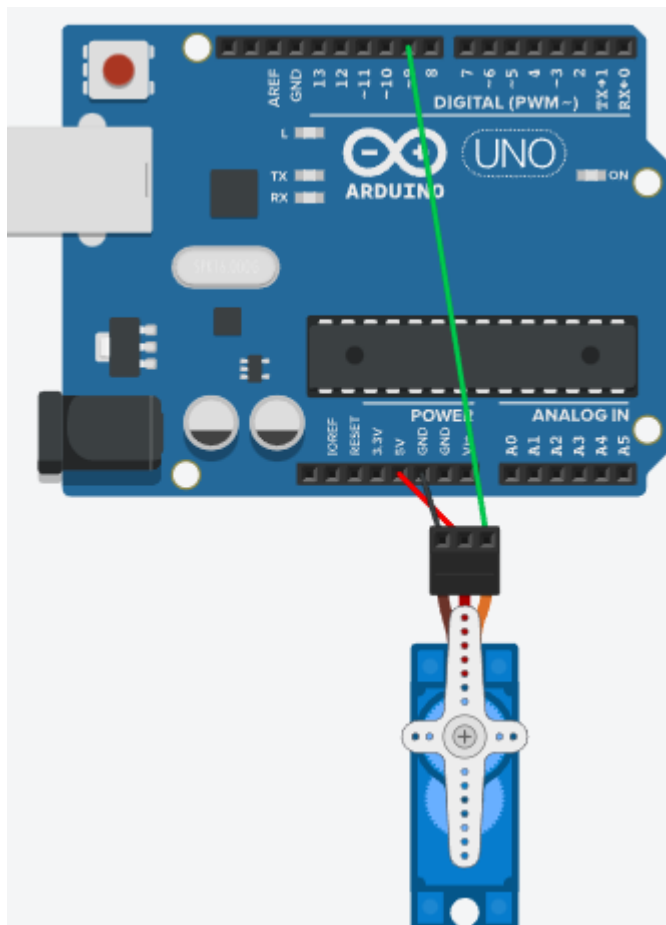
Control de actuador por puerto serie y terminal

+ Trabajo en parejas

+ Robot

Tarea: trabajando en parejas como hemos explicado antes, mueve el actuador servomotor enviando los grados por la terminal hacia el puerto serie.

1. Conecta la tierra (negro), la alimentación (rojo) y la señal (verde) del servo al pin 9 como abajo:



2. Pásale un código para que el ángulo del servo se ponga al valor recibido por puerto serie:

Código soloServo.ino:

```

1  #include <Servo.h>
2  const int servPin=9;
3  int inputPosition;
4  Servo myServo; // Create servo object
5
6  void setup()
7  {
8      myServo.attach(servPin); // Servo signal line to Ar
9      myServo.write(90); // Calibrate Servo
10     Serial.begin(9600);
11 }
12
13 void loop()
14 {
15     if(Serial.available()) {
16         inputPosition = Serial.parseInt();
17         if(inputPosition >= 10 && inputPosition < 170) {
18             myServo.write(inputPosition);
19         }
20     }
21 }

```

Observaciones:

- Serial.begin(9600): establece que la comunicación por el puerto será a 9600 baudios o bps (bits per seconds)
- inputPosition=Serial.parseInt(): lee texto del puerto serie hasta un salto de línea '\n' y lo convierte en entero que se almacena en inputPosition.
- Los valores min. y max. del ángulo del servo están entre 10 y 170 grados

3. Envíe un ángulo desde la terminal y comprobar que se mueve. Quizás sea necesario tener abierto Arduino IDE, enviar en modo root o añadirle salto de línea al número (echo «40\n» > /dev/ttyACM0)

```

juan@jamc:~/Schreibtisch/CursoB2Digital/PractGuiadaExeLearning$ sudo su
root@jamc:/home/juan/Schreibtisch/CursoB2Digital/PractGuiadaExeLearning# echo "40\n" > /dev/ttyACM0
root@jamc:/home/juan/Schreibtisch/CursoB2Digital/PractGuiadaExeLearning#

```

- /dev/ttyACM0: es el archivo dispositivo (orientado a caracteres) asociado al USB. Puede variar y en Windows suele ser COM1.

3. Envío de datos del cliente a terminal del servidor en RT (1h, individual)

T3. Cliente a servidor

Duración: 1h

Agrupamiento: individual

+ Socket e Id de mensaje

Nota sobre el agrupamiento: aunque la ejecución es individual, la copia de los códigos puede hacerse en parejas, uno index.html y sketch.js, otro app.js (siguiendo el principio de partir en subproblemas mencionado en *Trabajo en Parejas*).

