

Sistema domótico

Índice

[Índice](#)

[Ejercicio 1](#)

[Ejemplo calculadora](#)

[Ejemplo calculadora web](#)

[Ejemplo conexiones](#)

[Ejemplo mongo](#)

[Ejercicio 2 - Sistema domótico](#)

[Descripción](#)

[Arquitectura](#)

[Diseño](#)

[Comunicación en tiempo real](#)

[Instalación](#)

[Ejecución](#)

[Ejemplo de ejecución](#)

[Diagramas](#)

[Diagrama de arquitectura](#)

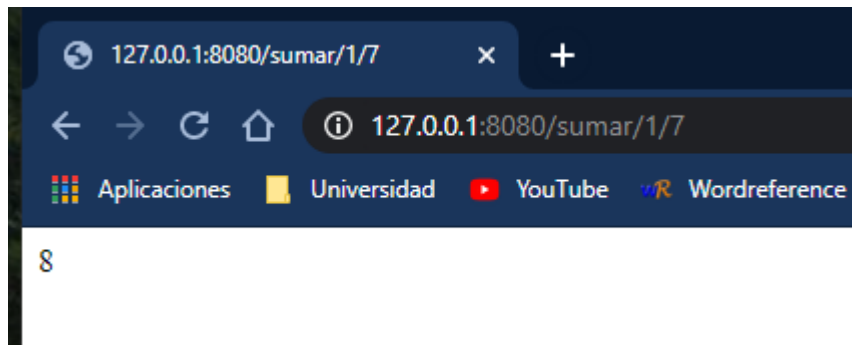
[Diagrama de clases](#)

Ejercicio 1

Ejemplo calculadora

En este ejemplo se crea un servidor web que escucha peticiones en el puerto 8080. Procesará las peticiones que le llegan para devolver una respuesta http con código 200 y el resultado del procesamiento.

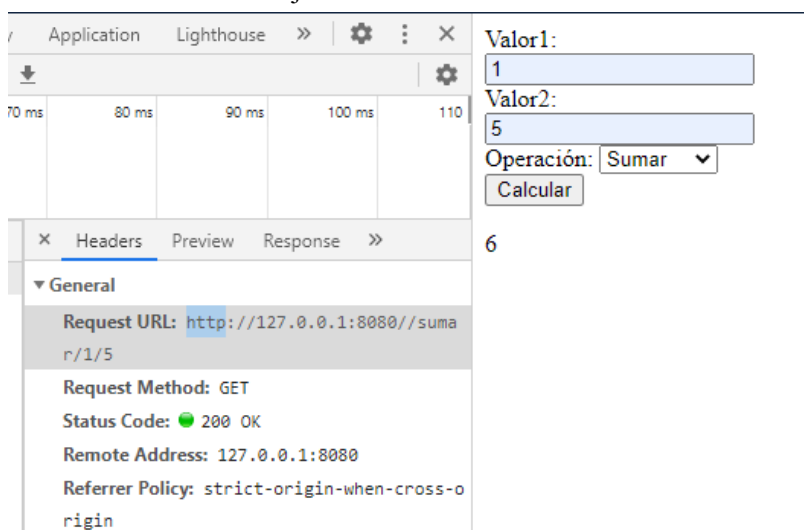
> node calculadora.js



Ejemplo calculadora web

Este ejemplo realiza lo mismo que el anterior, sólo que además manda al cliente un archivo html para que pueda hacer los cálculos de una forma más cómoda. Cuando el cliente pincha en el botón de calcular, forma una petición para comunicarse con el servidor. Esta petición es igual que la que hacíamos en el ejemplo anterior.

