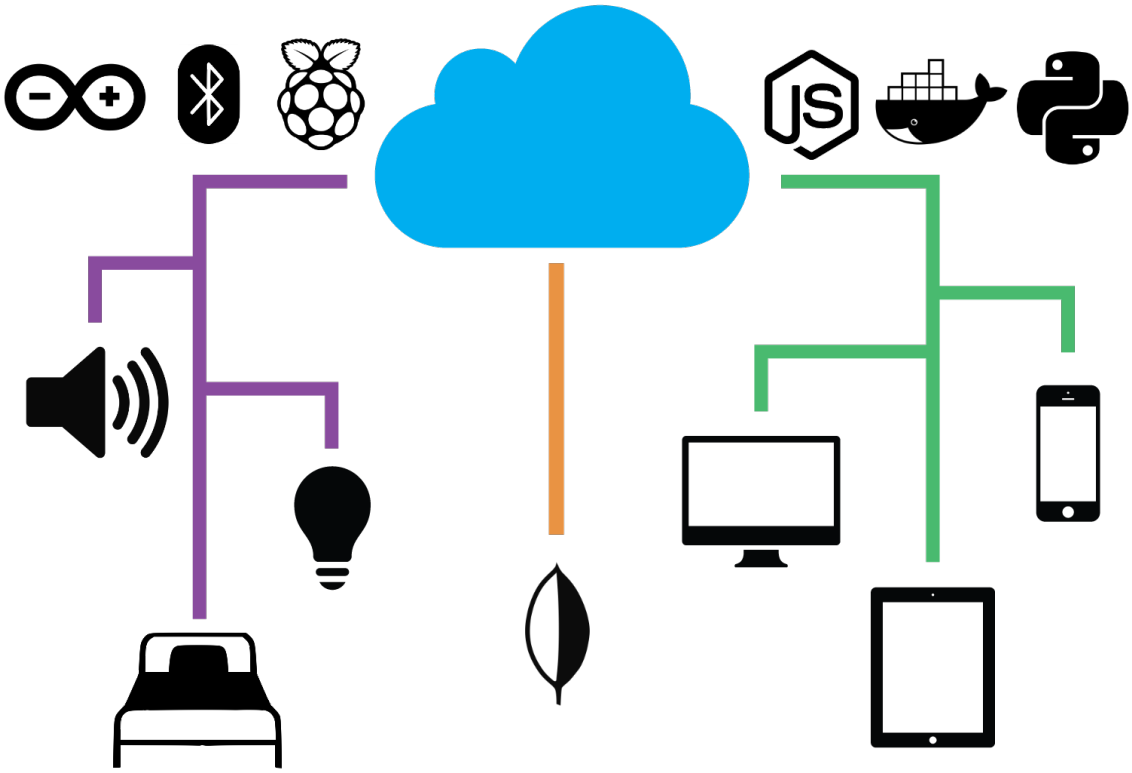
Computación ubicua:
Cama inteligente

Juan Casado Ballesteros

Álvaro de las Heras Fernández

Javier Fuentes Fernández

Daniel Mencía Berlinches



Índice

1. Tecnología 3

1.3. API Rest 9

1.4. Página web responsive 12

1.4.1. React 12

1.4.2. Bootstrap y CSS 12

1.4.3. Chart.js 13

2.Funcionamiento 18

3.Mejoras 18

Mejora del dashboard 18

Encriptación de datos en la API 18

Autorización y autenticación con OAuth 2.0 19

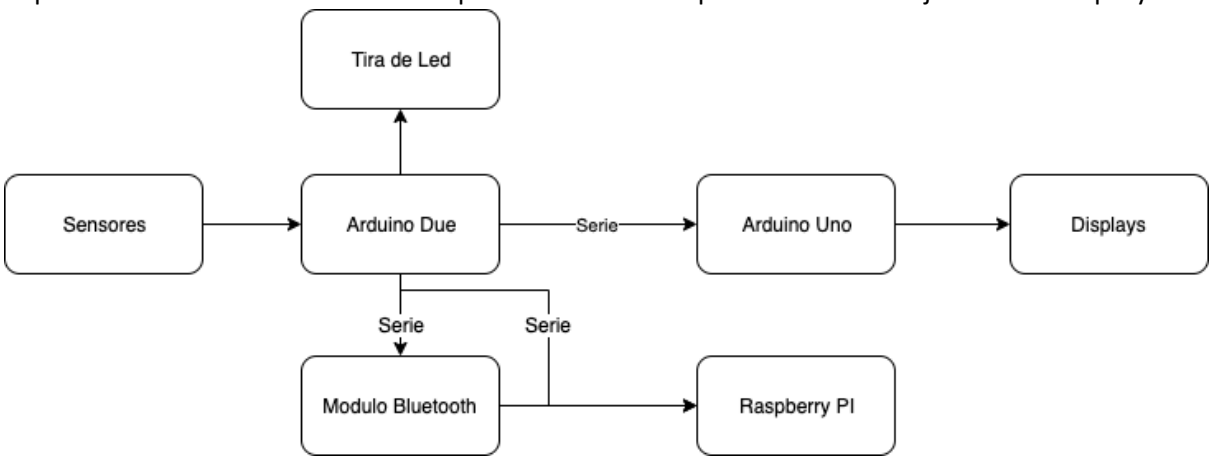
1. Tecnología

En el documento presentado para la PECL1 nos centramos en explicar tanto las intenciones que teníamos para capturar datos como para procesarlos, también hablamos sobre la metodología de trabajo que estábamos implementando como el enfoque que le habíamos dado al proyecto. En este documento proporcionamos un enfoque más técnico de lo que finalmente hemos implementado.

A lo largo del documento expondremos las tecnologías hemos utilizado, los problemas que nos han surgido y cómo hemos logrado superarlos. El discurso que presentamos se centra en flujo de la información a lo largo del producto. Desde cómo esta se captura a cómo se almacena y procesa y cómo finalmente se presenta a los usuarios. Explicaremos también las mejoras que hemos realizado sobre lo que planteamos originalmente.

