

Instituto Tecnológico de Estudios Superiores Monterrey

CAMPUS QUERÉTARO

Implementación de internet de las cosas

Pedro Nájera García, Sergio Eberth Manjarrez Quintanilla y Rodrigo Sánchez Luna

Grupo 1

Documentación 2 propuesta de proyecto.

PRESENTA:

Ariann Fernando Arriaga Alcántara - A01703556

Andrés Acevedo Caracheo - A01706897

Pablo César Jiménez Villeda - A01703517

Alan Fernando Razo Peña - A01703350

Fecha:

5 de diciembre de 2021

<b>Introducción:</b>	<b>3</b>
Propuesta:	3
Clientes potenciales:	4
Equipo:	4
<b>Implementación</b>	<b>5</b>
Recopilación de datos	5
Node MCU	5
Sensor de temperatura	7
Conexión a internet	8
Backend	9
Firebase Real Time Database	11
Node JS	12
Endpoints	15
Frontend - visualización y monitoreo	19
<b>Resultados</b>	<b>22</b>
Repositorio del servidor y página web:	22
Video:	23
<b>Bitácoras</b>	<b>23</b>
<b>Referencias</b>	<b>23</b>

## **Introducción:**

El internet de las cosas es un proceso que permite conectar objetos ordinarios al internet. Estos objetos pueden ser desde termostatos y bombillas, hasta recursos para la atención de salud. Los objetos reciben y transmiten datos a través de redes inalámbricas con poca intervención humana. Estos datos después pueden ser analizados, tanto por el usuario como por el objeto, para tomar algún tipo de acción o realizar algún proceso.

El transporte de órganos suele ser un proceso con muchas complicaciones y variables que pueden afectar la integridad de un órgano, sin embargo, debido a la urgencia de la mayoría de los casos estas variables no son monitoreadas de forma adecuada. Por tanto, se debe de planificar previamente la manera en que el órgano llegará a su destino de forma segura y rápida debido a que dependiendo del órgano estos tienen un tiempo límite que pueden permanecer funcionales fuera del cuerpo humano. Por lo general, si las distancias no son tan largas, se hacen transportes en coche y en casos de largas distancias se hacen viajes en tren o en avión. Este proceso es muy delicado y deben de tomarse muchas medidas de precaución para que el órgano llegue en las mejores condiciones posibles. Se debe de cuidar de movimientos bruscos, debe de estar en un ambiente esteril, debe de estar en una temperatura adecuada y no puede estar mucho tiempo fuera del cuerpo humano. A lo largo de este proceso tanto los médicos en el hospital que recibirá el órgano como quienes lo transportan constantemente se encuentran en apuros al no saber con certeza el estado del órgano y si las condiciones son las adecuadas, por lo que el internet de las cosas es una herramienta perfecta para la construcción de un dispositivo de monitoreo portátil que se encargue de eliminar la interrogante del equipo médico que se encarga de transportar los órganos y de apaciguar las preocupaciones de quienes esperan el órgano, pues usando el stack del internet de las cosas se pueden sensar estas variables y enviar la información a la nube para que pueda ser monitoreada mediante una interfaz gráfica.

## **Propuesta:**

Dadas las características del internet de las cosas y las necesidades actuales en la medicina se ha encontrado que una de las grandes áreas de oportunidad en las que se puede emplear esta herramienta es en el transporte de órganos, pues hoy en día los médicos que solicitan el transporte de un órgano no tienen ninguna forma de monitorear el estado y la integridad del mismo mientras esperan a que llegue al hospital en el que se encuentran para

realizar un trasplante. Las necesidades que surgen a partir de este problema podrían ser claramente resueltas con el internet de las cosas y el prototipo propuesto que se busca implementar haría uso del stack propuesto por el internet de las cosas. A grandes rasgos, la colección de datos será realizada mediante un sensor de temperatura que monitoree el bienestar del órgano en cuestión que debe de estar refrigerado a 4°C, un NodeMCU recibirá los datos de temperatura y se conectará a internet usando el protocolo MQTT con el que registrará la temperatura en una base de datos hosteada en firebase, por último, el monitoreo de la información será realizado en una página web a la que el doctor en cuestión que busque monitorear el estado del órgano se podrá conectar.

### **Clients potencials:**

Los clients potencials serien hospitals, o consultorios de doctores, que realitzan trasplantats de òrgans o que manejan materials delicats els quals requereixen una temperatura precisa per així poder mantenir un bon estat. Esto els permetria tenir un major control sobre dichos materials i prendre acció immediata en el cas de algun problema.