> node calculadora-web.js



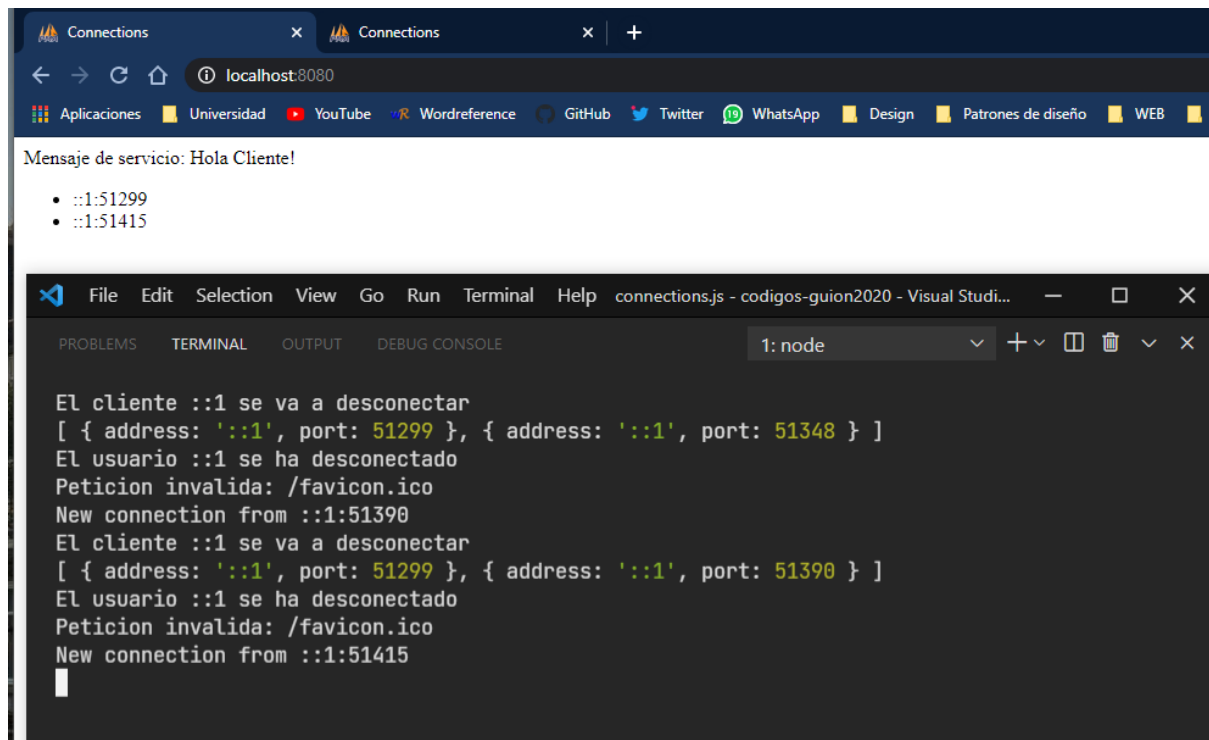
Ejemplo conexiones

Se crea un servidor web que manda una página a los clientes con la información del servicio y los clientes conectados (connections.html).

Cuando el cliente abre el archivo html, se ejecuta un script que crea una nueva conexión con un socket al servidor, que ha iniciado también un servidor de sockets.

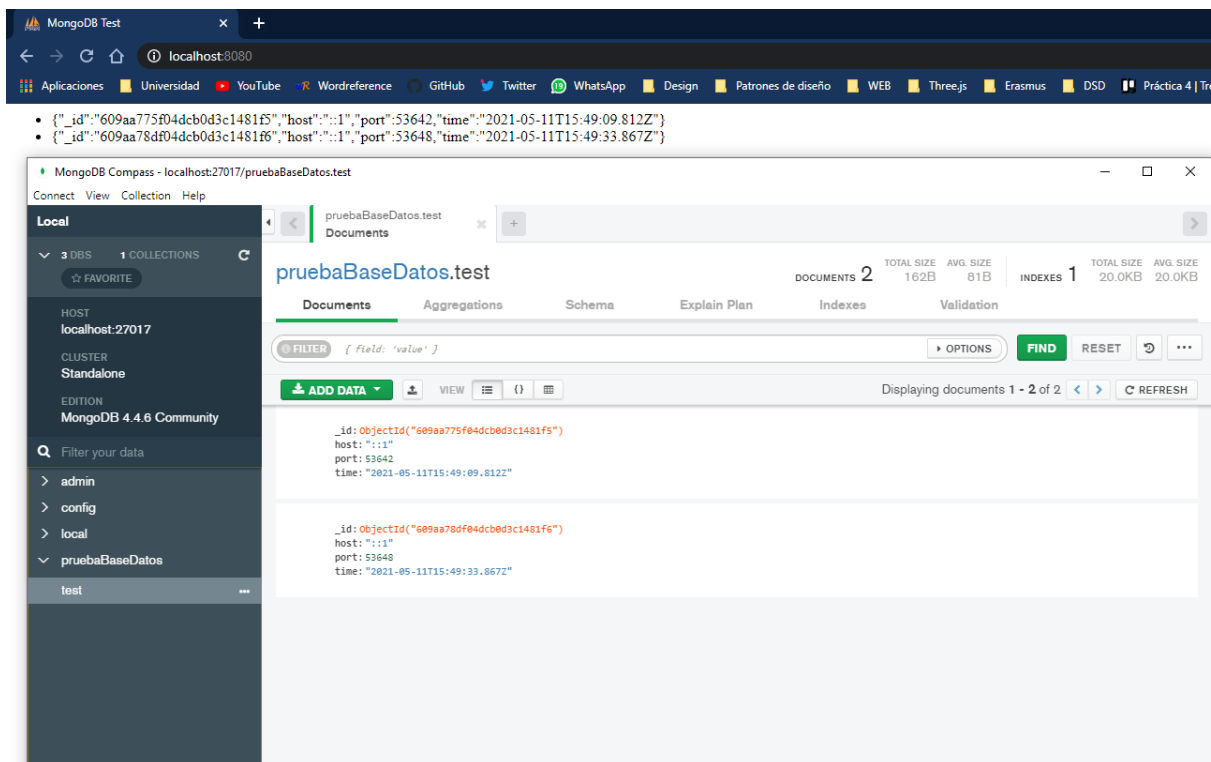
Según que evento se produzca, se actualiza la lista de usuarios, se manda un mensaje de bienvenida o les dice que el servidor ha dejado de funcionar. Estos eventos son emitidos por el servidor de conexiones.