1.1. Capturar los datos

Hablaremos aquí sobre cómo se capturan los datos. Cuál es la arquitectura hardware que implementamos. Nos centraremos por tanto en las parte de más bajo nivel del proyecto.



1.1.1. Placas controladoras

Utilizamos dos placas controladoras, una Arduino Uno y una Arduino Due. Ambas son dos placas controladoras open hardware mantenidas por la plataforma Arduino. La primera es una placa muy utilizada y de largo recorrido por lo que encontramos gran soporte para ella en forma de librerías. La segunda es una placa ARM de cortex M3 más nueva y con un soporte menor.

La ventaja de la segunda placa controladora reside en su mayor capacidad de cómputo tanto en velocidad de procesamiento como en cantidad de memoria. Adicionalmente esta placa tiene más cantidad de puertos nativos de comunicación hacia el exterior.

Inicialmente desarrollamos el proyecto solo con la placa Arduino Uno conscientes de que tendríamos que utilizar la segunda para mejorar los tiempos de respuesta, la velocidad de captura de la información.

El código desarrollado para las placas controladoras ha sido desarrollado íntegramente orientado a objetos en C++11. Esto nos ha permitido crear un código altamente modular que parte de objetos altamente cohesionados que implementan interfaces comunes y que son manejados a alto nivel permitiendo crear agrupaciones de ellos para proporcionar funcionalidad de mayor complejidad con un coste de desarrollo menor. Esto se ve cláramente en las clases FlexStrip y SwitchStrip donde

controlamos conjuntos de sensores Flex y Switch desde el alto nivel sin preocuparnos de los detalles internos de cómo cada una de esas clases maneja, procesa y filtra los datos que el hardware captura.

```
#pragma once

#include "Filter.h"



class Sensor {

public:

 virtual void init ();

 virtual float rawRead();

 virtual float read ();

};
```

Las mayores dificultades encontradas al utilizar dos placas controladoras, así como la gran cantidad de sensores y actuadores que tenemos han venido de cómo alimentar todo el sistema. Dicho problema lo hemos resuelto utilizando una pequeña fuente de alimentación portátil que alimenta todos estos dispositivos. Con ello hemos logrado que con una sola toma de corriente poder alimentar todo el conjunto excepto la Raspberry pi que alimentamos aparte. Alimentamos las controladoras por su pin VIN a 5v reduciendo el consumo respecto a hacerlo por el puerto USB. Además, esto nos permite tener una entrada de mayor amperaje que si no tuviéramos no nos permitiría conectar tantos dispositivos. La fuente de alimentación nos proporciona 2A a 5V, que tras medir el voltaje real se nos reduce a 4.5V. De ello sabemos que nuestro sistema consume 10W de potencia y requiere 2.2A para funcionar.

Debido a que estamos en la frontera de potencia que el sistema puede proporcionar esto nos ha obligado a hacer que los sensores no regulados se ajusten al voltaje de forma autónoma ya que sus medidas se realizan a partir de un divisor resistivo, explicaremos esto en más detalle en el apartado 1.1.2.

Otras de las ventajas de tener una fuente de alimentación en vez de utilizar los otros métodos es la seguridad. En caso de corto la fuente se deshabilita protegiendo el hardware en la medida de lo posible. A la hora de comprar una fuente de alimentación portátil con estas características es recomendable tener en cuenta la velocidad con la que esto sucederá. Si las medidas de seguridad se activan demasiado tarde es como si no las tuviéramos.

1.1.1.1. Arduino Due

Utilizamos esta placa para capturar la información de todos los sensores, así como para proporcionar la salida de la información sobre el estado de los sensores de flexibilidad. Utilizamos un total de cinco entradas analógicas para los sensores de flexibilidad, una para el de sonido, otra para el de luz. Adicionalmente utilizamos seis entradas digitales para seis interruptores y dos para leer los sensores de peso. Adicionalmente utilizamos tres puertos serie para comunicarnos con el exterior y once salidas pwm para los leds que muestran los valores de los sensores de flexibilidad y una salida digital para encender y apagar un led a cada ciclo del bucle de control.

La funcionalidad que se implementa consiste en leer los sensores, filtrar las medidas capturadas y enviarlas a la raspberry PI, a un módulo bluethooth y a los actuadores.

Al utilizar esta placa nos hemos encontrado con dos problemas que cuya definición y resolución expondremos a continuación.

Esta placa controladora utiliza entradas analógicas a 5v mientras que el resto de las entradas, así como la alimentación de la misma son a 3V. Hemos resuelto esto incluyendo divisores resistivos de 5V a 3V delante de las entradas analógicas. Ya que uno de los objetivos del diseño hardware era ser lo más modular posible para solventar el problema hemos diseñado un módulo con el que poder realizar esta conversión. El módulo es digamos coloquialmente de quita y pon, de modo que en el caso de utilizar esta placa se utilizará, pero para usar otra controladora cuyas entradas sean a 5V solo habrá que retirarlo. Se puede por tanto mantener el mismo diseño hardware con distintas controladoras, inicialmente estaba con la Arduino Uno sin tener el módulo logrando la misma funcionalidad.

Adicionalmente esta placa a diferencia de la Arduino Uno consta de un USB nativo, además de otro USB controlado por un chip fuera del intrado principal. Esto que a priori parece una ventaja realmente nos ha hecho eliminar una de las características de nuestro diseño. La Arduino Uno traduce el puerto USB como una interfaz Serie a partir del código de su bootloader. Esto nos permite producir un flanco de subida sobre su reset desde el módulo bluetooth para subir código a la placa mediante él. No obstante, esto no es posible en la Arduino Due ya que el bootloader en esta placa no realiza esta acción ya que no lo necesita pues dispone de otras interfaces para recibir el código.

