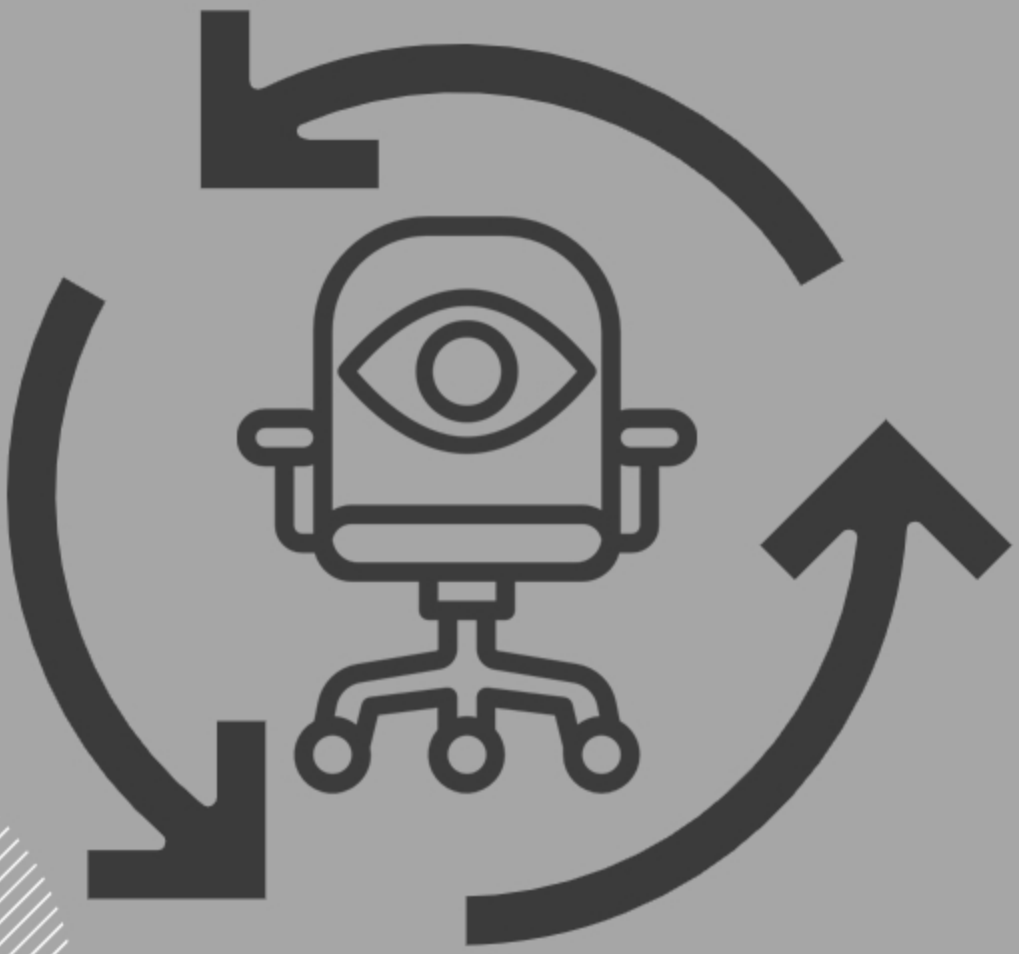
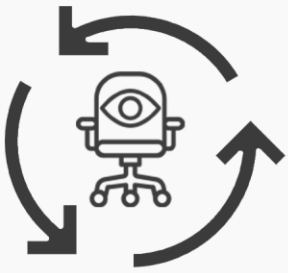


<<Chair Tracker>>

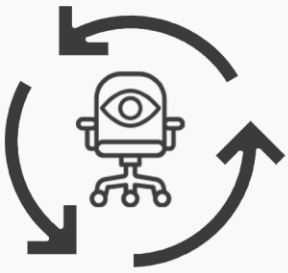
José Arciniega Salvatierra
José Antonio García Martagón

Asignatura: DAD
Curso: 2020/21
Versión: Tercer entregable

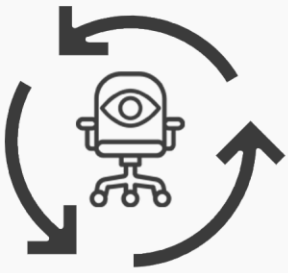


Chair Tracker

Índice



Chair Tracker



1. Introducción

1.1 Contexto

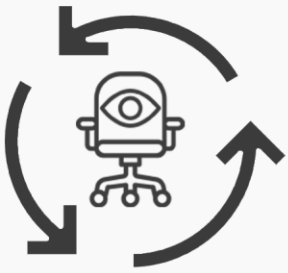
Cada vez son más las personas que pasan gran parte de su tiempo sentados frente al ordenador ya sea por trabajo u ocio.

A esto debemos de sumar el factor de la pandemia que nos ha acontecido durante el último año y que ha tenido como consecuencia un incremento en los deberes que han decidido deslocalizarse optando por un modelo telemático. Ejemplo de ello han sido ciertos sectores del mercado laboral, educación, actividades extracurriculares, etc.

Además, son varios los estudios que abalan los efectos negativos que produce en nuestro organismo el sedentarismo y la mala higiene postural derivada de este mismo.

Es por esto que ahora más que nunca debemos de ser conscientes de las horas que pasamos sentados y realizar ciclos alternando periodos de tiempo de trabajo en la silla y descanso en una posición erguida. Precisamente este es el objetivo principal de nuestro dispositivo.

Chair Tracker



Chair Tracker

1.2 Funcionalidades

1.2.1 Detección de presencia

Mediante un sensor de proximidad, podremos detectar si el usuario está sentado o no en la silla. Esta información será comunicada al servidor y el usuario podrá consultar unas estadísticas de los datos.

1.2.2 Vibrador

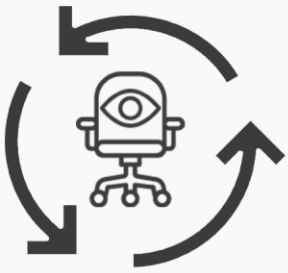
El usuario recibirá avisos mediante vibración al recibir una llamada enviada por otro usuario, de esta forma se informa de la llamada sin llegar a ser una distracción significativa como para afectar al ritmo de trabajo.

1.2.3 Alarma sonora

El usuario recibirá avisos sonoros según sea programada su alarma, esta sonará cada vez que el tiempo de trabajo se consuma avisando al usuario de que debe levantarse, de igual forma, se notificará cuando el usuario deba de sentarse al consumir su tiempo de descanso.

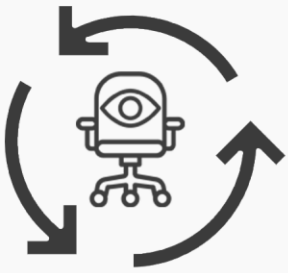
1.2.4 Web

El usuario dispondrá de una web donde revisar todas sus estadísticas, desde llamadas recibidas o enviadas hasta el tiempo de trabajo y descanso total dado un intervalo de tiempo. En caso de que el rol del usuario sea jefe, este podrá consultar las estadísticas de cada uno de sus empleados.



2. Glosario de términos

- **Usuario:** persona a la que se le asocia un sha1 de la MAC de una placa.
- **SHA-1:** función hash utilizada para encriptación de datos, en este caso, la MAC del ESP8266 que utilizamos como cliente.
- **Rol:** papel que puede tomar el usuario dentro de la empresa, distinguimos entre jefe y empleado.
- **Llamada:** aviso mediante vibración que podrá enviar un usuario a otro para notificar de cierto evento en concreto que puede especificar mediante una descripción.
- **Estado llamada:** la llamada puede presentar dos estados:
    - 'Pendiente' cuando el usuario aún no ha marcado como respondida la llamada desde la web.
    - 'Contestada' ha sido marcado el campo explicado anteriormente.
- **Alarma:** aviso sonoro que avisara al usuario de que debe levantarse o sentarse.
- **Ciclo trabajo/descanso:** número de veces que se ha completado un tiempo de trabajo o tiempo de descanso.
- **Remitente:** usuario que realiza la llamada.
- **Destinatario:** usuario que recibe la llamada.