> node connections.js



Ejemplo mongo

Este ejemplo hace lo mismo que el ejemplo anterior, sólo que los datos son guardados en una base de datos. Podemos ver en la captura las colecciones de la bbdd tras conectar dos usuarios, gracias a mongo compass.



> node mongo_test.js

Ejercicio 2 - Sistema domótico

Enlace al repositorio: https://github.com/JesusGonzalezA/IOT_System_Simulator

Descripción

Se pide desarrollar un sistema domótico que permita a los usuarios llevar el control de su casa. Unos sensores captarán información de la temperatura y la luminosidad de la estancia (en este caso a través de unos formularios). El usuario podrá cambiar estos valores, así como activar/desactivar el aire acondicionado o el motor de las persianas.

Cuando se dan ciertas condiciones, como una temperatura que excede del límite, un agente realiza cambios en el estado de los actuadores (activar el aire y cerrar las persianas) y notificará a todos los usuarios conectados al sistema.

Se permitirá al usuario ver el historial de medidas tomadas y cambios de estado de los actuadores.

Además, se ofrece una pantalla a modo de resumen con la que podrá ver el estado de todo su sistema.

Cabe destacar que todo el sistema funciona en tiempo real, gracias a una implementación basada en sockets, y que los datos del sistema son consistentes debido a que se utilizan bases de datos en el lado del servidor, mongodb.

Arquitectura

El proyecto sigue una arquitectura cliente servidor, como se puede mostrar en el [siguiente diagrama](#).

El servidor se compone de tres servicios fundamentales:

- Http: se encarga de enviar al cliente los archivos necesarios para que la interfaz de usuario pueda ser visualizada y de responder ante los formularios.
- Socket: se encarga de mantener el funcionamiento del sistema en tiempo real. Se responde a los clientes ante cualquier evento de actualización, para que puedan actualizar los datos en las interfaces de usuario. Además, permite que el agente transmita las alertas a los clientes.
- Agente: se encarga de comprobar los valores introducidos en la base de datos y actualizar los actuadores si la situación del sistema es crítica.
- DB: se encarga de leer y escribir la información que mantiene el sistema en la base de datos.

Diseño

Puede ver el diagrama de clases [aquí](#).

Las comunicaciones del cliente son resueltas por el servidor HTTP o el servidor de Socket.io, que tiene una referencia al primero. El cliente realiza una petición http a la dirección del servidor. Este la resuelve:

- Primero se entra a la función route. Esta actúa como un router. Se encarga de, dada una petición en una url x, realizar la operación correspondiente. Gracias a esto, se puede conseguir utilizar urls limpias.
 - El cliente realiza una petición en la página host:port/resumen

- El servidor detecta que la url es /resumen. Por tanto, envía la página html resumen.html, que contiene referencias a archivos javascript y css que se interpretan en el navegador del cliente.
 - Se lee del sistema de archivos resumen.html y se envía al cliente.
- El cliente recibe el archivo resumen.html. Este contiene referencias al icono, javascript y css, que pide al servidor.
 - El servidor lee los archivos solicitados que se encuentran en la carpeta public y se los envía al cliente.