1.1.1.1. Arduino Uno

Utilizamos esta placa para proporcionar salidas al usuario sobre las medidas que los distintos sensores están capturando mediante tres displays de siete segmentos de ánodo común. Utilizamos displays de siete segmentos debido a que nos permiten controlarlos mediante integrados que convierten BCD o números binarios de cuatro bits a salida que podemos introducir directamente a los displays. Son de ánodo común ya que los integrados que utilizamos tienen la entrada negada como es habitual. De este modo mantenemos la consistencia entre la representación interna en la controladora y lo mostrado en el display. Un valor de 0011 o 4 en decimal se mostrará cómo un 4 en el display en vez de utilizar 1100 como representación interna.

La integración entre la Arduino Due y la Arduino Uno se implementa mediante un protocolo de comunicación serie con una capa de aplicación basada en paquetes fragmentados en formato ASCII. Esto quiere decir que los paquetes no tienen por qué enviarse íntegros lo cual no es posible de realizar por el protocolo serie por lo que esto era más una necesidad que una funcionalidad. Los paquetes en el protocolo serie son a lo sumo de 8 bits. Que el protocolo implementado entienda caracteres ASCII nos ha facilitado el desarrollo pues hemos podido crear los paquetes de prueba de forma manual, así como utilizar las librería de formato para imprimir datos sobre el bus serie de Arduino.

El protocolo consta de los siguientes elementos. Primero se envía un byte inicial para indicar que se va a enviar un paquete nuevo, posteriormente byte a byte se envían caracteres ASCII que codifican un número entero o decimal. Finalmente se envía otro indicador que define el final del paquete. Tras eso la información enviada se procesa y se muestra sobre los displays. Una persona conectada a la placa podrá crear los paquetes cómodamente del mismo modo que lo hace la Arduino Due para enviar los valores de los sensores que sean mostrados en los displays.

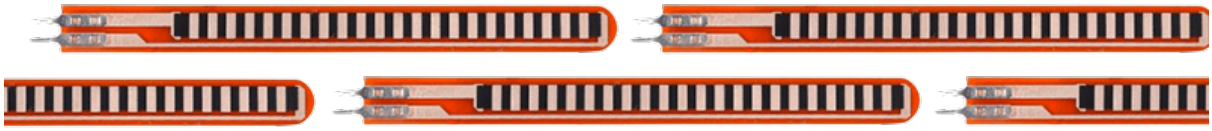
1.1.2. Sensores

A nivel hardware hemos configurado cada sensor como elementos aislados que puedan activarse o desactivarse cómodamente. Esto ha sido crítico para el desarrollo del proyecto ya que cada sensor se relaciona con el resto en la medida en la que la alimentación para cada uno de ellos es común. Al margen de eso cualquier sensor puede retirarse de la placa sin temor a afectar a los otros. Adicionalmente esto hace que el hardware sea más fácil de entender en un solo vistazo que cuando todo está mezclado con todo con cables colgando de unos puntos a otro de las placas.

1.1.2.1. Sensores de flexibilidad

Los sensores utilizados informan de cuan doblados están. Como ya indicamos en el anterior documento, hemos situado estos sensores en una disposición específica que nos permite saber con exactitud la postura y posición del usuario.

Esta disposición es, como se indica en la imagen, dos líneas paralelas que cubrirán el ancho de la cama, y en la que e todo momento el centro de uno de los sensores estará completamente doblado.



Realizando un seguimiento de esta información somos capaces de detectar la posición del usuario.

La implementación de este sensor conjunto no es directa si no que pasa por un proceso de filtrado y amplificación de la señal. La longitud de la cama se discretiza en el doble de segmentos de la cantidad de sensores de que disponemos. Cuando dos sensores próximos indican que están doblados podemos identificar que el lateral del cuerpo del usuario realmente entre ellos en vez de sobre ambos a la vez. Para realizar esto utilizaremos un filtro bayesiano configurado para proporcionar apertura sintética a las medidas.

Tratando la función discreta que generamos somos capaces de inferir la posición en la que está el usuario situado en la cama, así como tratar de deducir si su postura es buena o no. Entendemos que si el usuario ocupa mucha superficie sobre los sensores su postura será peor que cuando la superficie es mayor. Adicionalmente estos sensores combinados con los sensores de peso nos permiten determinar si alguien está tumbado en la cama. Combinando ambos sensores somos capaces de producir mejores resultados clasificando correctamente casos como que se haya puesto peso encima de la cama.

Para trabajar con estos sensores hemos tenido que utilizar divisores resistivos puesto que las entradas de de la placa Arduino Due son de 3V y no de 5V.

Debido a que la superficie sobre la que cada sensor se pueda encontrar puede ser ligeramente más o menos rígida o estar más o menos curvada por defectos en la cama o el colchón hemos implementado un filtro adaptativo. Con esto conseguimos que los sensores se puedan ajustar de forma dinámica en lugar de tener parámetros de ajuste fijos. De esta manera se obtiene un poco más de ruido en las lecturas, pero se sabe con mayor certeza si el sensor está doblado o en reposo. Las medidas en los extremos se verán perturbadas, no obstante, obtendremos mejores resultados para el conjunto de los sensores pues cada uno se ajusta a sí mismo en vez de todos compartir una configuración con compromisos. Esto también nos ha permitido superar problemas como las caídas de voltaje que,

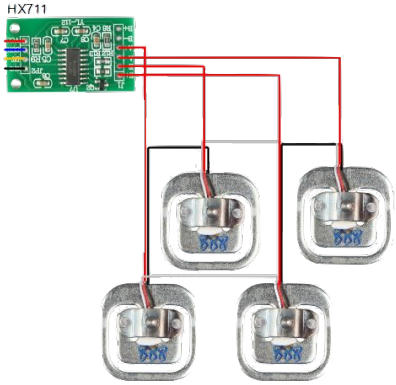
aunque pequeñas perturban a estos sensores de forma notable. Dichas caídas se producen cuando muchos leds están encendidos simultáneamente.

1.1.2.2. Sensores de peso

Este sensor nos permitirá saber con facilidad cuando el usuario está o no encima de la cama. Para ello lo combinamos con los sensores de flexibilidad para obtener resultados menos propensos a producir falsos positivos.