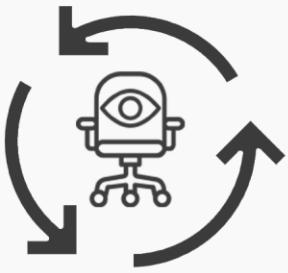
3. Catálogo de requisitos

3.1 Objetivos

RG-01: Monitorización del tiempo.
Como jefe.
Quiero saber el tiempo de trabajo y descanso de mis empleados.
Para tener un registro actualizado sobre las horas tiempo productivo en relación con el descanso para así mejorar la higiene postural de mis empleados.

RG-02: Llamadas entre dispositivos.
Como usuario.
Quiero poder contactar con otros usuarios del sistema sin necesidad de abandonar el puesto de trabajo.
Para poder notificar de diferentes eventos como la finalización de una tarea o inicio de una reunión sin interrumpir al otro usuario.

RG-03: Control de ciclos de trabajo y descanso.
Como usuario.
Quiero poder establecer los tiempos de trabajo y descanso en un ciclo.
Para poder recibir un aviso sonoro al cumplir un ciclo para no exceder los tiempos establecidos sin necesidad de gestionarlo de forma manual.



Chair Tracker

3.2 Reglas de negocio

RN-01 Evitar NIF incorrectos en el sistema.
Como administrador.
Quiero que los usuarios estén bien identificados mediante un NIF con el formato de ocho números y una letra.
Para solventar el problema de datos erróneos en el sistema.

RN-02 Número máximo de alarmas por usuario.
Como administrador.
Quiero que la cantidad máxima de alarmas establecidas para un usuario sea de cinco.
Para solventar un exceso de alarmas residuales sin validez y posibles problemas de memoria asociados.

RN-03 Evitar horario solapado de alarmas.
Como jefe.
Quiero que el horario de las alarmas esté bien establecido para que no se solapen los horarios de trabajo.
Para que no se solapen los horarios de trabajo.

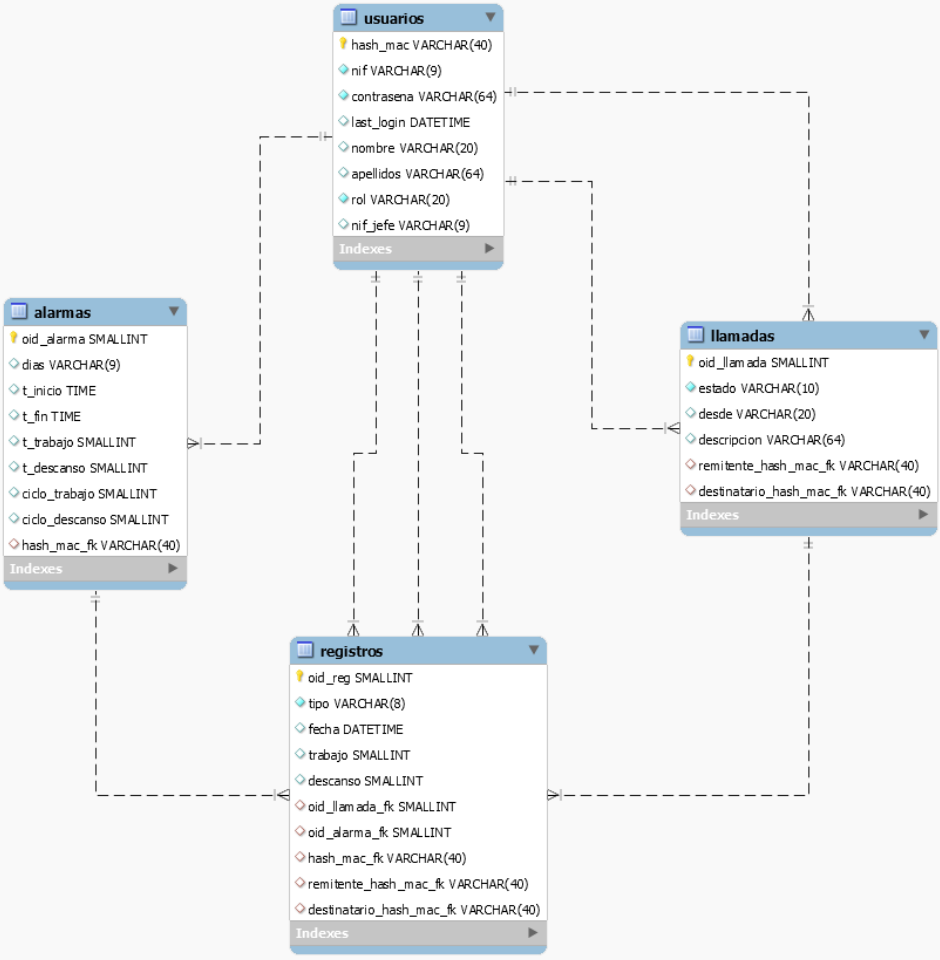
RN-04 Control de roles.
Como administrador.
Quiero que un usuario de tipo empleado tenga asociado el NIF de un jefe que se encuentre registrado en el sistema.
Para garantizar la jerarquía de la empresa dentro de nuestra aplicación.

4. Base de datos

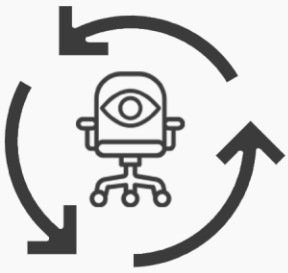
4.1 Explicación

La base de datos para nuestro sistema es bastante simple, debido a que la única información que deberemos de almacenar será la de usuarios, placas que se le asocian y sus respectivas alarmas y llamadas, que además dispondrán de registros para llevar a cabo un historial y seguimiento de estas.

4.2 UML



Chair Tracker



5. Verticles

Hemos tomado la decisión de desplegar tres verticles. Uno dedicado al servidor HTTP, otro a la conexión a la BBDD y un último dedicado a ser cliente MQTT de nuestro broker Mosquitto.

Estos tres verticles se comunicarán a través de paso de mensajes mediante el bus de eventos.

6. API REST

6.1 Métodos GET

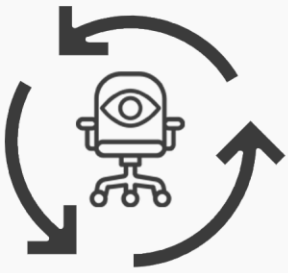
Tenemos métodos de obtención de los datos de cada tabla en su totalidad, además, hemos implementado otros métodos para obtener datos relacionados con el usuario que nos serán de utilidad a la hora de realizar peticiones desde el cliente.