Puede ser que el cliente ya haya recibido la página y quiera comunicarse con el servidor. Esto puede ser, a través de un formulario, realizando una petición POST, o a través de una petición GET para pedir los datos de la base de datos.

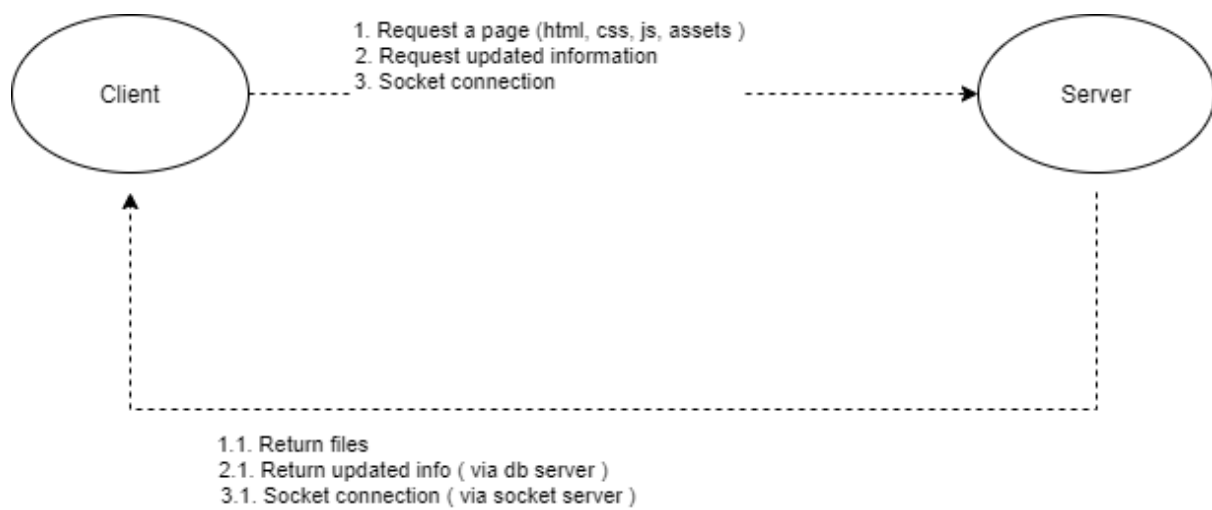
- El cliente envía la petición al servidor
- El servidor, en la función route, detecta que la petición no es una ruta ni un archivo css, javascript, css o ico, sino que es una acción. Según la ruta y el verbo HTTP, el cliente ejecuta un script diferente, que puede hacer referencia al servidor de bases de datos.
 - En caso de que se pida/envíe información al servidor de bases de datos, se espera a que la operación sea ejecutada y se envía al cliente (promise).

Con esto, ya tenemos un sistema cliente-servidor completamente funcional. El problema es que si el sistema es actualizado, el cliente debe de recargar la página para realizar las peticiones get para tener las interfaces con la información actualizada. Para evitar este problema se incluye un servidor de sockets que permite la comunicación en tiempo real.