Esto nos permitirá registrar los horarios de sueño y saber si el usuario se ha sentado o tumbado en la cama fuera de horario nocturno.

Estos sensores no son utilizados de forma directa si no a través de un HX711, un convertidor analógico digital diseñado para ser utilizado con sensores de peso. Los registros de este integrado se pueden rellenar directamente mediante la entrada digital que nos proporciona como interfaz de comunicación. Uno de los registros actúa como offset permitiéndonos desplazar en el eje de ordenadas (vertical, peso) las medidas que emitirá a lo largo del eje de abscisas (horizontal, tiempo). El otro registro que nos expone nos permite establecer una ganancia o factor de escalado por el que las medidas se multiplican antes de ser enviadas.



medida = toDigital(medida de los sensores , 24) * ganancia + offset

Este amplificador será leído desde las placas Arduino mediante una entrada digital lo que proporcionará la información del peso y otra salida digital para generar una señal cuadrada que marcará el ritmo al que el amplificador enviará la información.

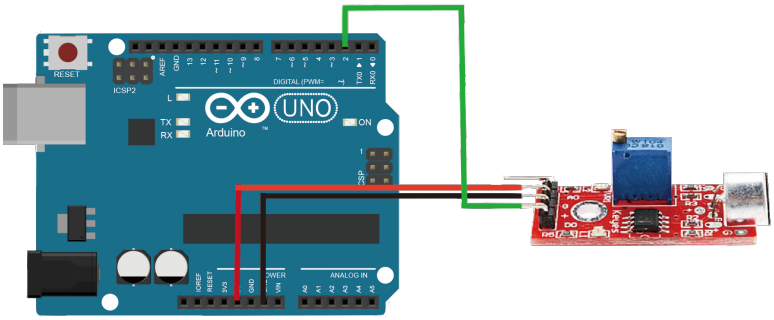
1.1.2.3. Sensores de ruido

Utilizamos un sensor KY-038 para detectar el sonido ambiente y valorar cómo éste afecta a la calidad del sueño.

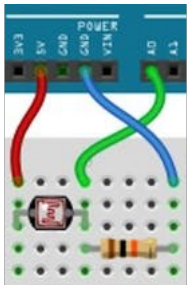
Para este sensor, en la mayoría de las aplicaciones solo es necesario utilizar su salida digital que indica cuando el nivel de ruido supera el umbral que nosotros fijemos. Detectar que umbral es el correcto para nuestro propósito es una tarea crítica en la configuración del sensor.

También se podría utilizar la otra salida analógica del sensor y en este caso se fijaría el umbral en el código de nuestra aplicación. Pero esto requeriría de configurar un umbral a partir del que considerar que hay mucho ruido.

No obstante, nosotros hemos decidido utilizar una aproximación transversal para utilizarlo. El sensor produce señales estables cuando el nivel de ruido es estable y señales inestables cuando el nivel de ruido sube o baja. Debido a que los ronquidos son picos sobre el nivel de ruido podremos aprovechar esto para detectarlos. En vez de utilizar la salida del sensor directamente la utilizaremos en su forma derivada de modo que tendremos valores altos cuando haya picos de sonido y valores bajos cuando el sonido sea estable. Con esto no solo logramos detectar los ronquidos si no también eliminar falsos positivos como ruido de fondo que podría quedar o no por debajo de un umbral mal calibrado produciendo malas medidas.



1.1.2.4. Sensores de luz



Utilizamos estos sensores para detectar la cantidad de luz ambiente con las que el usuario está durmiendo. En base a ella podremos recomendar que procure dormir con una cantidad de luz si fuera muy alta.

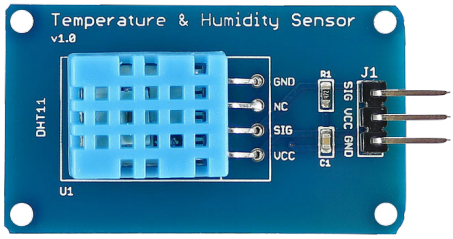
El sensor elegido para hacer esto es una LDR que leeremos con una entrada analógica a través de un divisor resistivo.

1.1.2.5. Sensores de temperatura y humedad

Estos dos sensores funcionan de forma conjunta ya que tienen un propósito común. En nuestro caso en conocer la calidad ambiental, basándonos en la temperatura y la humedad.

Conocer estos dos valores nos permitirá saber si la calidad del sueño ha podido verse afectada por una temperatura poco cómoda o una humedad fuera de lo normal.

El sensor utilizado es una variante del DHT11 en el que ya se incorporan resistencias para pull-up y un condensador de filtrado entre GND y VCC.



1.1.3. Actuadores

Utilizamos una tira de LEDs para mostrar el estado de los sensores de flexibilidad desde la Arduino Due. Adicionalmente utilizamos los tres displays de siete segmentos para mostrar el valor de los sensores. Podemos mediante los interruptores indicar a la Arduino Due el sensor que deseamos que nos muestre sobre los displays.

Los actuadores nos han permitido agilizar el diseño ya que nos permiten ver la información de forma cómoda directamente sobre el hardware para configurar con la mayor precisión posible cada sensor.

Adicionalmente se puede hasta cierto punto considerar al módulo bluetooth como un actuador. Utilizamos este módulo para proporcionar de comunicación inalámbrica al dispositivo permitiendo a otros que se conecten a él. Por el módulo bluetooth se envían los datos de los sensores en formato JSON. Podremos con nuestro portátil, móvil o desde las Raspberry PI conectarnos al módulo para ver la información que se está produciendo en cada momento.

Adicionalmente incorporamos un modo bomba en el que se inicia una cuenta atrás en el display que cuando llega a hace que la Raspberry PI haga el sonido de una explosión. Si no nos lo pasamos bien con la tecnología no sé con qué lo vamos a hacer.

1.2. Raspberry PI

En esta sección hablaremos sobre cómo se hace el primer procesamiento de la información y sobre como esta es enviada desde la Arduino Due a la Raspberry y desde está a la base de datos para ser almacenada.