Los métodos son los siguientes:

- obtenerUsuarios
  - Routing: `/api/usuarios/`
  - Entrada: json vacío
  - Salida:

```
"1":{
  "hash_mac": "valor",
  "nif": " valor",
  "contrasena": " valor ",
  "last_login": valor,
  "nombre": " valor ",
  "apellidos": " valor ",
  "rol": " valor ",
  "nif_jefe": valor
},
```

Chair Tracker



- obtenerAlarma
  - Routing: `/api/alarmas/`
  - Entrada: json vacío
  - Salida:

```
1:{
 "oid_alarma": 1,
 "dias": " valor ",
 "t_inicio": " valor ",
 "t_fin": "valor",
 "t_trabajo": valor,
 "t_descanso": valor,
 "ciclo_trabajo": valor,
 "ciclo_descanso": valor,
  "hash_mac_fk": "valor"
},
```

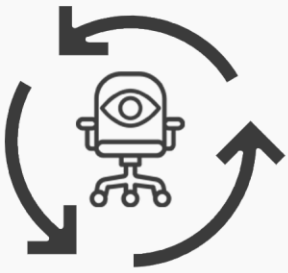
- obtenerAlarmasUsuario
  - Routing: `/api/alarmas/hash_mac`
  - Entrada:

```
{
 "hash_mac_fk": "  mac  del  usuario  a
observar "
}
```

  - Salida:

```
"1":{
 "oid_alarma": 1,
 "dias": " valor ",
 "t_inicio": " valor ",
 "t_fin": "valor",
 "t_trabajo": valor,
 "t_descanso": valor,
 "ciclo_trabajo": valor,
 "ciclo_descanso": valor,
  "hash_mac_fk": "valor"
},
```

Chair Tracker



- obtenerLlamadas
  - Routing: json vacío
    `/api/llamadas/`
  - Entrada: json vacío
  - Salida:

```
"1":{
  "oid_llamada": 1,
  "estado": "valor",
  "desde": " valor ",
  "descripcion": " valor",
  "remitente_hash_mac_fk": " valor ",
  "destinatario_hash_mac_fk": " valor "
},
```

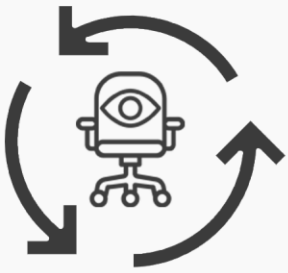
- obtenerLlamadasEnviadasUsuario
  - Routing:
    `/api/llamadas/enviadas/hash_mac`
  - Entrada:

```
{
  "remitente_hash_mac_fk": "   mac   del usuario a observar "
}
```

  - Salida:

```
"1":{
  "oid_llamada": 1,
  "estado": "valor",
  "desde": " valor ",
  "descripcion": " valor",
  "remitente_hash_mac_fk": "   mac   del usuario a observar ",
  "destinatario_hash_mac_fk": " valor "
}
```

Chair Tracker



- obtenerLlamadasRecibidasUsuario
  - Routing: `/api/llamadas/recibidas/hash_mac`
  - Entrada:

```
{
  " destinatario_hash_mac_fk": " mac del usuario a observar "
}
```

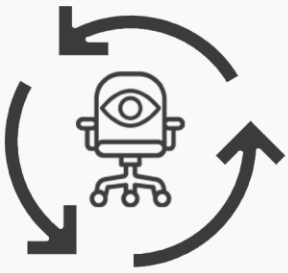
  - Salida:

```
"1":{
  "oid_llamada": 1,
  "estado": "valor",
  "desde": " valor ",
  "descripcion": " valor",
  "remitente_hash_mac_fk": " valor "
  "destinatario_hash_mac_fk": "  mac  del usuario a observar "
}
```

- obtenerRegistros
  - Routing: json vacío `/api/registros`
  - Entrada: json vacío
  - Salida:

```
"1":{"oid_reg": 1,
  "tipo": "valor",
  "fecha": "2021-05-23T21:36:52",
  "trabajo": 2910,
  "descanso": 980,
  "oid_llamada_fk": null,
  "oid_alarma_fk": 1,
  "hash_mac_fk": "mac123",
  "remitente_hash_mac_fk": null,
  "destinatario_hash_mac_fk": null
}
```

Chair Tracker



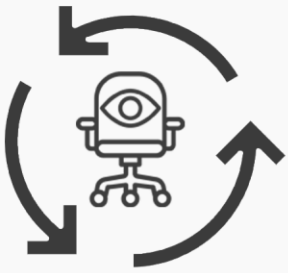
- obtenerRegistrosLlamadas
  - Routing: json vacío
    `/api/registros/llamadas`
  - Entrada: json vacío
  - Salida:


```
"4": {
    "oid_reg": 4,
    "tipo": "valor",
    "fecha": "valor",
    "trabajo": null,
    "descanso": null,
    "oid_llamada_fk": 1,
    "oid_alarma_fk": null,
    "hash_mac_fk": null,
    "remitente_hash_mac_fk": "valor",
    "destinatario_hash_mac_fk": "valor"
}
```


- obtenerRegistrosAlarmas
  - Routing: json vacío
    `/api/registros/alarmas`
  - Entrada: json vacío
  - Salida:


```
"1": {
    "oid_reg": 1,
    "tipo": "A",
    "fecha": "2021-05-23T21:36:52",
    "trabajo": 2910,
    "descanso": 980,
    "oid_llamada_fk": null,
    "oid_alarma_fk": 1,
    "hash_mac_fk": "mac123",
    "remitente_hash_mac_fk": null,
    "destinatario_hash_mac_fk": null
}
```

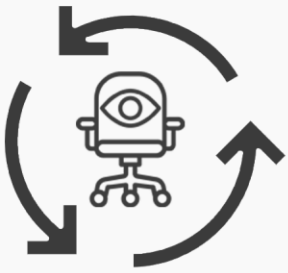

Chair Tracker



- obtenerRegistrosAlarmasUsuario
  - Routing: `/api/registros/alarmas/hash_mac`
  - Entrada:
```
{
    "hash_mac_fk": " mac del usuario a observar "
}
```
  - Salida:
```
"1": {
    "oid_reg": 1,
    "tipo": "A",
    "fecha": "2021-05-23T21:36:52",
    "trabajo": 2910,
    "descanso": 980,
    "oid_llamada_fk": null,
    "oid_alarma_fk": 1,
    "hash_mac_fk": "mac123",
    "remitente_hash_mac_fk": null,
    "destinatario_hash_mac_fk": null
}
```

- obtenerRegistrosLlamadasEnviadas
  - Routing: `/api/registros/llamadas/enviadas/hash_mac`
  - Entrada:
```
{
    "remitente_hash_mac_fk": " mac del usuario a observar "
}
```
  - Salida:
```
"4": {
    "oid_reg": 4,
    "tipo": "L",
    "fecha": "valor",
    "trabajo": null,
    "descanso": null,
    "oid_llamada_fk": 1,
    "oid_alarma_fk": null,
    "hash_mac_fk": null,
    "remitente_hash_mac_fk": "  mac  del usuario a observar ",
    "destinatario_hash_mac_fk": "valor
}
```

Chair Tracker



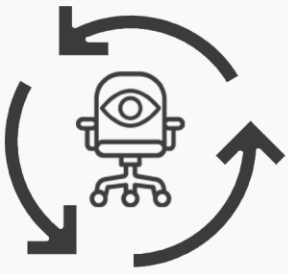
- obtenerRegistrosLlamadasRecibidas
  - Routing: `/api/registros/llamadas/recibidas/hash_mac`
  - Entrada:
```
{
    "destinatario_hash_mac_fk": "  mac del usuario a observar "
}
```

  - Salida:
```
"4": {
    "oid_reg": 4,
    "tipo": "L",
    "fecha": "valor",
    "trabajo": null,
    "descanso": null,
    "oid_llamada_fk": 1,
    "oid_alarma_fk": null,
    "hash_mac_fk": null,
    "remitente_hash_mac_fk": "valor",
    "destinatario_hash_mac_fk": "  mac  del usuario a observar”
}
```


6.2 Métodos PUT

También poseemos métodos para la edición de todas las tablas exceptuando la de registro que será actualizada automáticamente mediante triggers.