### **Equip:**

Per poder formalitzar la venda d'este tipus de productes seria necessari crear una entitat real que se encarregue de la venda de los dispositivos propuestos, per això tomando en compte a los clients potencials i el tipus de aplicació que se tendrà que desenvolupar se usará el nom de CodeMed, el qual per un costat fa referència a los clients que pertenecerán al àrea de la medicina i per l'altre costat fa allusió a una de les eines usades en la fabricació del prototip que es la programació.



### **Lema:**

Soluciones sustentables para problemas interconectados

### **Visió:**

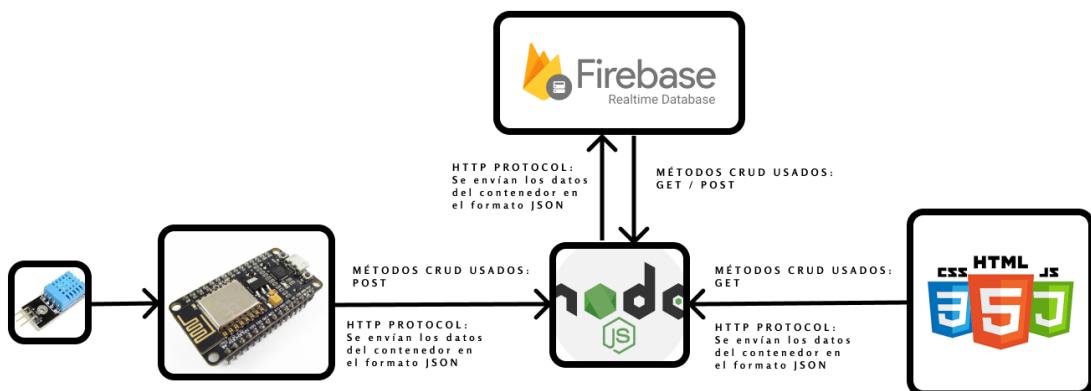
Ser la empresa de desarrollo de software sostenible más relevante de latinoamérica, generando las soluciones más eficientes para el bienestar de la sociedad.

### Misión:

Crear soluciones de software sostenible usando tecnologías innovadoras como el internet de las cosas, e implementarlas en el sector público de salud.

## Implementación

Para la elaboración de un proyecto de tal amplitud se asignan diferentes roles entre los integrantes del equipo de acuerdo a las especializaciones e intereses de cada uno logrando así que cada uno explorará a fondo una de las partes del stack del internet de las cosas.