Utilizamos dos modelos distintos de Raspberry PI, el modelo 3B ha sido con el que comenzamos a trabajar, pero para navidades tuvimos que utilizar el modelo B+. Las diferencias entre ambos residen principalmente en la capacidad de procesamiento, el modelo 3B tiene más memoria y un procesador más rápido. Para utilizar el modelo B+ tuvimos que aumentar la frecuencia del reloj para poder utilizarlo más cómodamente.

Otra diferencia entre ambos modelos es que el 3B dispone tanto de wifi como de bluetooth incorporado mientras que el B+ no. Esto ha hecho que hayamos tenido que adaptar nuestro código a funcionar tanto por cable como de forma inalámbrica a la espera de saber qué modelo podremos utilizar para la presentación. Hacerlo no ha sido demasiado complicado.

El procesamiento base de los sensores es realizado en la RaspberryPI. En ella se obtienen tres medidas nuevas a partir de los sensores. Estas medidas son la calidad del sueño calculada como un porcentaje a partir de la humedad, la temperatura y el ruido de la habitación. Si estos valores difieren en exceso de lo que deberían la calidad bajará. También inferimos la posición del usuario en la cama utilizando los sensores de flexibilidad. Para ello trataremos el vector de medidas como una función de energía que queremos aproximar a una gaussiana. Calcularemos la media de la gaussiana y su desviación y aproximamos esto a indicar si el usuario está en un lateral o en el centro de la cama y si está esparcido en la cama o tumbado recto. Adicionalmente utilizando los sensores de peso y de flexibilidad inferimos su hay alguien tumbado en la cama o no.

Hemos creído conveniente hacer esto en la Raspberry PI para no saturar al servidor y hacer el producto más escalable. Si cada Raspberry PI procesa los sensores conectados a ella esto supondrá una carga de trabajo muy pequeña. Pero si el servidor tuviera que procesar los de todos los usuarios se convertiría en una tarea muy pesada.

Adicionalmente el resto de los valores calculados como la media de las horas de sueño o de temperatura se calculan en el cliente, es decir, en la página web. Esto se hace bajo la misma premisa de no saturar al servidor. La página web tiene que extraer de la base de datos estos valores para mostrarlos, además calcular la media de un día solo tiene sentido si el usuario desea verla por lo que nos ha parecido conveniente implementarlo así y calcular solo lo que hace falta cuando hace falta y no antes.

En la RaspberryPI tenemos además un altavoz que nos permite hacer sonar una alarma que el usuario puede configurar desde la web. Este altavoz es también utilizado en el modo bomba. Cuando es necesario se cargan y reproducen los archivos de audio que indiquemos.

Mencionamos también que utilizamos systemd para lanzar nuestro código como un daemon cuando la Raspberry es encendida. Explicaremos esto en más detalle en el apartado 1.5.1

1.3. API Rest

Hemos desarrollado una API Rest para gestionar el acceso a los datos de la base de datos en MongoDB. Esta API se ha desarrollado con Node.js y mongoose, mediante el uso de mongoose conseguimos añadir una capa a MongoDB teniendo así un control sobre los datos que se introducen, evitando así

valores erróneos, para tener así una base de datos coherente. Las peticiones que se han implementado dentro de esta son:

- **GET:** se han implementado llamadas GET para obtener todos los datos, obtener datos relativos a un usuario mediante su Id y para obtener los datos de un objeto mediante el Id del objeto.

```
app.get("/alarms/:id", async (request, response, next) => {
    try {
      var record = await AlarmModel.findById(request.params.id).exec();
        response.send(record);
    } catch (error) {
        response.status(500).send(error);
    }
});
```

- **PUT:** se ha implementado una petición PUT para poder modificar objetos, principalmente se usa para actualizar los estados de las alarmas de la cama. Recibiendo en un JSON los campos a actualizar, además de requerir el Id del objeto.

```
app.put("/alarm/:id", async (request, response, next) => {
    try {
        var record = await AlarmModel.findById(request.params.id).exec();
        record.set(request.body);
        var result = await record.save();
        response.send(result);
    } catch (error) {
        response.status(500).send(error);
    }
});
```

- **POST:** se tiene una llamada POST para cada objeto de tal forma que pueden crearse nuevos, siempre y cuando sigan el formato que Mongoose obliga. Este recibe como cuerpo el JSON con los datos.

```
app.post("/alarm", async (request, response, next) => {
    try {
        var record = new AlarmModel(request.body);
        var result = await record.save();
        response.send(result);
    } catch (error) {
        response.status(500).send(error);
```

```
    }
});
```

- **DELETE:** se tiene una petición DELETE para eliminar los objetos que no se vayan a usar, en este caso alarmas. Para ello se ha de pasar el Id del objeto a borrar.

```
app.delete("/alarm/:id", async (request, response, next) => {
    try {
        var result = await AlarmModel.deleteOne({ _id: request.params.id
}).exec();
        response.send(result);
    } catch (error) {
        response.status(500).send(error);
    }
});
```

Todas estas operaciones se han desarrollado para cada tipo de dato que hemos definido con mongoose, habiendo muchos más modelos para los datos generados por los sensores.

Un ejemplo de estos modelos sería el siguiente definido para los datos de la matriz de sensores de flexibilidad:

```
const FlexModel = _mongoose.model("flex",{
    userId:     {
        type: String,
        required: true
    },
    value:
    {
        type:  [Number],
        required: true
    },
    captured:
    {
        type: Date,
        required: true
    }
});
```

1.4. Página web responsive

Para visualizar los datos se ha desarrollado una página web responsive usando React, CSS, Chart.js y Bootstrap.