Los métodos son los siguientes:



- editarUsuario
  - Routing: `/api/alarmas/editarUsuario`
  - Entrada: {

```
        "hash_mac": "mac del usuario a editar",
        "nif": "valor",
        "contrasena": "valor",
        "last_login": valor,
        "nombre": "valor",
        "apellidos": "valor",
        "rol": "valor",
        "nif_jefe": valor
    }
```

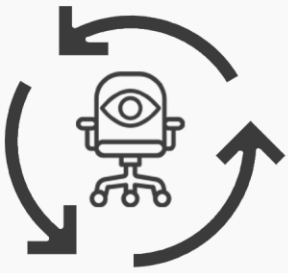
  - Salida: mensaje de confirmación

- editarAlarma
  - Routing: `/api/alarmas/editarAlarma`
  - Entrada: {

```
 "oid_alarma": oid alarma a editar,
 "dias": " valor ",
 "t_inicio": " valor ",
 "t_fin": "valor",
 "t_trabajo": valor,
 "t_descanso": valor,
 "ciclo_trabajo": valor,
 "ciclo_descanso": valor,
  "hash_mac_fk": "valor"
}
```

  - Salida: mensaje de confirmación

Chair Tracker



- editarLlamada
  - Routing: `/api/alarmas/editarLlamada`
  - Entrada:

```
{
 "oid_llamada": oid llamada a editar,
 "estado": "valor",
 "desde": " valor ",
 "descripcion": " valor”,
 "remitente_hash_mac_fk": " valor ",
 "destinatario_hash_mac_fk": " valor "
}
```

  - Salida: mensaje de confirmación

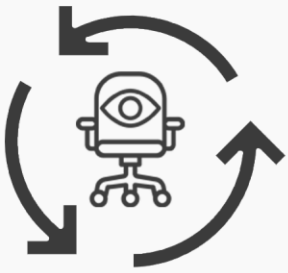
6.3 Métodos POST

Los métodos para añadir elementos a las tablas son los siguientes:

- anadirUsuario
  - Routing: `/api/usuarios/anadirUsuario`
  - Entrada:

```
{
 "hash_mac": "valor",
 "nif": " valor",
 "contrasena": " valor ",
 "last_login": valor,
 "nombre": " valor ",
 "apellidos": " valor ",
 "rol": " valor ",
 "nif_jefe": valor
}
```

  - Salida: mensaje de confirmación



- anadirAlarma
  - Routing: `/api/usuarios/anadirAlarma`
  - Entrada: {

```
"dias": " valor ",
"t_inicio": " valor ",
"t_fin": "valor",
"t_trabajo": valor,
"t_descanso": valor,
"ciclo_trabajo": valor,
"ciclo_descanso": valor,
"hash_mac_fk": "valor"
}
```

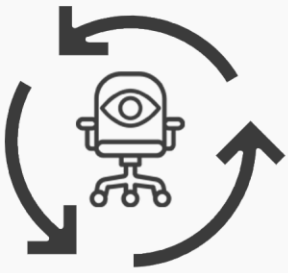
  - Salida: mensaje de confirmación

- anadirLlamada
  - Routing: `/api/usuarios/anadirLlamada`
  - Entrada: {

```
"oid_llamada": 1,
"estado": "valor",
"desde": " valor ",
"descripcion": " valor”,
"remitente_hash_mac_fk": " valor ",
"destinatario_hash_mac_fk": " valor "
}
```

  - Salida: mensaje de confirmación

Chair Tracker



6.4 Métodos DELETE

Si en algún momento necesitamos eliminar algún elemento de la BBDD, usaremos estos métodos (no será disponible eliminar llamadas ni registros para evitar malentendidos):

- borrarUsuario
  - Routing: `/api/usuarios/borrarUsuario`
  - Entrada:

```
{
 "hash_mac": "mac del usuario a borrar"
 }
```

  - Salida: mensaje de confirmación

- borrarAlarma
  - Routing: `/api/usuarios/borrarUsuario`
  - Entrada:

```
{
 "oid_alarma": 1
}
```

  - Salida: mensaje de confirmación

6.5 Capturas Postman

Se adjuntan diversos ejemplos de las funciones, en este caso son get, post, put y delete de la clase usuario, además de algunos métodos referentes a llamadas y alarmas. Los métodos no mostrados del resto de clases son muy similares.

Chair Tracker

localhost:8084/api/usuarios/

Save

GET localhost:8084/api/usuarios/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (2) Test Results 200 OK 273 ms 590

Pretty Raw Preview Visualize JSON


```
{
    "mac123": {
        "hash_mac": "mac123",
        "nif": "49163961H",
        "contrasena": "hola",
        "last_login": null,
        "nombre": "Manuel",
        "apellidos": "Garcia",
        "rol": "J",
        "nif_jefe": null
    },
```


localhost:8084/api/usuarios/anadirUsuario

Save

POST localhost:8084/api/usuarios/anadirUsuario

Params Authorization Headers (8) Body Pre-request Script Tests Settings

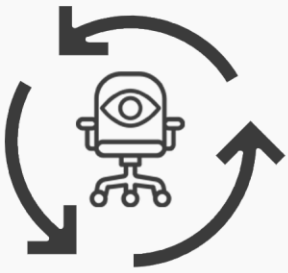
none form-data x-www-form-urlencoded raw binary GraphQL JSON


```
{
    "hash_mac": "mac111",
    "nif": "49168731H",
    "contrasena": "hola",
    "last_login": null,
    "nombre": "Antonio",
    "apellidos": "Perez",
    "rol": "J",
    "nif_jefe": null
```


Body Cookies Headers (2) Test Results 200 OK 14 ms 119

Pretty Raw Preview Visualize JSON

```
Añadido el usuario Antonio con hash_mac: mac111
```



Chair Tracker

localhost:8084/api/usuarios/editarUsuario

Save

PUT localhost:8084/api/usuarios/editarUsuario

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hash_mac": "mac111",
    "nif": "49168731H",
    "contrasena": "contrasena123",
    "last_login": null,
    "nombre": "Antonio",
    "apellidos": "Perez",
    "rol": "J",
    "nif_jefe": null
}
```

Body Cookies Headers (2) Test Results 200 OK 34 ms 118

Pretty Raw Preview Visualize JSON

```
Editado el usuario Antonio con hash_mac: mac111
```

localhost:8084/api/usuarios/borrarUsuario

Save

DELETE localhost:8084/api/usuarios/borrarUsuario

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hash_mac": "mac111",
    "nif": "49168731H",
    "contrasena": "contrasena123",
    "last_login": null,
    "nombre": "Antonio",
    "apellidos": "Perez",
    "rol": "J",
    "nif_jefe": null
}
```

Body Cookies Headers (2) Test Results 200 OK 18 ms 97

Pretty Raw Preview Visualize JSON

```
Borrado del usuario mac111
```

Chair Tracker

localhost:8084/api/llamadas/enviadas/hash_mac

Save

GET localhost:8084/api/llamadas/enviadas/hash_mac

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "remitente_hash_mac_fk": "mac123"
}
```

Body Cookies Headers (2) Test Results 200 OK 6 ms 253

Pretty Raw Preview Visualize JSON