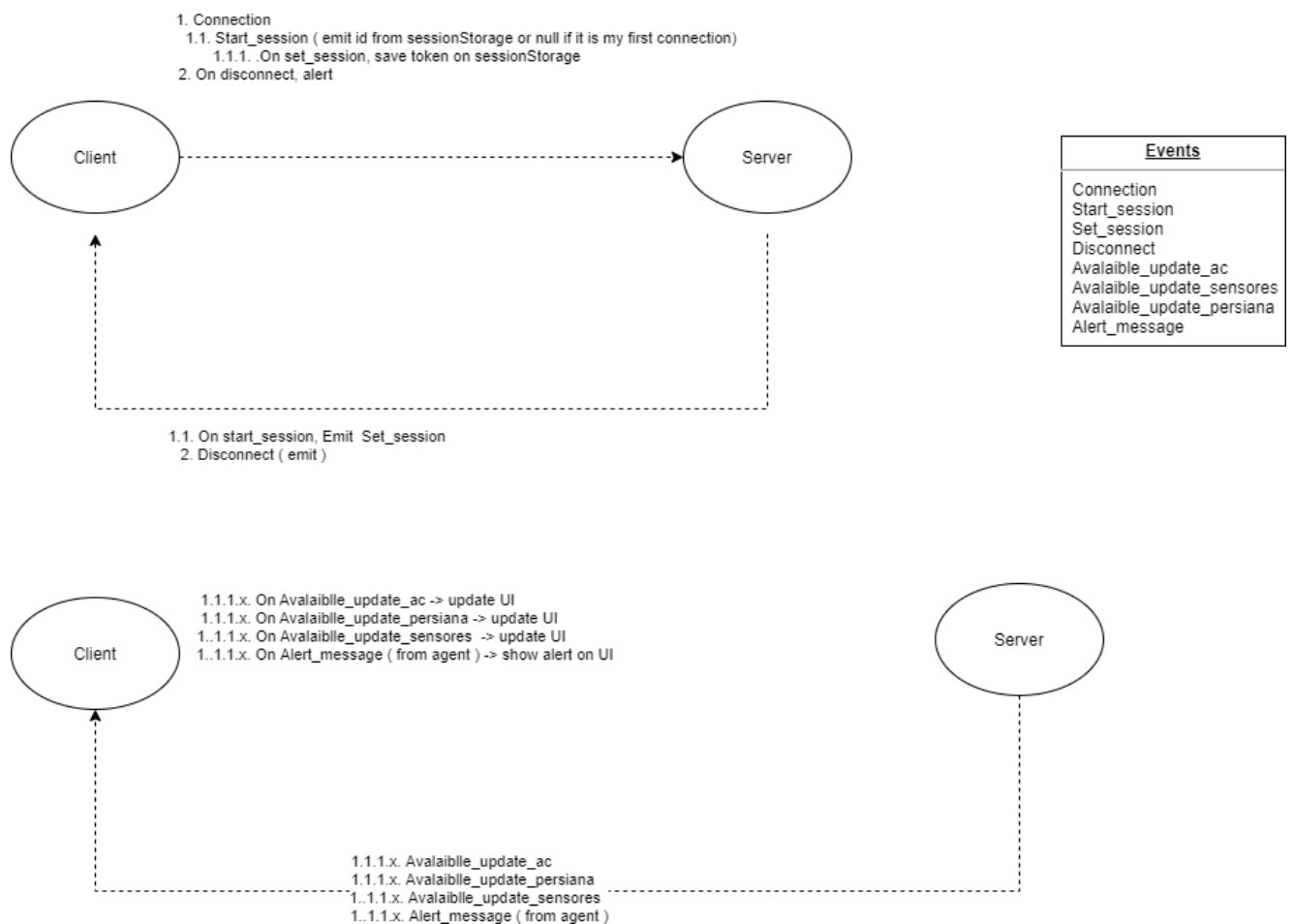
Comunicación en tiempo real

Como el cliente tiene varias páginas y no se utiliza un framework o librería como React, Angular... que nos permita trabajar con un frontend SPA (Single Page Application), se conectaría y desconectaría un socket cada vez que se cambia de página. Para evitar esto, se ha realizado un sistema de control de sesiones para que el usuario pueda recuperar el socket. Esto se consigue en la parte del cliente gracias a la utilización de SessionStorage. Por su parte, en el servidor se crea un identificador uuid versión 4 gracias a un paquete de node. Realmente, debido a lo básico que es el sistema no haría falta aún, pero para versiones futuras podríamos implementar un sistema protegido y crear salas de sockets gracias a esto. Imaginemos que tenemos dos tipos de clientes: administradores y básicos. Quizás información como la temperatura pueda ser mandada a ambos, pero información como en una tienda, dinero disponible, control de acceso de los empleados, etc debería de ser mandada sólo a clientes con cierto privilegio. Para ello podríamos crear salas de administradores incluyendo sus identificadores y hacer un multicast a los mismos de esta información, y con la información básica hacer simples broadcast. Además, se descongestiona el servidor de esta forma.

Comunicación básica entre el cliente y servidor:



Actualizaciones en tiempo real. En caso de que las comunicaciones vengan del agente, se muestra una alerta por pantalla al cliente. En cualquier caso se actualiza la interfaz de usuario.



Instalación

Para una correcta instalación, se necesita conexión a internet, node js y npm instalados en el sistema.

> npm install

Ejecución

Si desea realizar cambios sobre el código y ver su actualización en tiempo real:

> npm run dev

Si desea probar el sistema

> npm run start

El cliente se servirá en <http://localhost:8080>

Un servicio de mongodb deberá estar corriendo en segundo plano en su ordenador. Se creará una base de datos con nombre “iot”, que utilizará el sistema domótico.