1.4.1. React

Se ha empleado React debido a que es una librería de javascript que permite el desarrollo páginas web de una sola vista mediante el uso de componentes. Se trataba de una herramienta ideal para la página debido a su buen manejo de los datos, además de poder ejecutarse con Node.js y ser compatible con móviles gracias a React native. Por lo que todo el back-end de la página se basa en Node.js y React.

```
class Login extends React.Component {
  constructor(props) {
    super(props);
    this.state = { clicked: false};
    this.state = { error: false};
    this.handleUser = this.handleUser.bind(this);
    this.handlePassword = this.handlePassword.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
```

Clase y constructor de componente de React.

1.4.2. Bootstrap y CSS

Para el diseño de la web, hemos querido hacerla responsive de tal forma que vale para múltiples dispositivos, como ayuda hemos utilizado Bootstrap que ofrece una gran cantidad de recursos para la creación de páginas así. Principalmente hemos usado Bootstrap en el dashboard, mientras que el resto de la página se ha hecho con clases en CSS3, para hacer uso de las media-queries o medidas como vw o vh, permitiendo así tener el diseño responsive que queremos.

```
  .chart-wrapper {
    padding: 2%;
    display: inline-block;
  }

  .main.chart-wrapper {
    width: 50%;
    height: 400px;
  }
  .sub.chart-wrapper {
```

```
    width: 29%;
    height: 300px;
  }
@media screen and (max-width: 768px) {
  .main.chart-wrapper {
    width: 96%;
    height: 400px;
  }
  .sub.chart-wrapper {
    width: 96%;
  }
```

Fragmento de código con la media query.

1.4.3. Chart.js

Para mostrar los datos en el dashboard hemos implementado gráficas interactivas usando la librería de Chart.js. Esta librería ofrece una gran cantidad de gráficas totalmente personalizables, además dispone de módulos que aportan más funcionalidad aún como pueden ser anotaciones. Las gráficas que hemos empleado son de barras, líneas y polares.

```
    var sleepData = {
      labels: ["Sleeped", "Max", "Min", "Last Week"],
      datasets: [{
        // Remove undefined
        data: [yesterday.filter(Number), 9, 7, week.filter(Number)],
        backgroundColor: [
          "#003f5c",
          "#f95d6a",
          "#ffa600",
          "#665191"
        ]
      }]
    };
    // Chart with Chart.js.
    this.sleepChart = new Chart(this.sleepRef.current, {
      type: 'polarArea',
      options: {
```

```
        responsive: true,
        maintainAspectRatio: false,
        title: {
          display: true,
          text: 'Sleep time and recommendations'
        }
      },
      data: sleepData
    });
  };
```

Gráfico polar de Chart.js

1.5. Despliegue

Dividiremos este apartado en lo que el despliegue supone para el hardware creado y en lo que el despliegue supone para el software creado.

1.5.1. Despliegue hardware

Respecto del hardware el que realmente realizará el despliegue será el usuario al comprar el producto. Es deseable que este sea capaz de enchufarlo a la corriente darle al botón de encendido y que todo esté listo para que lo pueda utilizar. Respecto de las placas controladoras esto no supone demasiado problema, en cuanto son alimentadas comienzan a capturar y transmitir datos. No obstante, esto no sucede así con la Raspberry PI. Lograr que esto se produzca puede conseguirse de múltiples maneras, las más sencillas consisten en incorporar esto como parte de algún otro programa que se inicie cuando el sistema operativo se carga. No obstante, esto resulta poco elegante y poco fiable.

Para lograrlo hemos recurrido a systemd. Systemd es el programa encargado de iniciar los componentes del kernel de LINUX en la mayoría de sus distribuciones. Adicionalmente este programa es utilizado para lanzar daemons que deban existir cuando el sistema operativo esté cargado. Una de las ventajas de utilizar systemd consiste en que podemos indicar cuando queremos que nuestro código se ejecute, así como la forma en la que queremos que lo haga. En nuestro caso indicaremos que queremos lanzar nuestro código como un daemon o programa sin padre y que queremos que se ejecute cuando el entorno multi-usuario esté cargado que entre otras cosas implica que tendremos acceso a internet a la comunicación serie y al bluetooth que son los requisitos de nuestra aplicación. También indicamos que si el proceso muere por un error systemd deberá reiniciarlo y que queremos que se ejecute como root para poder controlar la interfaz serie.

```
[Unit]

Description=SmartBed

After=multi-user.target

[Service]
```

```
Type=simple

ExecStart=/usr/bin/python3 /home/pi/COMPUTACION_UBICUA/src/raspberry/main.py

KillMode=process

Restart=on-failure

User=root

Group=root



[Install]

WantedBy=multi-user.target
```

Con todo esto en cuanto nuestro producto sea alimentado comenzará a funcionar automáticamente.

1.5.2. Despliegue software

El despliegue software por el contrario está más orientado a los desarrolladores. Estos deberán actualizar el servidor cuando se realicen cambios sobre él, así como instalar el código necesario en las placas controladoras y la raspberry.

Para automatizar todos estos procesos hemos creado distintos shell script que los realizan de modo que resulta más sencillo añadir nuevos pasos al proceso o modificar algunas de sus partes. Tendremos un shell script que está pensado para ser ejecutado en el servidor y que descarga de github el código, desactiva los procesos de docker antiguos, carga los nuevos y muestra su estado, otro que actualiza systemd para indicarle cómo iniciar nuestro proceso lo cual es útil para configurar una nueva raspberry, otro que actualiza las librerías de arduino para incluir las nuevas y así con cada tarea.

Adicionalmente debido a que nuestro servidor estará sobre un equipo cloud alquilado como IAS por uno de nosotros hemos decidido utilizar docker para realizar allí el despliegue de nuestro código. Esto nos permitirá eliminarlo sin dejar rastro de posibles dependencias o programas instalados. Adicionalmente todas las dependencias que tengamos estarán encapsuladas dentro de un contenedor. Docker también nos permite controlar nuestros procesos de una forma más cómoda.