```
{
    "1": {
        "oid_llamada": 1,
        "estado": "Pendiente",
        "desde": "Despacho de Martagon",
        "descripcion": "implementa registros",
        "remitente_hash_mac_fk": "mac123",
        "destinatario_hash_mac_fk": "mac789"
```

localhost:8084/api/alarmas/hash_mac

Save

GET localhost:8084/api/alarmas/hash_mac

Params Authorization Headers (8) Body Pre-request Script Tests Settings

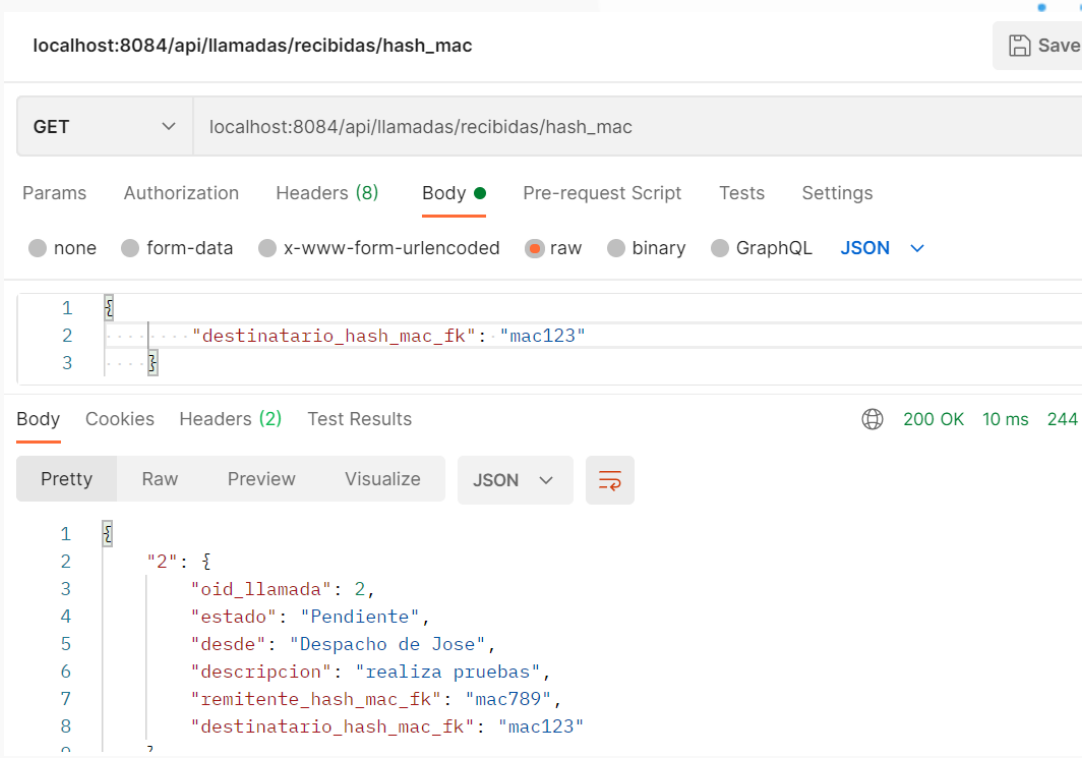
none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hash_mac_fk": "mac123"
}
```

Body Cookies Headers (2) Test Results 200 OK 12 ms 401

Pretty Raw Preview Visualize JSON

```
{
    "1": {
        "oid_alarma": 1,
        "dias": "L",
        "t_inicio": "PT20H12M30S",
        "t_fin": "PT21H15M",
        "t_trabajo": 30,
        "t_descanso": 10,
```



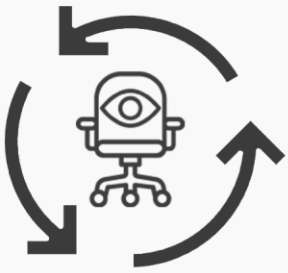
6. MQTT

Para la implementación del estándar MQTT en nuestro proyecto hemos hecho uso del broker de Mosquitto, el cual corre en el puerto 8083 y necesitaría de identificación.

6.1 Servidor

Desde el servidor utilizamos la librería de VERT.X para actuar como clientes al broker de Mosquitto y poder publicar mensajes. Estos son:

- Realización de una publicación de mensaje bajo la ruta `/destinatario_hash_mac/llamadas/` cuando añadimos una llamada desde la API REST usando el método anadirLlamada.

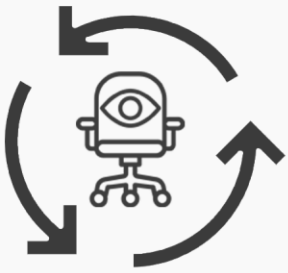


- Realización de una publicación de mensaje bajo la ruta `/hash_mac_fk/refresh/` cuando añadimos, editamos o añadimos una alarma desde la API REST usando los métodos anadirAlarma, editarAlarma y borrarAlarma respectivamente. Esto se hace con el objetivo de notificar al cliente siempre que haya un cambio en sus alarmas.

6.2 Cliente ESP8266

Desde el ESP8266 hemos implementado la librería PubSubClient para actuar como clientes respecto al broker al igual que hacemos en el servidor, en este caso estaremos suscritos a las rutas en las cuales publica el servidor, correspondiendo destinatario_hash_mac y hash_mac_fk al sha1 que realizamos a nuestra MAC al conectarnos a la red WIFI. De esta forma únicamente nos llegarán mensajes asociados a nuestro usuario, evitando así el recibir mensajes destinados a otros clientes. Según sean estos realizaremos distintas acciones:

- Al recibir una publicación de mensaje bajo la ruta `/destinatario_hash_mac/llamadas/` haremos uso de los actuadores, haciendo vibrar el dispositivo.
- Al recibir de una publicación de mensaje bajo la ruta `/hash_mac_fk/refresh/` realizaremos una búsqueda de la próxima alarma más cercana con los datos actualizados desde el servidor.



7. Cliente

7.1 Actuadores

Nuestro principal actuador se trata de un sensor de distancia por ultrasonidos, concretamente el HC-SR04, que gestionamos mediante la librería Ping y que nos permite saber si el usuario se encuentra sentado o no en la silla para actuar en consecuencia en el contador que se explicara a continuación con mayor detalle.

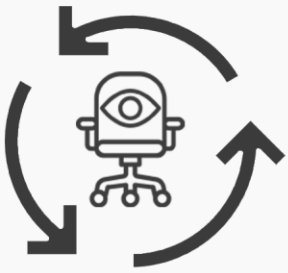
Además, para complementar a este nuestro sistema presenta una alarma (buzzer activo) que enviará avisos sonoros para que el usuario descanse o vuelva al puesto de trabajo.

Para el sistema de notificación de llamadas usamos un motor de vibración que permitirá al usuario percatarse de esta llamada sin confundirlo con el aviso sonoro que generan las alarmas.

7.2 Gestión alarmas

Para la gestión de alarmas nuestro sistema realizará una búsqueda de la próxima alarma más cercana en el tiempo y que corresponda al día actual, tomando como referencia para la hora y fecha los obtenidos mediante un NTP Client.

Esta operación se llevará a cabo no sólo en el setup, sino también cuando finalice nuestra alarma actual y en caso de no existir ninguna activa, realizaremos esta búsqueda cuando recibamos un mensaje MQTT en la ruta comentada anteriormente.



La búsqueda es realizada con la invocación del método obtenerProximaAlarma() que se encarga de ejecutar una petición por medio de la API REST de las alarmas asociadas a nuestra hash_mac para así realizar los cálculos pertinentes con la lista actualizada tal y como la encontramos en la base de datos.

Una vez nos encontremos en la franja horaria de una de las alarmas, se iniciará un contador para los ciclos de trabajo y otro para los ciclos de descanso. Dependiendo de si el usuario se encuentra sentado en la silla o levantado, se irán actualizando estos contadores y cuando estos coincidan con el tiempo de trabajo o descanso, haremos uso de la alarma para notificar al usuario.

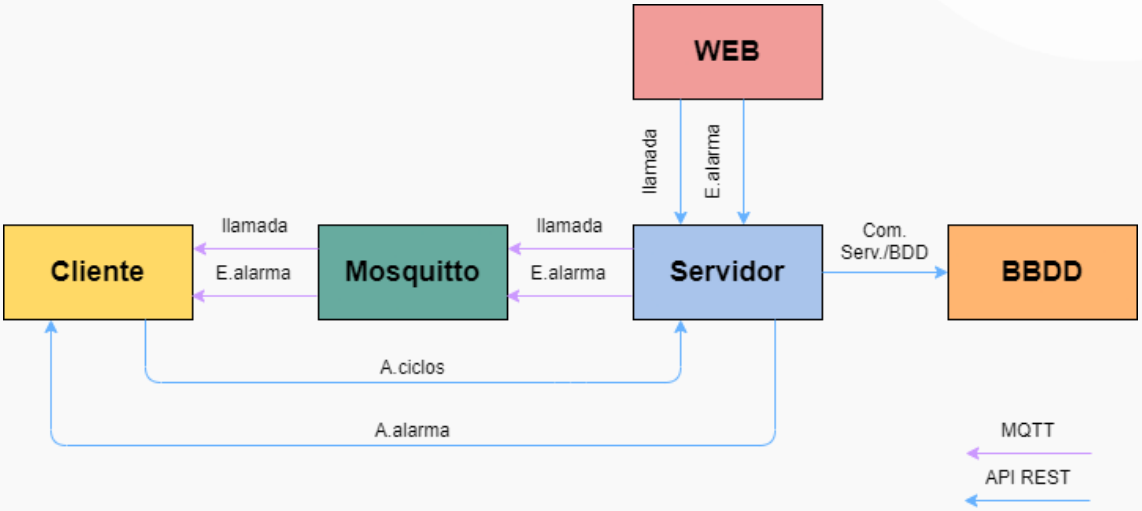
Tras completar su franja horaria, realizaremos un update a través de la API REST para actualizar los tiempos totales de trabajo y descansos realizados durante esta franja horaria.