Ejemplo de ejecución

Puede ver aquí un ejemplo de ejecución con tres clientes: <https://youtu.be/TF9Tz7XHR9s>

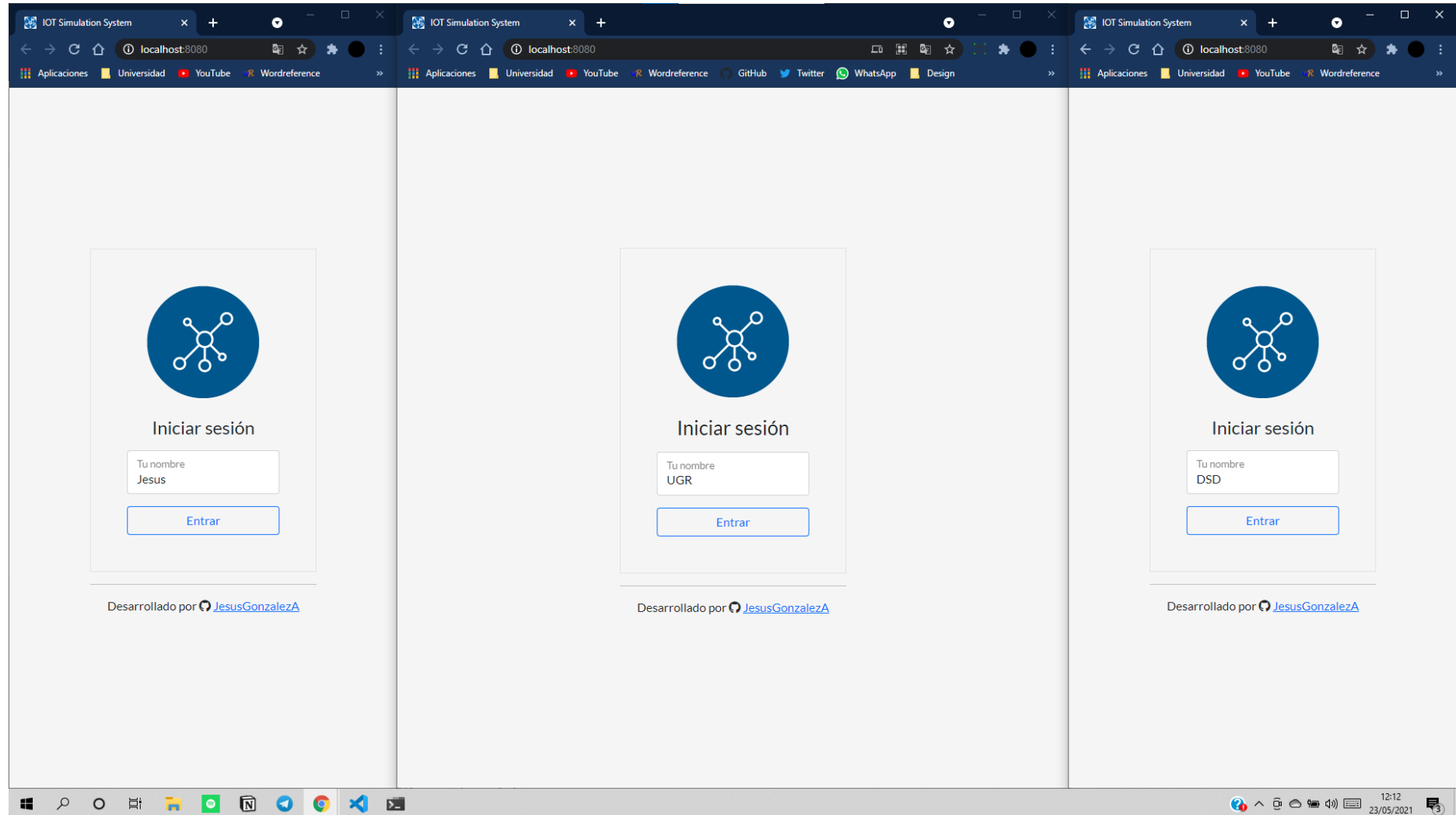
> npm run start

```
> node src/index.js

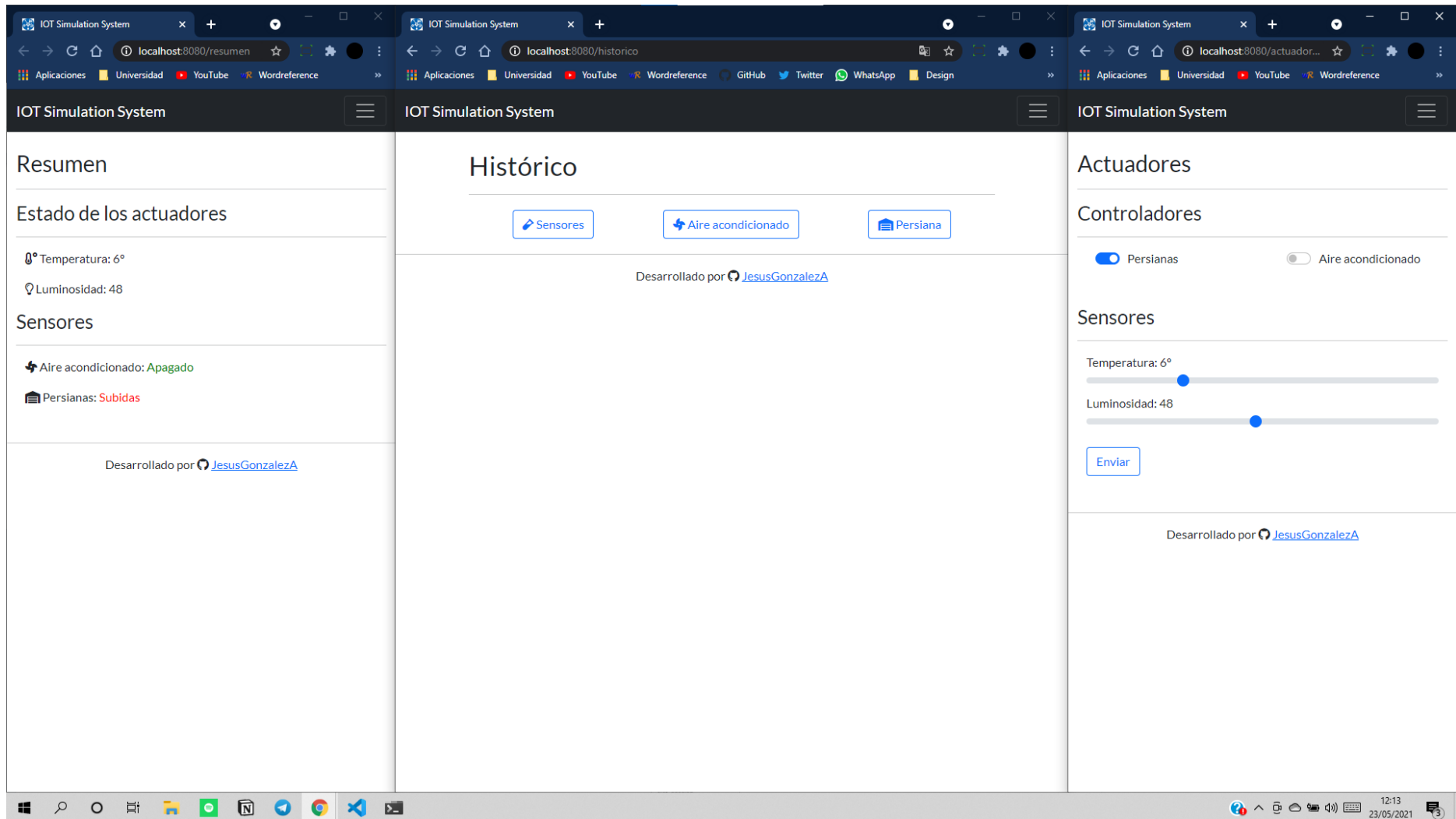
-----
|                                     |
|  Welcome to your IOT Simulation System  |
|                                     |
|-----|

Server is ready:  http://localhost:8080
Socketio is ready
MongoClient is ready
```