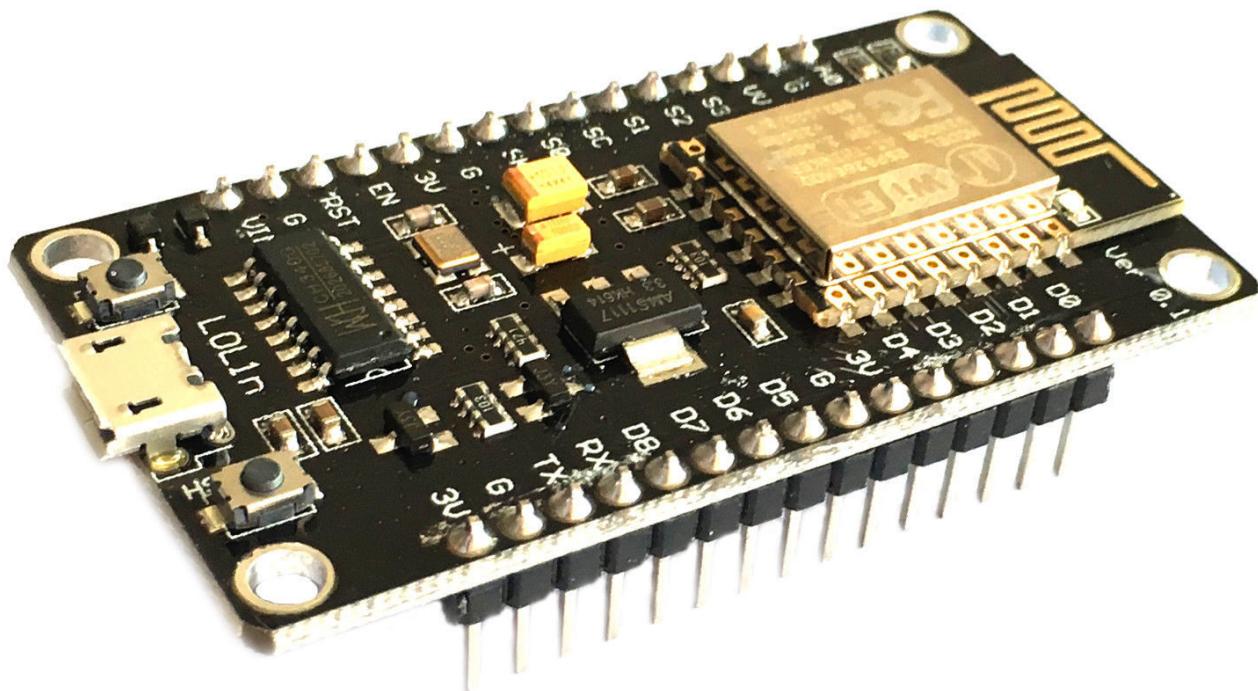


### Recopilación de datos

Para la creación del prototipo se utilizó un protoboard en el que se colocaría un chip NodeMCU y un sensor de temperatura, así como un led que alertará en cuanto la temperatura sea menor que la deseada para el órgano.

### Node MCU

Para la realización de este prototipo de contenedor para órganos se hará uso de un NodeMCU v3 con WiFi.



## Funcionamiento de NodeMCU ESP8266

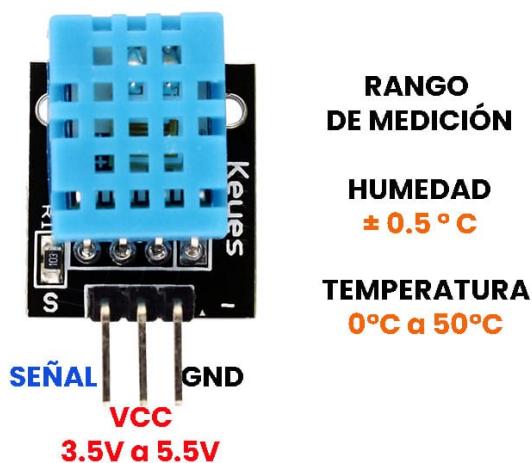
Pines	Función
A0	Entrada que recibe una señal analógica de un sensor.
G	Es una referencia de tensión de 0 voltios conocida como tierra.
3V	Es una salida de 3.3 voltios que sirven para energizar elementos externos.
D0-D8	Entradas digitales por las que entra un bit, el máximo número de bits que puede recibir este modelo es de 9.
RX/TX: Son dos pines usados para descargar código de un PC.	RX: Es un pin que recibe datos de una PC.
	TX: Es un pin que transmite datos a una PC.
SK	Señal de reloj usada para sincronizar el NodeMCU con otros iguales.
Vin	Entrada de suministro de poder externo.

Micro-usb port	Puerto de entrada de energía de 5 voltios que se puede conectar a un ordenador.
EN	El chip ESP8266 se activa cuando este pin se energiza con un high y cuando está en low trabaja al poder mínimo.
RST	El pin RST como su nombre lo indica es usado para resetear el chip.
SDIO	Estos pines son conocidos como Secure Digital Input/Output interfaz y son usados para conectar directamente tarjetas SD. (SD0, SD1, SD2, SD3)
SC	El pin SC es conocido como SPI y es un chip select o slave select que es utilizado para seleccionar uno o un grupo de circuitos integrados de varios conectados a un bus de computador, es un pin de comando en que conecta los pines de E / S del dispositivo a los circuitos internos de ese dispositivo.
VU	El pin VU (Vusb) es un pin de salida para energizar elementos externos que está conectado a + 5V en el bus USB del socket del NodeMCU. Este pin se puede usar para alimentar componentes externos cuando está conectado a USB, dentro de los límites de la fuente de alimentación USB a la que está conectado, es útil para alimentar módulos de relé de 5 V.