7.3 Gestión llamadas

Para la implementación de llamadas en el cliente debemos de gestionar la recepción de llamadas puesto que el envío se realizará en un futuro mediante la interfaz web.

Para ello, cuando se añada una alarma en la BBDD por medio de la API REST, recibiremos un mensaje MQTT que será el aviso para el cliente de que deberá de activar el motor de vibración para notificar al usuario de una llamada entrante.

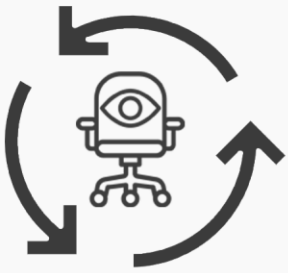
8. Flujo de datos y resumen del funcionamiento



El flujo de datos tiene comienzo desde la interfaz WEB que desarrollaremos, desde esta el usuario podrá establecer alarmas o realizar llamadas a otro usuario.

En el caso de que se establezca una alarma desde la web, se hará uso de la API REST alojada en el servidor que introducirá los datos en la BBDD, tras esto, el cliente realizará un get, el cual es el método obtenerAlarmasUsuario() de la API REST para obtener la lista actualizada de alarmas y poder calcular la próxima alarma con el método obtenerProximaAlarma(), una vez activa el cliente se encargará de realizar los cálculos temporales para más tarde, al finalizar la alarma, actualizar los datos en la BDD a través de la API REST del server con el método editarAlarma().

Cuando finalice la alarma se volverá a obtener la próxima o en caso de no estar activa ninguna de ellas, recibiremos un mensaje MQTT para actualizar la lista del cliente a la que se encuentra en la BBDD.



Cuando hacemos una llamada desde la WEB mediante la API REST, nuestro servidor publicará un mensaje en Mosquitto que recibirá nuestro cliente para activar el motor vibrador.

9. Implementación hardware del prototipo

9.1 Prototipo en fase inicial de desarrollo

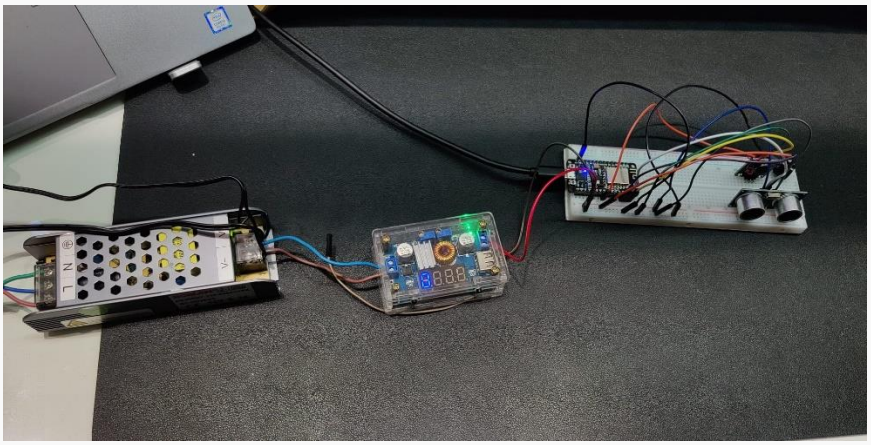
Para el prototipado inicial se ha hecho uso de una fuente 12V 8.3A, por lo que el voltaje ha tenido que ser regulado con un convertidor DCDC que permite regular el voltaje para obtener los 5V que serán usados para la alimentación del sensor de proximidad y el motor de vibración debido a las limitaciones del voltaje del ESP.

Para evitar el ruido en las señales de lectura que estábamos obteniendo en el HC-SR04 alimentado con una fuente externa, hemos tenido que normalizar las señales de tierra, conectando así el ESP a la tierra de la fuente para tomar esa tierra común como referencia.

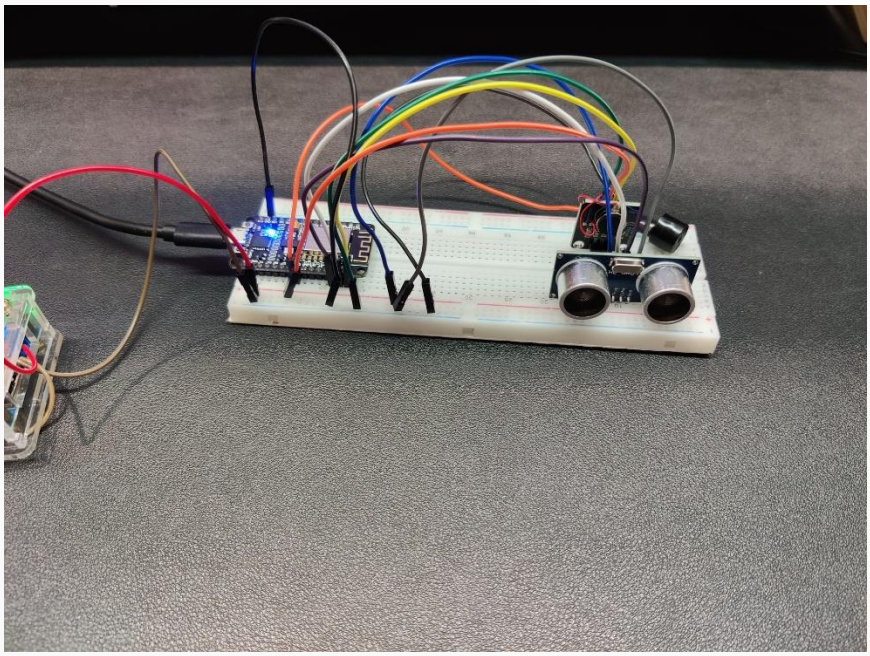
Las señales TRIGGER y ECHO están conectadas al ESP en las líneas D1 y D2. La alarma y motor vibrador están conectados a las D5 y D0, respectivamente.

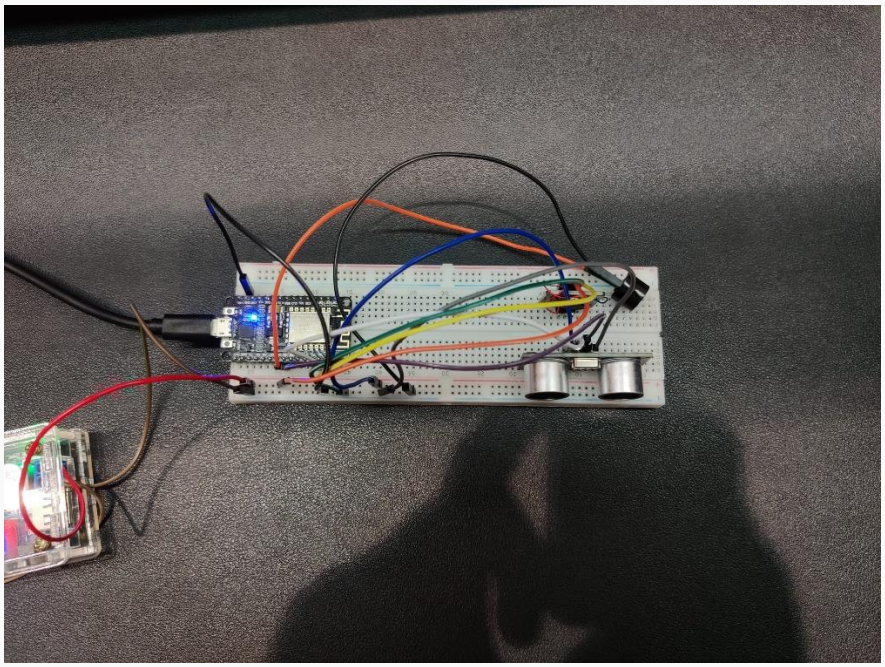
El ESP es aún alimentado mediante el USB del PC para así facilitar el desarrollo y tener acceso a la consola Serial del dispositivo.