Tarea: haz una web cliente con slider (deslizador) que envíe su valor en Tiempo Real (RT) y que el servidor saque este valor por su consola (terminal)

1. Crea una carpeta llamada p2SliderAConsola y dentro de esta los archivos y directorios siguientes:

```
p2SliderAConsola
├── app.js
├── publicJuan
│   ├── index.html
│   └── sketch.js
```

2. Copia los siguientes códigos y coméntalos en español:

index.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Serial Data Viewer</title>
</head>

<body>
  <h1> Actuadores (servo) </h1>
  <input type="range" id="slider" min="0" max="180" value="90">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.1.3/socket.io.js"></script>
  <script src="sketch.js"> </script>
</body>
</html>

```

- <input type=»range»..> crea el slider
- <script src=»sketch.js»> enlaza el JavaScript

sketch.js:

```

//Object socket from class io
const socket = io();
//object for the slider
const slider = document.getElementById('slider');

/*Slider servomotor*/
//sent slider value to server
slider.addEventListener('input', () => {
  const value = slider.value;
  socket.emit('sliderValueJuan', value); //sent value with Id sliderValueJuan
});

```

- slider: representa el objeto deslizador y slider.value su valor en cada instante
- slider.addEventListener('input', () => {...}): a cualquier elemento HTML le podemos asociar un escuchador de eventos con el formato element.addEventListener(event, function). Event suele ser 'input', 'click', 'onclick' o 'mousedown'
- () => {...}: define la función anónima a lanzar en cada evento, en este caso enviar al servidor un mensaje con Id sliderValueJuan mediante el socket creado. Esta notación se llama 'arrow function expresión' y se podría haber reemplazado por: function() {...}. Se usa junto a map, filter, .. en programación funcional.

app.js:

```
//Carga de frameworks
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
//
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
const port = 3000;

//only the files from public can be served to a client
app.use(express.static('./publicJuan'));

//WebSocket connection handling
io.on('connection', (socket) => { //server waiting client connection event

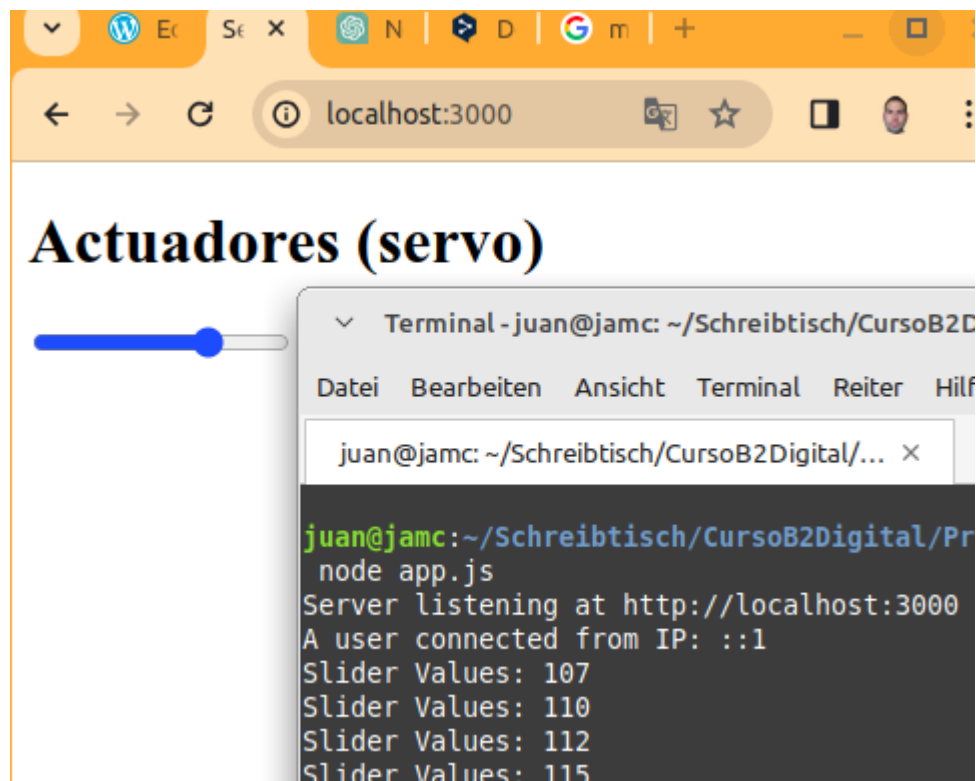
  const userIP = socket.handshake.address; //IP del cliente |
  console.log(`A user connected from IP: ${userIP}`); //message when client connected

  /*Slider servomotor */
  // Function to execute when a message with Id 'sliderValueJuan' is recieved
  socket.on('sliderValueJuan', (value) => { //receive client value
    console.log(`Slider Values: ${value}`); //console
  });
});

// Start the server
server.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});
```

- express, app y port: son como antes, pero ahora el servidor no lo inicia app.listen sino server.listen (que es de tipo http) por puerto 3000.
- io.on(): es una función que recibe un objeto socket (estructura con datos como IP del cliente...). Permite comunicación RT bidireccional (duplex) y lanza procesos cuando un nuevo cliente se conecta al servidor. El primer proceso es mostrar la IP del cliente por consola del servidor
- socket.on('sliderValueJuan', (value) ..): es un proceso de io.on que cada vez que recibe un mensaje con Id 'sliderValueJuan' del cliente, el servidor lo saca por su consola.

3. Resultado de ejecución:



- Cuando un nuevo cliente se conecta muestra su IP

Tarea: conéctate al servidor de tu compañero y mueve el slider. Pon captura donde se vean los resultados

4. Control en RT de un actuador (1h, en parejas)

Control simplex en RT de un actuador

+ Comunicación simplex

Tarea: modifica el código anterior para que el cliente mueva un servo del servidor

1. Conecta el servo a Arduino y métele el código soloServo.ino que vimos más arriba.
2. Los códigos index.html y sketch.js son como antes, solo cambia el app.js de abajo:

app.js:

```
1  |//Carga de frameworks
2  const { SerialPort } = require('serialport')
3  const serialport = new SerialPort({path: '/dev/ttyACM0', baudRate: 9600 })
4  //
5  const express = require('express');
6  const http = require('http');
7  const socketIo = require('socket.io');
8  //
9  const app = express();
10 const server = http.createServer(app);
11 const io = socketIo(server);
12 const port = 3000;
13
14 //only the files from public can be served to a client
15 app.use(express.static('./publicJuan'));
16
17 //WebSocket connection handling
18 io.on('connection', (socket) => { //server waiting client connection event
19
20     const userIP = socket.handshake.address; //IP del cliente
21     console.log(`A user connected from IP: ${userIP}`); //message when client connected
22
23     /*Slider servomotor */
24     // Function to execute when a message with Id 'sliderValueJuan' is recieved
25     socket.on('sliderValueJuan', (value) => { //receive client value
26         console.log(`Slider Values: ${value}`); //console
27         serialport.write(`${value}\n`); //write value to serial port
28     });
29 });
30
31 // Start the server
32 server.listen(port, () => {
33     console.log(`Server listening at http://localhost:${port}`);
34 });
```

- `const { SerialPort } = require('serialport');` importa la librería del puerto serie. Se pone entre llaves para «asignar des-estructurando» el módulo (para extraer y asignar de un solo paso)
- `serialport.write(`${value}\n`);` escribe al puerto `value` y le añade un salto de línea (`\n`)

Tarea: conectate al servidor de tu compañero y mueve su servo.

T4. Un actuador (control duplex)

Duración: 1h

Agrupamiento: en parejas

Tarea: El código simplex anterior es sencillo y lo normal es que el servidor también reenvíe al cliente el dato recibido, haya comunicación duplex. Para ello, modifica los 3 códigos como abajo. Una vez le funcione guarde capturas y muéstresele al profesor.

app.js:

```
//Carga de frameworks
const { SerialPort } = require('serialport')
const serialport = new SerialPort({path: '/dev/ttyACM0', baudRate: 9600 })
//
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
//
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
const port = 3000;

//only the files from public can be served to a client
app.use(express.static('./publicJuan'));

//WebSocket connection handling
io.on('connection', (socket) => { //server waiting client connection event

  const userIP = socket.handshake.address; //IP del cliente
  console.log('A user connected from IP: ${userIP}'); //message when client connected

  /*Slider servomotor */
  // Function to execute when a message with Id 'sliderValueJuan' is received
  socket.on('sliderValueJuan', (value) => { //receive client value
    io.emit('updateSliderJuan', value); // broadcast the updated value to all connected clients
    console.log('Slider Values: ${value}'); //console
    serialport.write(`${value}\n`); //write value to serial port
  });
});

// Start the server
server.listen(port, () => {
  console.log('Server listening at http://localhost:${port}');
});
```

- `io.emit('updateSliderJuan', value);` emite por broadcast (a todos los clientes conectados) el valor actual del servo. El id del mensaje es `'updateSliderJuan'`

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Serial Data Viewer</title>
</head>

<body>
  <h1> Actuadores (servo) </h1>
  <input type="range" id="slider" min="0" max="180" value="90">
  <p id="sliderValue">Slider Value: 90</p>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.1.3/socket.io.js"></script>
  <script src="sketch.js"> </script>
</body>
</html>
```

- <p id=»sliderValue»>: crea un párrafo con ese id que mostrará en RT el valor nº del slider

sketch.js:

```
//Object socket from class io
const socket = io();
//object for the slider
const slider = document.getElementById('slider');
const sliderValue = document.getElementById('sliderValue');

/*Slider servomotor*/
//sent slider value to server
slider.addEventListener('input', () => {
  const value = slider.value;
  socket.emit('sliderValueJuan', value); //sent value with Id sliderValueJuan
});

//replace the slider value number with the latest»
socket.on('updateSliderJuan', (value) => { //receive server value with id updateSliderJuan
  slider.value = value;
  sliderValue.textContent = `Slider Value: ${value}`;
});
```

- const sliderValue: objeto que representa el slider asociado con getElementById
- socket.on('updateSliderJuan', (value)...): socket que espera a recibir el mensaje del servidor con Id 'updateSliderJuan'. Dicho mensaje incluye el valor del servo actual
- sliderValue.textContent y slider.value: modifican el slider y el párrafo del index.html con el valor actual del servo recibido. Esto cierra el «ciclo de comunicación duplex» en RT como muestra la Fig.

+ Ciclo de comunicación duplex

Tarea: reflexiona sobre las ventajas e inconvenientes del *ciclo de comunicación duplex*.