### Sensor de temperatura

También se utilizó el sensor de temperatura y humedad como se describe a continuación:

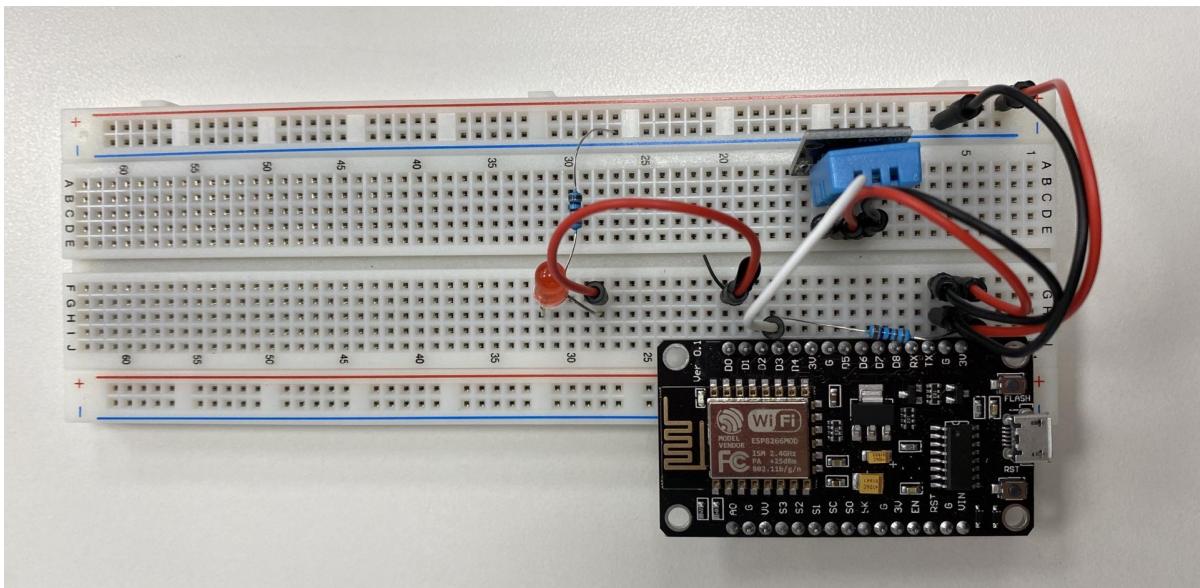
#### **PINOUT Sensor De Temperatura Y Humedad Módulo KY-015**



### Funcionamiento del DHT11:

Pines	Función
Señal	Entrada que recibe una señal analógica de un sensor.
VCC	Es una salida de 3.5 a 5.5 voltios que sirve para energizar elementos externos.
GND	Es una referencia de tensión de 0 voltios conocida como tierra.

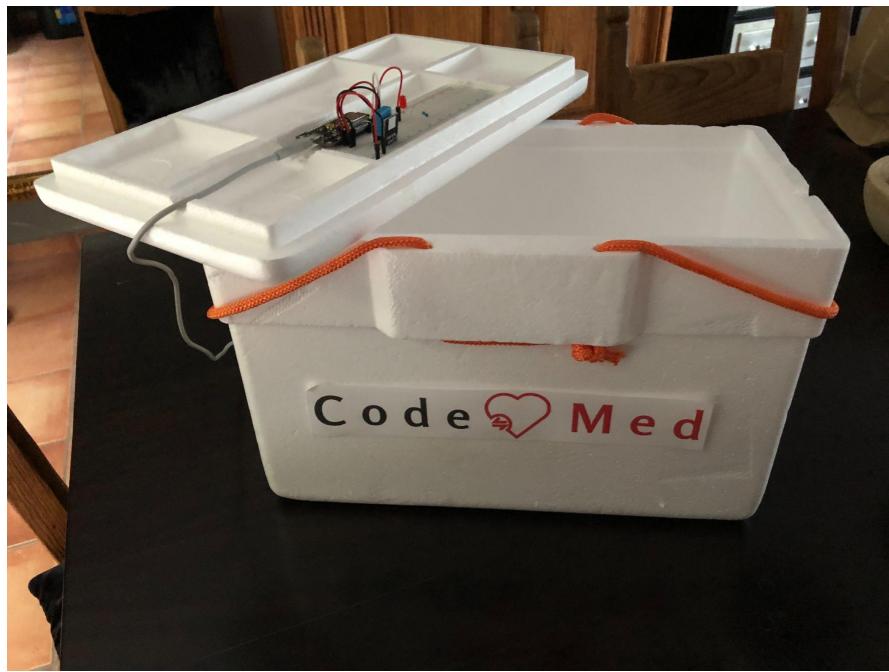
### Conexiones del Node MCU



Se energiza el sensor a través del pin 3V ya que es una salida de 3.3 voltios que sirven para energizar elementos externos. También sus demás conexiones están en G una referencia de tensión de 0 voltios conocida como tierra y en la señal que se manda al pin D3 el cual sirve como entrada digital de un bit que mandará los datos a través del chip ESP8266 con conexión Wi-Fi.