El uso de docker y docker-compose ha sido crítico pues nos ha permitido trabajar con el código localmente con la garantía de que este funcionará sin mayor configuración cuando lo despleguemos sobre el servidor.

```
version: '3.3'

services:

 database_api:

   container_name: database_api

   build: ./database_api/
```

```
  volumes:
    - ./database_api/:/database_api
  ports:
    - 8080:8080
  depends_on:
    - "mongo"
web:
  container_name: web
  build: ./web/
  volumes:
    - ./web/:/web
  ports:
    - 80:80
  depends_on:
    - "mongo"
    - "database_api"
mongo:
  container_name: mongo
  image: mongo:4.2.2-bionic
  restart: always
  volumes:
    - ./mongo/setup:/docker-entrypoint-initdb.d
    - ./mongo/data:/data/db
  ports:
    - 27017:27017
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: root
    MONGO_INITDB_DATABASE: smart_bed
  command: mongod --auth
```

2.Funcionamiento

Explicaremos a continuación de forma compacta y resumida el funcionamiento del producto completo. Para hacerlo nos centraremos en el flujo de la información de modo que comenzaremos por cómo se capta y terminando en cómo ésta llega al usuario.

Arduino->Raspberry Pi-> API Rest -> MongoDB -> API Rest -> Web

Capturamos los datos de los sensores mediante la Arduino Due, los filtramos y pre procesamos mediante distintos tipos de filtro según las necesidades de cada sensor. En algunos utilizamos técnicas más avanzadas como la apertura sintética o la configuración adaptativa. La Arduino Due muestra el estado de los sensores de flexibilidad en los leds mediante pwm y el del sensor que hayamos elegido con los interruptores se lo envía a la Arduino Uno. La Arduino Uno mostrará los datos que recibe por la interfaz serie en los displays. Finalmente, la Arduino Due también publica las medidas de los sensores en formato JSON por dos interfaces Serie, una de ellas conectada a un módulo bluetooth.

La Raspberry PI se conecta a una de las interfaces serie en las que se están publicando los datos, esto lo puede hacer por cable o por bluetooth. Finalmente procesa esos datos decidiendo la postura del usuario en la cama o la calidad del sueño. A los datos procesados se les añade el nombre de usuario y una marca de tiempo tras lo cual son enviados por peticiones POST a la API REST.

En la API REST un servidor express recibirá las peticiones. De las peticiones se extrae su contenido y se valida contra un modelo mongoose. Si la validación es correcta los datos se enviarán entonces a una base de datos MongoDB. Tanto el servidor como la base de datos están en contenedores docker en un servidor cloud alquilado como IAS.

Los usuarios utilizarán su navegador para acceder al servidor que en su puerto 80 tiene un proceso que corre sobre otro contenedor docker y cuya función es proporcionar a los usuarios con la página web que soliciten.

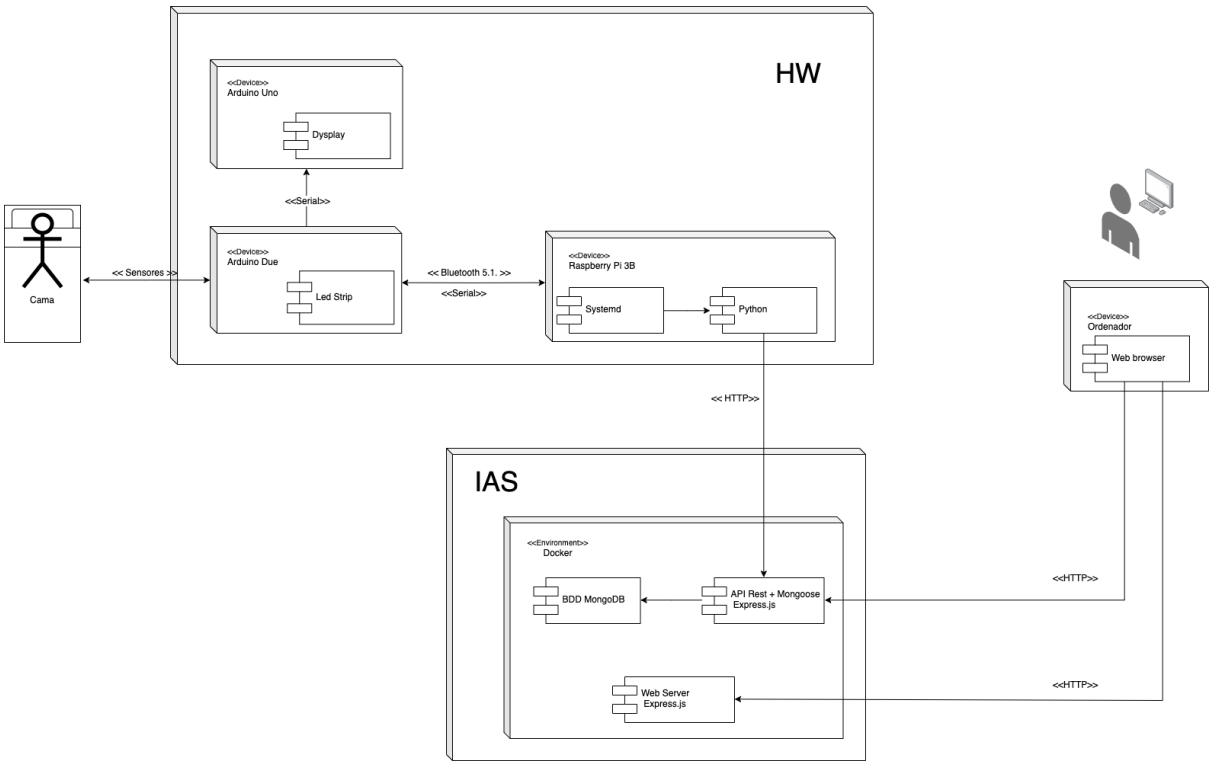
La web se encargará de mostrar los datos almacenados de una forma visual y sencilla al cliente, para ello accede a los datos que se han recogido y almacenado en la base de datos con la API. Una vez el servidor web tiene los datos los procesa para alimentar en su mayoría a los gráficos que se realizan con Chart.js.