5. Control de un sensor en RT (1h, en parejas)

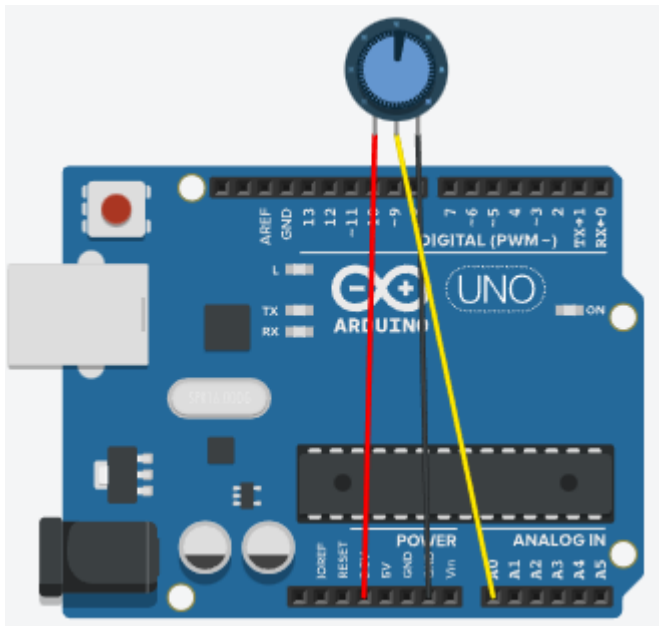
T5. Un sensor

Duración: 1h

Agrupamiento: en parejas

Tarea: haz una web que muestre en RT el valor de un potenciómetro

1. Conecta el potenciómetro a Arduino con la tierra (negro) a GND, alimentación (rojo) a 3.5V y señal (amarillo) al A0



2. Pásale a Arduino este código:

```
//Juan Andres
const int potPin = A0;
int value;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  value=analogRead(potPin);
  value=map(value,0,690,0,255);
  Serial.println(value);
}
```

- value=analogRead(potPin): lee el valor del potenciómetro cuyo valor mínimo es 0 y máximo 690 (a 3.5V)
- map(value,0,690,0,255): mapea el valor del potenciómetro entre 0 y 255
- Serial.println(value): saca por puerto serie el valor y saca salto de linea '\n' (println) al final

3. Copia los siguientes 3 códigos con la misma estructura de ficheros anterior:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Juan A. Serial Data Viewer</title>
</head>

<body>
  <h1>Sensores (potenciómetro) </h1>
  <div id="serialDataBox"></div>
  <script src="https://cdn.socket.io/4.1.3/socket.io.min.js"></script>
  <script src="sketch.js"> </script>

</body>
</html>
```

- <div id="serialDataBox"></div> crea el divisor donde JS escribirá en RT los valores del potenciómetro. Su id es la que aparece

sketch.js

```
//Juan A. Morales|
const socket = io();
const serialDataBox = document.getElementById('serialDataBox');

/*Box potentiometer*/
socket.on('serialDataJuan', (data) => {
  // Replace the content of the box with the latest data
  serialDataBox.textContent = data;
});
```

- const serialDataBox: objeto que representa el <div> anterior
- socket.on('serialDataJuan', (data)..) cuando recibe un mensaje del servidor con id 'serialDataJuan' y valor del potenciómetro, modifica el valor del objeto anterior.

app.js

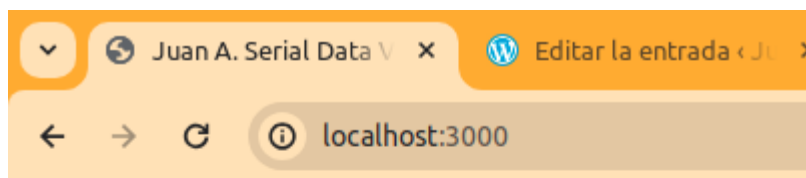
```
//Juan A.
const { SerialPort } = require('serialport')
const serialport = new SerialPort({path: '/dev/ttyACM0', baudRate: 9600 })
const { ReadlineParser } = require('@serialport/parser-readline');
//
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
//
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
const port = 3000;
//only the files from public can be served to a client
app.use(express.static('./publicJuan'));

//WebSocket connection handling
io.on('connection', (socket) => { //server waiting for the connection client event
  //
  const userIP = socket.handshake.address; //IP del cliente
  console.log(`A user connected from IP: ${userIP}`); //message when client connected

  /*Box potentiometer*/
  // Create a parser to read lines from the serial port
  const parser = serialport.pipe(new ReadlineParser({ delimiter: '\n' }));
  // Listen for incoming serial data and emit it to the connected clients using WebSocket
  parser.on('data', (data) => { //when parser receives a data (until delimiter)
    //execute the following
    io.emit('serialDataJuan', data.trim()); //broadcast message (with id=serialDataJuan)
    //to all clients connected to socket
  });
});
// Start the server
server.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});
```

- const { ReadlineParser }: librería de parseo (análisis) de datos del puerto serie
- const parser = serialport.pipe(new ReadlineParser({ delimiter: '\n' }));: parseador que leerá dato hasta encontrar salto línea '\n'
- parser.on('data', (data)=>...): parseador escuchando y cuando detecta '\n' lo envía al servidor
- io.emit('serialDataJuan', data.trim()); emite mensaje al cliente con id 'serialDataJuan'

4. Resultado de la ejecución:



Sensores (potenciometro)

141