Adicionalmente se agregó un sensor led con resistencia en el pin D0 para poder indicar si el nivel de temperatura traspasa cierto rango.

Las conexiones se hicieron lo más cercanas posibles al Nodemcu para que no hubiera fallas y el cableado fuera fácil y rápido de distinguir.



El NodeMCU solicita los datos de temperatura recopilados por el sensor cada 7 segundos. Además de enviar estos datos a nuestro servidor, también dentro del código del NodeMCU se checa que la temperatura sea ideal para los órganos transportados. En caso de que no lo sea, se prenderá el LED conectado. Esto se realizó para seguir teniendo feedback de la temperatura, aun cuando el NodeMCU no esté conectado al internet.

```
void loop(){
    temp = dht.readTemperature();
    Serial.println("Temperatura: " + String(temp));
    if(temp > 4){
        digitalWrite(D0, HIGH);
    }
}
```

## Conexión a internet

En nuestra implementación nosotros utilizamos el protocolo http para conectar el NodeMCU al internet. Http es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor. Clientes y servidores se comunican intercambiando mensajes individuales. Los mensajes que envía el cliente, en este caso el NodeMCU, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

Para conectar el NodeMCU a la red, dentro del setup(), configuramos el módulo utilizando las librerías, ESP8266WiFi.h y ESP8266WiFiMulti.h, e iniciamos la conexión pasando el nombre de red y contraseña del punto de red portátil de nuestro celular.

```

void setup() {
  Serial.begin(115200);
  dht.begin();
  delay(1000);

  //Inicio conexión a WiFi
  WiFi.mode(WIFI_STA);
  wifiMulti.addAP("Iot_prueba", "Prueba12");
  int cont = 0;
}

```

Una vez que verificamos que la conexión se realizó correctamente, utilizando el protocolo http configuramos el NodeMCU como cliente y realizamos una petición de “POST”, para realizar un post inicial en “<https://code-med-iot.herokuapp.com/>”. Este post inicial solo se realizará dentro del primer ciclo loop y creará un “contenedor” nuevo, en donde se empezaran a guardar los datos de temperatura que se registren en los loops siguientes.

```

if((wifiMulti.run()==WL_CONNECTED)){
  if(cont == 0){
    //Post Inicial(Creando contenedor nuevo)
    StaticJsonBuffer<200> jsonBuffer;
    WiFiClient client;
    HTTPClient http_postinicial;
    Serial.println("[HTTP] Iniciando...");
    http_postinicial.beginClient(post);
    http_postinicial.print("HTTP/1.1");
    http_postinicial.addHeader("Content-Type", "application/json");
    http_postinicial.POST("{\"contenido\": \"Corazón\", \"from\": \"Hospital general de Huichapan\", \"to\": \"Hospital San José\", \"temperaturaOptima\": \"4\", \"duracion\": \"8\"}");

    //Recibimos una respuesta del servidor con un ID unico que modifica la liga original, para realizar los posts siguientes.
    String respuesta = http_postinicial.getString();
    Serial.print("respuesta:");
    JsonObject& root = jsonBuffer.parseObject(respuesta);
    const char* id_char = root["id"];
    String id_string(id_char);
    post.concat(id_string);
    http_postinicial.end();
    cont = 1;
  }
  else{
    //Post de temperatura dentro de contenedor
    WiFiClient client;
    HTTPClient http;
    Serial.println("[HTTP] Iniciando...");
    http.beginClient(post);
    http.print("HTTP/1.1");
    http.addHeader("Content-Type", "application/json");
    http.POST("{\"temperatura\":"+String(temp)+"}");
    String respuesta = http.getString();
    Serial.print("respuesta:");
    http.end();
  }
}

```

A partir de este POST inicial el servidor generado retornará una llave de identificación del contenedor donde se almacenaron los datos pasados como argumento en la primera iteración de este ciclo. Esta respuesta retornada por el servidor es un string y tiene la siguiente forma:

```

Status: 200 OK  Size: 97 Bytes  Time: 7.62 s
Response Headers  Cookies  Test Results  []
1 = []
2   "status": "success",
3   "message": "The container was added successfully",
4   "id": "-MprQKIogexkB4L0NcCg"
5 []

```

Esta respuesta la convertimos en un objeto de tipo JSON. JSON es un formato de texto sencillo de JavaScript que sirve para el intercambio de datos. Este formato nos permite separar el texto y solamente obtener la parte con el nombre “id”. Posteriormente, se accederá a un nuevo endpoint que consistirá en la concatenación de la liga “<https://code-med-iot.herokuapp.com/>” y el “id” generado. A este nuevo punto se enviarán nuevos POST con la información de la última temperatura recopilada por el sensor.