Iniciamos sesión con tres clientes



A la izda, el resumen del sistema. En medio, el histórico. Los tres botones permiten mostrar/ocultar la información de la base de datos. A la derecha, encontramos un formulario para actualizar los sensores. Si cambiamos los datos de los controladores se envían al servidor individualmente.



Resumen

IOT Simulation System Resumen Actuadores Histórico

Casa de Jesus

Resumen

Estado de los actuadores


🌡 Temperatura: 19°

💡 Luminosidad: 46

Sensores

✚ Aire acondicionado: **Apagado**

🏠 Persianas: **Bajadas**

Desarrollado por  [JesusGonzalezA](#)

Actuadores

IOT Simulation System

ResumenActuadoresHistórico

Casa de Jesus

Actuadores

Controladores

☐ Persianas

☐ Aire acondicionado

Sensores

Temperatura: 19°

Luminosidad: 46

Enviar

Histórico

IOT Simulation System

ResumenActuadoresHistórico

Casa de Jesus

Sensores

Aire acondicionado

Persiana

Fecha	Sensores	Valores
23/5/2021, 12:30:24	Luminosidad Temperatura	46 19°
23/5/2021, 12:30:13	Luminosidad Temperatura	94 -5°
23/5/2021, 12:30:08	Luminosidad Temperatura	94 -5°
23/5/2021, 12:30:00	Luminosidad Temperatura	94 35°
23/5/2021, 12:29:52	Luminosidad Temperatura	94 35°
23/5/2021, 12:29:46	Luminosidad Temperatura	94 35°
23/5/2021, 12:29:34	Luminosidad Temperatura	94 35°
23/5/2021, 12:29:05	Luminosidad Temperatura	47 35°
23/5/2021, 12:28:51	Luminosidad Temperatura	47 18°
23/5/2021, 12:28:12	Luminosidad Temperatura	47 12°
23/5/2021, 12:27:58	Luminosidad	47

localhost:8080/historico#tableSensores

Formulario

IOT Simulation System

Resumen

Actuadores

Histórico

Casa de Jesús

Actuadores

Controladores

☒ Persianas


☐ Aire acondicionado

Sensores

Temperatura: 19°

Luminosidad: 46

Enviar



Éxito

La acción se realizó correctamente

OK

Alerta del agente

IOT Simulation System

Resumen

Actuadores

Histórico

Casa de Jesus

Actuadores

Controladores

Persianas

Aire acondicionado

Sensores

Temperatura: 42°

Luminosidad: 46

Enviar

i

Alerta del agente

Temperatura superior al máximo. Enciendo el aire.
Temperatura superior al máximo. Cierro las persianas.

OK

Diagramas

Diagrama de arquitectura

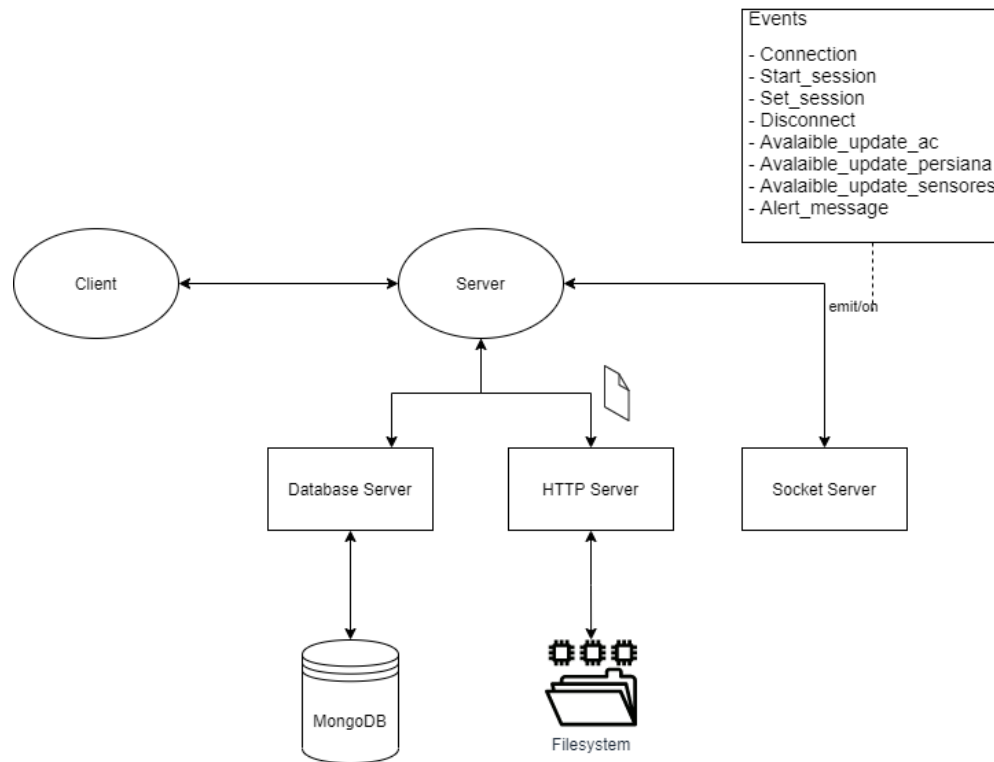


Diagrama de clases