```
Terminal - juan@jamc: ~/Schreibti
Datei Bearbeiten Ansicht Terminal Reiter Hilfe
juan@jamc: ~/Schr... X juan@jamc: ~/Schr... X
juan@jamc:~/Schreibtisch/CursoB2Digital/PractGuiadaExeLearning/p3PotAWeb$ node app.js
Server listening at http://localhost:3000
A user connected from IP: ::1
```

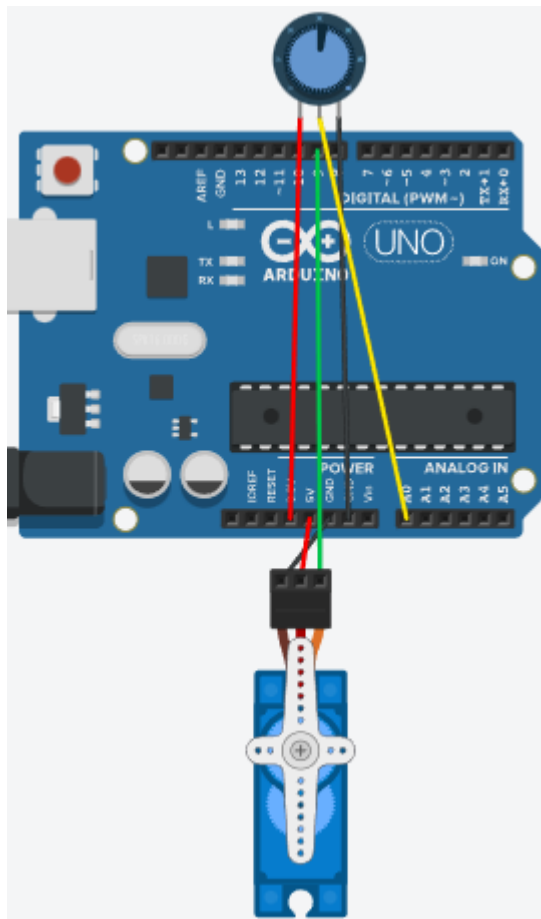

6. Control de un sensor y un actuador en RT (1h, en parejas)

T6. Actuador y sensor

Duración: 1h

Agrupamiento: en parejas

Tarea: junta los dos códigos anteriores para crear una web que controle un servo (actuador) y un potenciómetro (sensor) como en la Fig.



Ejecución:



index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Juan A. Serial Data Viewer</title>
</head>
<body>
  <h1> Actuadores (servo) </h1>
  <input type="range" id="slider" min="0" max="180" value="90">
  <p id="sliderValue">Slider Value: 90</p>
  <script src="https://cdn.jsdelivr.net/npm/socket.io/4.1.3/socket.io.js"></script>

  <h1>Sensores (potenciometro) </h1>
  <div id="serialDataBox"></div>
  <script src="https://cdn.socket.io/4.1.3/socket.io.min.js"></script>

  <script src="sketch.js"> </script>
</body>
</html>
```

sketch.js:

```

//Juan A. Morales
const socket = io();
const serialDataBox = document.getElementById('serialDataBox');
//
const slider = document.getElementById('slider');
const sliderValue = document.getElementById('sliderValue');
|
/*Slider servomotor*/
//sent slider value to server
slider.addEventListener('input', () => {
  const value = slider.value;
  socket.emit('sliderValueJuan', value); //sent value with Id sliderValueJuan
});

//replace the slider value number with the latest
socket.on('updateSliderJuan', (value) => { //receive server value with id updateSliderJuan
  slider.value = value;
  sliderValue.textContent = `Slider Valuei: ${value}`;
});

/*Box potentiometer*/
socket.on('serialDataJuan', (data) => {
  // Replace the content of the box with the latest data
  serialDataBox.textContent = data;
});

```

app.js:

```

//Juan A.
const { SerialPort } = require('serialport')
const serialport = new SerialPort({path: '/dev/ttyACM0', baudRate: 9600 })
const { ReadlineParser } = require('@serialport/parser-readline');
//
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
//
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
const port = 3000;
//only the files from public can be served to a client
app.use(express.static('./publicJuan'));

//WebSocket connection handling
io.on('connection', (socket) => { //server waiting for the connection client event
  //
  const userIP = socket.handshake.address; //IP del cliente
  console.log(`A user connected from IP: ${userIP}`); //message when client connected

  /*Slider servomotor */
  // Function to execute when a message with Id 'sliderValueJuan' is recieved
  socket.on('sliderValueJuan', (value) => { //receive client value
    io.emit('updateSliderJuan', value); // broadcast the updated value to all connected clients
    console.log(`Slider Values: ${value}`); //console
    serialport.write(`${value}\n`); //write value to serial port
  });

  /*Box potentiometer*/
  // Create a parser to read lines from the serial port
  const parser = serialport.pipe(new ReadlineParser({ delimiter: '\n' }));
  // Listen for incoming serial data and emit it to the connected clients using WebSocket
  parser.on('data', (data) => { //when parser receives a data (until delimiter)
    //execute the following
    io.emit('serialDataJuan', data.trim()); //broadcast message (with id=serialDataJuan)
    //to all clients connected to socket
  });
});

// Start the server
server.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});

```

7. Amplia el sistema, busca usos (1 a 3h, en parejas)

T7a. Amplia CSS

Duración: 1h

Agrupamiento: individual

Tarea: dale estilo a tu web con CSS. Ej. añadiendo un fondo gif y que el valor del potenciómetro se refleje en un círculo que cambia de tamaño. Mira este video:

https://www.youtube.com/embed/PltzUs3gk_0

Tarea: reflexiona distintas aplicaciones (que mejoren el entorno social) de tu sistema si lo ampliaras con más actuadores y sensores. Esto te servirá para orientar la siguiente tarea.