Raspberry Pi <- API Rest <- MongoDB<- API Rest <- Web

La comunicación es bidireccional por lo que el usuario puede interactuar con su propio hardware desde la web. Los usuarios pueden configurar alarmas para que suenen a la hora y día que él desee.

La fecha de la alarma es enviada desde la web a la API REST que la almacena en la base de datos. La RaspberryPI pregunta periódicamente por las alarmas de su usuario y hace sonar el altavoz conectado a ella en caso de que sea necesario.

Adicionalmente los usuarios para entrar a la web deben identificarse para lo cual introducen su nombre y contraseña que son contrastados con los almacenados en la base de datos para comprobar que existen. Esto se realiza preguntando a la API REST por la existencia de un usuario.



3. Mejoras

A lo largo del proyecto ha habido algunas características que nos hubiera gustado incorporar o que hemos visto he proporcionarán funcionalidad de interés. Ya que uno de nuestros objetivos si no el principal es aprender hemos explorado estas características y las exponemos ahora. En caso de que el proyecto fuera a continuar tras esta entrega estos serían los siguientes puntos en los que deberíamos de trabajar.

3.1. Mejora del dashboard

El dashboard es el núcleo de la aplicación web y se caracteriza por su modularidad, de tal forma que es un compuesto de distintos módulos. Al tener este diseño es muy fácil introducir nuevas mejoras y cambios, entre ellos hemos pensado añadir un sistema para poder mover, ampliar, cerrar y minimizar estos módulos dentro del dashboard. Además de añadir futuros módulos para realizar más cálculos con los datos como estimar calorías diarias, fases del sueño del usuario y cálculos para la hora óptima de ir a dormir.

3.2. Encriptación de datos en la API

Al tratar con datos de carácter sensible, cuando el usuario quiera acceder a ellos desde la web, los recibirá cifrados, de tal forma que si se interceptan por un atacante no pudiera hacer nada. La encriptación se realiza con clave pública y privada de tal forma que los usuarios y servidores posean las claves.

3.3. Autorización y autenticación con OAuth 2.0

Un aspecto de mucha importancia en el proyecto es el de crear un sistema seguro, para ello tenemos como objetivo implementar los mejores mecanismos de seguridad. Por eso implementaremos OAuth en nuestra API para realizar autenticación y autorización, porque se trata de un estándar abierto que permite proteger las credenciales de los usuarios además de acceder a ellos. Se utiliza por grandes empresas como Google, Facebook o Microsoft.

3.3. Mejoras en la alimentación

La forma en la que alimentamos las placas controladoras y la Raspberry PI es buena pero no ideal. Nos hemos centrado en las ventajas que nos proporciona tener una fuente de alimentación externa lo cual realmente es más barato y más fiable que utilizar baterías siendo siempre recomendable cuando dispongamos de una toma de corriente.

No obstante, aunque la fuente de alimentación elegida es buena debido a la seguridad que implementa esta carece de potencia suficiente para alimentar nuestro sistema. Adicionalmente ahora la raspberry y las placas controladoras se alimentan por separado lo cual tampoco es deseable.

Sería recomendable aumentar la potencia de la alimentación hasta 5A 5V de pico para poder cubrir nuestros requisitos actuales de potencia que están cerca de los 22W.

3.4. Añadir más sensores de flexibilidad

Está claro que con solo cinco sensores de flexibilidad no podemos cubrir el ancho de ninguna cama. No obstante, debido a que lo implementado era un prototipo y que este tipo de sensores no es barato precisamente hemos decidido reducir su número a esta cantidad. Para cubrir una cama entera harían falta entre 20 y 30 sensores que para poder ser utilizados en nuestro hardware actual habría que multiplexar para poderlos leer.

3.5. Incorporar websockets

Utilizar una API REST nos ha permitido desarrollar más rápido debido a que hay gran cantidad de aplicaciones que nos permiten realizar peticiones de forma manual. No obstante, tiene sus limitaciones como recurrir a polling para saber cuándo hay nuevas medidas de ciertos valores a los que nos queremos suscribir.

Utilizar otros protocolos como MQTT o websockets nos permitirían implementar canales de comunicación permanentes mediante los que realizar esta subscripción. Utilizar MQTT ampliamente aceptada, utilizada y probada. No obstante, para nuestro caso concreto creemos que utilizar websockets sería una solución igualmente buena.

Mediante websockets la raspberry podría suscribirse a las alarmas de su usuario para saber cuándo este las modifica y así saber cuándo hacer sonar la alarma. Adicionalmente la web podría conectarse al servidor para obtener sin recurrir a polling los valores actuales de los sensores y poderlos mostrar.

3.6. Enlazar al usuario con su hardware

Actualmente sabemos qué dispositivo pertenece a qué usuario pues incorporamos el nombre de usuario en el código de la raspberry como un String. Esto no es buena praxis pues indica que un usuario

tendrá solo un dispositivo asignado. Además, decirle al usuario que si cambia su nombre debe cambiarlo en un archivo del código de su dispositivo resulta algo poco adecuado.

Como solución proponemos que cada dispositivo cuente con un identificador único visible para el usuario en forma de QR. Cuando un usuario compre un dispositivo podrá capturar este QR enlazándolo con su cuenta. Ya que ahora el nombre del dispositivo y del usuario es distinto y que el enlace entre ambos se reflejará como una relación en la base de datos nada nos impide que un usuario pueda tener más de un dispositivo enlazado con su cuenta.

3.7. Permitir al usuario seleccionar intervalos de tiempo

En el estado actual de nuestra web, se muestran en las gráficas los últimos 24 datos recogido por los sensores. Una de las opciones que se plantean para una posible mejor es permitir al usuario observar los datos de las últimas 24 horas, de la semana anterior o de un día concreto, permitiendo esto ver una evolución más progresiva y prolongada en los datos del usuario.

4. Acceso al producto

GitHub del proyecto:

https://github.com/JuanCasado/COMPUTACION_UBICUA

Página web con la dashboard:

http://163.172.80.168/

- Usuario: gato | Contraseña: gato
- Usuario: SmartBed1 | Contraseña: SmartBed1