En iteraciones futuras el sistema será autónomo y estará alimentado con baterías 18650 que tendrán la posibilidad de ser recargadas.

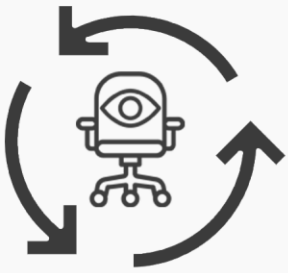


Chair Tracker





Chair Tracker

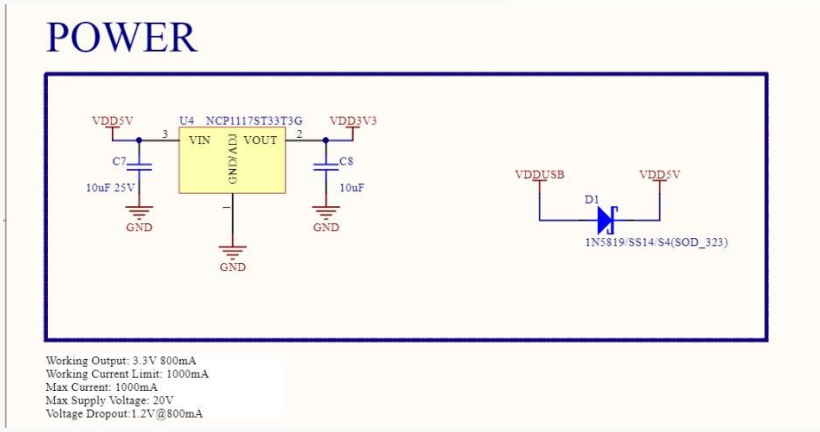


9.2 Prototipo en fase final de desarrollo

Para el prototipado final se ha hecho uso de un battery shield que hace uso de dos baterías 18650 y que nos suministra una línea de 5V, otra de 3.3V y alimentación mediante USB. Además, ofrece la posibilidad de recargar las baterías mediante un USB.

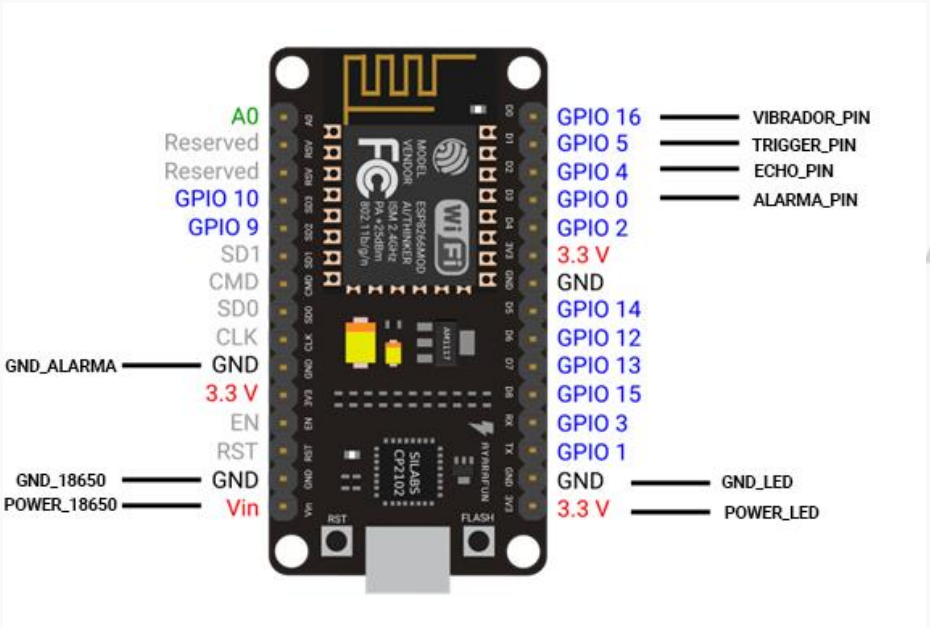
Como comentamos en la fase inicial necesitamos estos 5V para alimentar los sensores HC-SR04 y el motor vibrador. El plan inicial era alimentar el ESP8266 mediante el USB, pero hemos considerado que debido a que se trata de un producto en vías de desarrollo, lo más adecuado era realizar un prototipo orientado precisamente a esta parte del desarrollo por lo que era necesario añadir un USB que nos permitiera conectar el dispositivo a la consola Serial.

De esta forma, alimentamos el microcontrolador mediante la entrada Vin y hemos instalado un adaptador USB OTG MicroUSB-USB A hembra, a través del cual conectaremos un cable USB A macho-macho dirigido al PC de desarrollo. Es de importancia recalcar que es posible alimentar el ESP por la entrada Vin mientras mandamos datos por el USB gracias al diodo Schottky que se encuentra instalado entre la línea Vin y la USB. Tal y como se muestra en el esquemático:



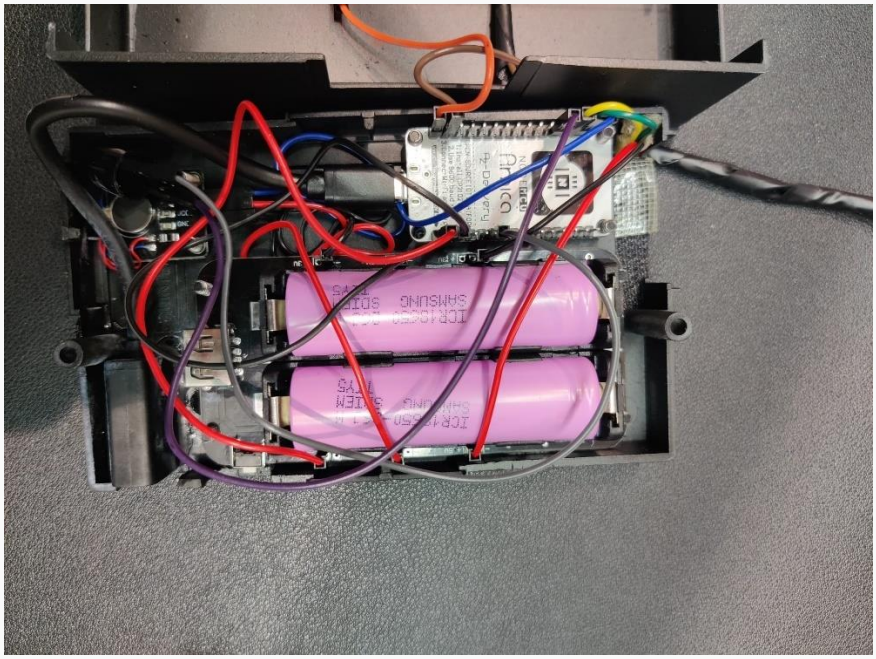
Esto nos obliga a que si queremos conectar el USB debemos de encender antes el dispositivo haciendo uso de las baterías o podríamos tener problemas de voltaje. Pero esto no supone problema al tratarse de una unidad de desarrollo y no un producto final de cara al usuario el cual no tendría la posibilidad de conectarse al serial y alimentaríamos el microcontrolador mediante el USB, lo cual le brinda más protecciones al usuario.