T7b. Amplia elementos

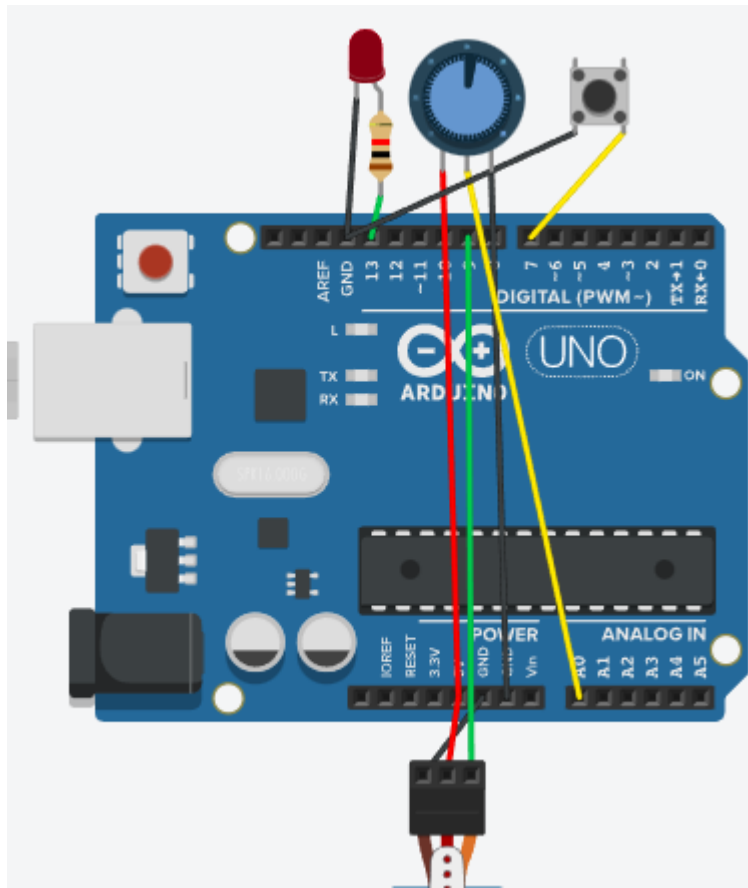
Duración: 2h

Agrupamiento: en parejas o individual

+ **Placa de pruebas**

Tarea-voluntaria: sirviendote de una protoboard añade al menos un sensor más (ej. botón al 7) y un actuador más (ej. LED al 13). Hazlo en función de la utilidad que le vas a dar a tu sistema IoT.

Circuito: ejemplo de ampliación



Código `ampliaActSens.ino`: donde al leer el botón le damos la vuelta a los bits, enviamos por puerto serie una cadena de tamaño fijo (añadiendo ceros con `%03d`) del estado de todos los sensores (ej. `Po231Bu1`) y recibimos por serie una cadena de tamaño fijo del estado deseado de todos los actuadores (ej. `c10c21s1175`)

```

1  #include <Servo.h>
2
3  const int ledRPin = 13; const int ledGPin = 12;
4  const int potPin = A0; const int servPin=9;
5  const int butPin = 7;
6  Servo myServo; // Create servo object
7  int potVal; int servPos; int butState; int ledRVal;
8  char s[8]; //sensor string to emit ex. Po%03dBu%d
9  char a[12]; //a="c10c21s1175"; string to receive
10
11 void setup()
12 {
13   pinMode(ledRPin, OUTPUT); pinMode(ledGPin, OUTPUT);
14   pinMode(butPin, INPUT_PULLUP); myServo.attach(servPin);
15   Serial.begin(9600); myServo.write(90); // Calibrate Servo
16 }
17
18 void loop()
19 {
20   /*Sensors*/
21   potVal=analogRead(potPin); potVal=map(potVal,0,1023,0,255); //for 5V
22   //potVal=map(potVal,0,690,0,255); //for 3.5V
23   //Sensor Button: we invert digitalRead(butPin) is 1 (not pressed) or 0 (pressed)
24   butState = 1-digitalRead(butPin);
25   sprintf(s, "Po%03dBu%d", potVal, butState);
26   Serial.println(s); //Emit sensor string
27
28   /*Actuators*/
29   if(Serial.available()) {
30     Serial.readBytes(a,12);
31     ledRVal = a[2]-48; //a[2]-'0'; convert to Int
32     //servPos = (a[8]-'0')*100+(a[9]-'0')*10+(a[10]-'0');
33     //servPos = (a[8]*100+a[9]*10+a[10]*1)-'0'*(100+10+1);
34     servPos = (a[8]*100+a[9]*10+a[10])-5328;
35     if(servPos >= 5 && servPos < 175) {myServo.write(servPos);}
36     if(ledRVal==1) { digitalWrite(ledRPin, HIGH);}
37     else {digitalWrite(ledRPin, LOW); }
38   }
39 }

```

index.html:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <title>Juan A. Serial Data Viewer</title>
5  </head>
6
7  <body>
8    <h1> Juan Andrés </h1>
9
10   <h1> Actuadores (servomotor, luz) </h1>
11   <p id="upVal"> Cadena valores de actuadores (re-actual): xxxxxxxxxx </p>
12   <p> Servo: </p>
13   <input type="range" id="slider" min="0" max="180" value="90">
14   <p> Luces: </p>
15   <input type="checkbox" id="c1"> rojo </input>
16   <input type="checkbox" id="c2"> verde </input>
17   <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.1.3/socket.io.js"></script>
18
19
20   <h1>Sensores (potenciometro, botón) </h1>
21   <p id="serialDataBox"> Cadena valores de sensores (actual): xxxxxxxx </p>
22   <p id="pot"> Potenciometro: xxx </p>
23   <input type="checkbox" id="c3"> botón </input>
24   <script src="https://cdn.socket.io/4.1.3/socket.io.min.js"></script>
25   <script src="sketch.js"> </script>
26
27 </body>
28 </html>

```

sketch.js:

```

1 //Juan A. Morales
2 const socket = io();
3 const serialDataBox = document.getElementById('serialDataBox');
4 const s1 = document.getElementById('slider');
5 const upVal = document.getElementById('upVal');
6 const c1 = document.getElementById('c1');
7 const c2 = document.getElementById('c2');
8 const c3 = document.getElementById('c3');
9 const p1 = document.getElementById('pot');
10 var eVal, s1ValPad;
11 /*Actuators: Slider and checker*/
12 function eFun(){
13     //emited Values (+ c1.checked) convert boolean to 0/1
14     s1ValPad=s1.value.toString().padStart(3, "0");
15     eVal = 'b1' + (+ c1.checked) + 'b2' + (+ c2.checked) + 's1' + s1ValPad;
16     socket.emit('valuesJuan', eVal); //sent value with Id sliderValueJuan
17 };
18 //sent slider value to server
19 c1.addEventListener('change', eFun);
20 c2.addEventListener('change', eFun);
21 s1.addEventListener('input', eFun);
22 //replace the actuators values with the one reemited by the server
23 socket.on('updatedValuesJuan', (val) => { //receive server value with id updateSliderJuan
24     s1.value=val.substring(8, 11);
25     upVal.textContent = `Cadena valores de actuadores (re-actual): ${val}`;
26 });
27
28 /*Sensors: Box potentiometer and button*/
29 socket.on('serialDataJuan', (data) => {
30     // Replace the content of the box with the latest data
31     serialDataBox.textContent = `Cadena valores de sensores (actual): ${data}`; //Po255Bu1
32     p1.textContent = `Potenciometro: ${data.substring(2,5)}`;
33     c3.checked=!!+data.substring(7,8); //convert string '0' or '1' to false or true
34 });

```

app.js:

```

1 //Juan A.
2 const { SerialPort } = require('serialport')
3 const serialport = new SerialPort({path: '/dev/ttyACM0', baudRate: 9600 })
4 const { ReadlineParser } = require('@serialport/parser-readline');
5 const express = require('express'); const http = require('http');
6 const socketIo = require('socket.io');
7 const app = express(); const server = http.createServer(app);
8 const io = socketIo(server); const port = 3000;
9 //only the files from public can be served to a client
10 app.use(express.static('./publicJuan'));
11
12 //WebSocket connection handling
13 io.on('connection', (socket) => { //server waiting for the connection client event
14     const userIP = socket.handshake.address; //IP del cliente
15     console.log(`A user connected from IP: ${userIP}`); //message when client connected
16     /*Actuators: slider servomotor */
17     // Function to execute when a meassage with Id 'ValuesJuan' is recieved
18     socket.on('valuesJuan', (rVal) => { //receive client value
19         io.emit('updatedValuesJuan', rVal); // re-broadcast the updated value to all connected clients
20         console.log(`Sent to serial: ${rVal}`); //console
21         serialport.write(`${rVal}\n`); //write value to serial port
22     });
23     /*Sensors: box potentiometer*/
24     // Create a parser to read lines from the serial port
25     const parser = serialport.pipe(new ReadlineParser({ delimiter: '\n' }));
26     // Listen for incoming serial data and emit it to the connected clients using WebSocket
27     parser.on('data', (data) => { //when parser receives a data (until delimiter) execute that:
28         io.emit('serialDataJuan', data.trim()); //broadcast message (with id=serialDataJuan)
29     }); //to all clients connected to socket
30 });
31
32 // Start the server
33 server.listen(port, () => {
34     console.log(`Server listening at http://localhost:${port}`);
35 });

```


Ejecución: observa la cadena del estado deseado de los actuadores re-actual (enviada por el cliente al servidor y devuelta al cliente ej. b11b20s1139) y la cadena de los sensores actual (enviada por el servidor ej. Po067Bu0)

The screenshot shows a web browser window titled 'Juan A. Serial Data Viewer' at the URL 'localhost:3000'. The page displays the name 'Juan Andrés' and a section for 'Actuadores (servomotor, l...'. It shows the 'Cadena valores de actuadores (re-actual): b11b20s1139'. Below this, there is a 'Servo:' control with a slider and 'Luces:' control with checkboxes for 'rojo' (checked) and 'verde'. Another section for 'Sensores (potenciometro, b...' shows the 'Cadena valores de sensores (actual): Po067Bu0' and a 'Potenciometro: 067' value with a 'botón' checkbox.

Overlaid on the right is a terminal window titled 'Terminal - juan@jamc: ~/Schreibtisch'. It shows the command 'node app.js' being executed, followed by the server listening at 'http://localhost:3000'. A user connects from IP '::1', and the server sends a series of commands to the serial port: 'b10b20s1123', 'b11b20s1123', 'b11b20s1127', 'b11b20s1129', 'b11b20s1131', 'b11b20s1134', 'b11b20s1137', and 'b11b20s1139'.

+ Agrupación de datos de estado en cadena de texto

Tarea: reflexiona, considerando la escalabilidad, las ventajas y desventajas de enviar desde el cliente los estados deseados de todos los actuadores en una sola cadena de texto. Haz lo mismo con la cadena de sensores. Discute otras alternativas. ¿Crees que los seres vivos hacen algo similar para transmitir el sentido del tacto? ¿Qué tiene que ver esto con el procesamiento paralelo y las GPUs?

8. Defensa y evaluación del producto final (1h)

T8a Defensa del Blog

Duración: 5 min

Agrupamiento: en parejas

Tarea: durante el proyecto deberíais de haber realizado una entrada WordPress similar a [esta](#) del profesor y que se muestra en la Fig. de abajo. Debe incluir:

- Capturas firmadas y resúmenes de las ideas más importantes aprendidas.
- Una reflexión sobre la utilidad y el impacto de tu sistema IoT en la sociedad.
- Opcionalmente: un vídeo corto o un GIF de tu producto final funcionando.

El profesor os hará una entrevista donde tendrás que defender dicha entrada de tu Blog. La mejor propuesta será presentada al resto del grupo.

The screenshot shows a WordPress blog post on the domain `juancordovillaesjg.wordpress.com`. The post title is "Proyecto: Control en tiempo real de un robot IoT via web (para 2ºBach. TIC y 2ºSMR AW)" and it was published on "diciembre 13th, 2023". The author is "JUAN ANDRÉS MORALES CORDOVILLA".

The main content of the post is a diagram titled "Tareas del proyecto 'Control en tiempo real de un robot IoT via web'". The diagram illustrates the system architecture and workflow:

- 1. Nodo.js y "bota" servidor a cliente (1h, individual):** Capturar y procesar los datos de los sensores y actuadores.
- 2. Servidor de y terminal a servidor (1h, parvicio):** Capturar y procesar los datos de los sensores y actuadores.
- 3. Cliente a terminal de servidor (1h, individual):** Capturar y procesar los datos de los sensores y actuadores.
- 4. Un actuador (1h, parvicio):** Capturar y procesar los datos de los sensores y actuadores.
- 5. Un sensor (1h, parvicio):** Capturar y procesar los datos de los sensores y actuadores.
- 6. Un actuador y un sensor (1h, parvicio):** Capturar y procesar los datos de los sensores y actuadores.
- 7. Ampliación CS3 (1h, parvicio):** Ampliación de contenido.
- 8. Producto final (1h):** Definición de producto final.

The diagram also includes a flowchart showing the data flow between the server, terminal, and robot components.

T8b. Test final


Duración: 55 min


Agrupamiento: individual

Tarea: evalúate y evalúa al profesor mediante el siguiente [Test Autocorregible](#)

Examen Test del Proyecto IoT Web RT

Prof. Juan A. M. Cordovilla

jmorcor755@g.educaand.es [Cambiar de cuenta](#) 

 No compartido

* Indica que la pregunta es obligatoria

Escriba su nombre, apellidos y grupo (ej. Juan A. Morales Cordovilla, 2º Bach. A) *

Tu respuesta

Marque la palabra correcta

`app.listen` `getElementById` `/dev/ttyACM0` `app.js` `addEventListener`

La carpeta con el index.html suele estar en misma carpeta que el ...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
... es el archivo dispositivo del USB	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
.... modifica el objeto HTML slider Value	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Conclusión y créditos

Resumen y trabajo futuro (orientado al profesor)

Este proyecto o práctica guiada ha pretendido enseñar a crear un sistema para controlar remotamente, en RT y mediante webs un dispositivo robótico simple. La importancia para el sistema educativo (competencias que trabaja, metodología o evaluación) se ha descrito en la Guía Didáctica del comienzo. Destacamos la [partición en subtareas](#) para alcanzar el producto final: sistema con varios actuadores y sensores pensado para algo útil.

Se ha optado por dar las soluciones de los códigos para avanzar más rápido, si bien a costa de perder creatividad del alumnado. Sin embargo creemos que lo importante es que el alumno comprenda eficazmente ciertos conceptos básicos como:

- modularización del código (index.html, sketch.js y app.js)
- socket e Id de mensaje en Node.js
- control duplex de un actuador y un sensor
- agrupación de datos de estado en cadena de texto

y que use este proyecto como referente para crear futuros sistemas que contribuyan al bien de la sociedad. El futuro del control robótico vía web puede ser muy interesante si empezamos a comunicar unas webs con otras (ej. un ChatGPT que controla una web de IoT de regadío de una finca).

Créditos

Información general sobre este recurso educativo

Título	Control en tiempo real de un robot IoT via web
Descripción	Control en tiempo real de un robot IoT via web usando Node.js. Para 2º de Bachillerato asignaturas de Programación y Computación y

	Tecnologías de la Información y la Comunicación II o para el ciclo de FP 2º de SMR asignatura de Aplicaciones Webs
Autoría	Juan Andrés Morales Cordovilla
Licencia	Creative Commons BY-SA 4.0

Este contenido fue creado con [eXeLearning](https://exelarning.com/), el editor libre y de fuente abierta diseñado para crear recursos educativos para el curso "*Competencia Digital Docente Nivel B2*" de la Junta de Andalucía (España) en Enero de 2024.

Descargar el fichero .elp

Obra publicada con [Licencia Creative Commons Reconocimiento Compartir igual 4.0](#)