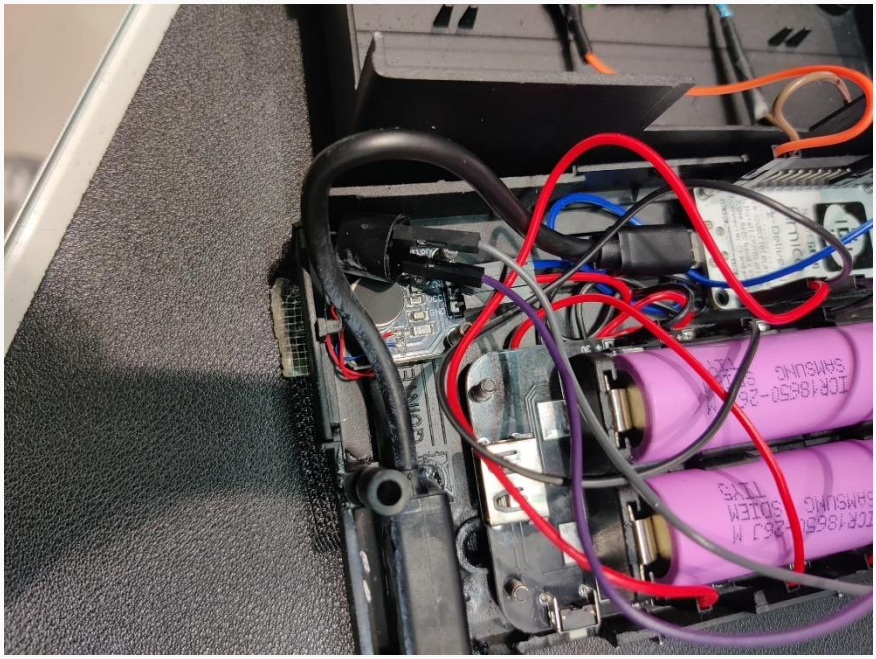
Quedando el conexionado tal y como se muestra en la siguiente imagen:



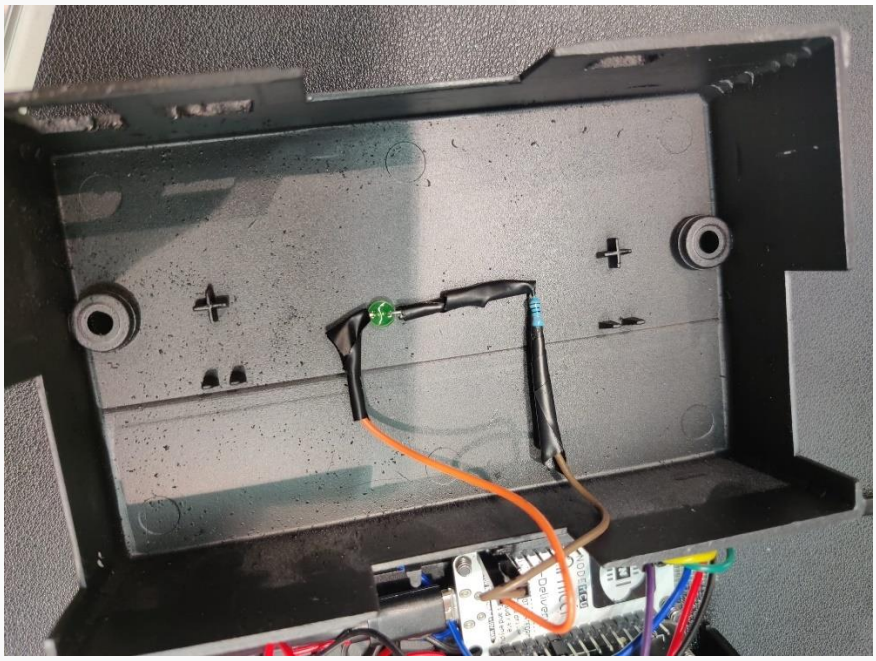
Chair Tracker

Hemos usado una caja contenedora la cual alberga los componentes sujetos mediante tornillos e irá instalada en la parte inferior de la silla y el sensor es instalado en el exterior con un soporte, en este caso hemos optado por instalarlo en el brazo de la silla. El montaje completo se puede observar en las siguientes imágenes:



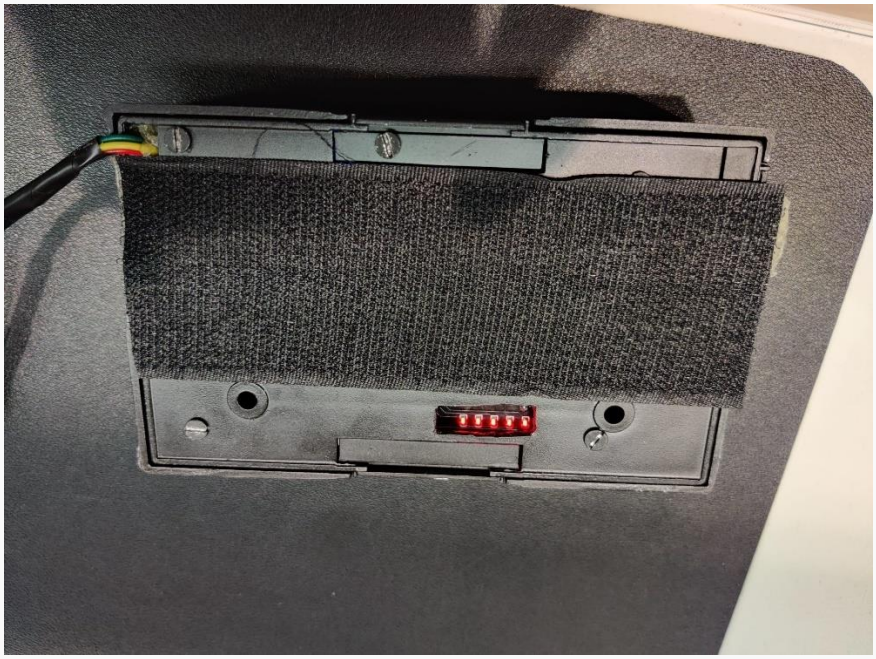


Chair Tracker



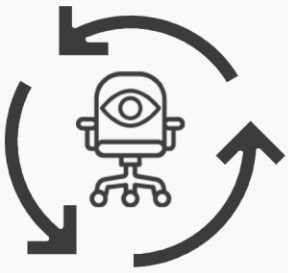
Chair Tracker

Chair Tracker



Chair Tracker

Chair Tracker



10. Cambios respecto a anteriores entregables

10.1 Cambios del segundo entregable respecto al primero

10.1.1 BBDD

Eliminación de la tabla de placas, asociando así la MAC de una placa a un usuario que será hará la nueva clave primaria. Todas las asociaciones con esta tabla han sido actualizadas.

Modificaciones en la tabla de alarmas, eliminando el parámetro de estado y sustituyendo el parámetro ciclo por ciclos trabajo y ciclos descanso.

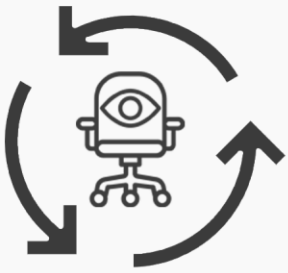
Desarrollo de nuevos triggers para el cumplimiento de las reglas de negocio definidas anteriormente.

10.1.2 Server

Implementado MQTT, publicando mensajes cuando se haga uso de ciertos métodos de la API REST junto con la modificación de los mismos para el correcto envió de datos por mensajes.

10.1.3 Cliente

Se han desarrollado métodos para el correcto funcionamiento del sistema con control de alarmas, llamadas, haciendo uso de subscripciones MQTT y API REST cuando sea necesario. Implementados actuadores.



10.2 Cambios del tercer entregable respecto al segundo

10.2.1 BBDD

Corrección de triggers dedicados al control del solapamiento de las horas introducidas por el usuario.

10.2.2 Cliente

Ajuste del criterio de los parámetros de detección del usuario.

10.2.3 Implementación física

Realizada implementación física final.

11. GITHUB

Acceso al repositorio

Chair Tracker