## Backend

Durante la exploración de posibles implementaciones de medios de almacenamiento de los datos monitoreados se llegó a la conclusión de que la mejor forma de implementar esta parte del stack sería creando un servidor usando la herramienta node js y el framework express para servir al cliente la página web para el monitoreo de datos y poder responder a los pedidos de información del node MCU a través del protocolo http.

En primer lugar, para almacenar la información se usó una base de datos en tiempo real en firebase, la cual es una forma de recopilar información de forma estructurada y de forma electrónica. En estas se pueden realizar distintos tipos de operaciones como consultas e inserciones de datos. La base de datos en tiempo real de firebase es no relacional orientada a documentos, la principal diferencia es cómo se estructuran y se relacionan los datos:

- **RELACIONAL:** una fila no puede tener varios datos, es decir, se tienen que crear tablas adicionales si un atributo tiene más de un valor.
- **NO RELACIONAL:** estas bases de datos pueden tener más de un valor en cada atributo evitando la necesidad de crear tablas o relaciones adicionales.

Además, esta es una base de datos orientada a documentos, lo que representa la alternativa más popular en las aplicaciones modernas. Pues su estructura es fundamentalmente la misma que la de JSON, el lenguaje de comunicación de datos más popular en la web a través del protocolo http. Fundamentalmente estas bases de datos consisten de arreglos y objetos, en los que un arreglo o colección tiene varios objetos que a su vez pueden tener colecciones. El principal motivo por el que se eligió esta bases de datos es que los documentos insertados no tienen una estructura rígida como en los modelos relacionales, es decir, que si en algún momento los atributos de un elemento se expanden no existiría ningún conflicto, lo que es especialmente útil para este proyecto debido a que permitiría almacenar distintos tipos de datos de acuerdo al órgano que se transporte, pues aunque todos requieren de una temperatura estable algunos de ellos requieren del cuidado de otras variables como la humedad del contenedor o que haya poco movimiento del mismo

debido a su fragilidad. Gracias a esta elección se tendrá una mayor capacidad para extender la aplicación, sus funciones e incluso la cantidad de aspectos que se pueden monitorear de forma sencilla.

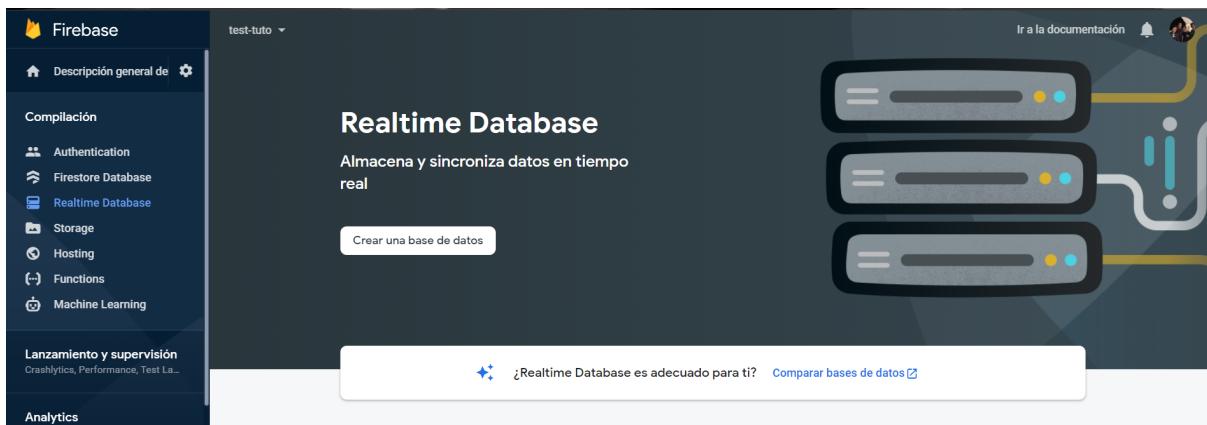
Para esta base de datos se eligió la plataforma firebase, la cual es una plataforma digital que se utiliza para facilitar el desarrollo de aplicaciones web o móviles de una forma efectiva, rápida y sencilla. Busca principalmente mejorar el rendimiento de la aplicación, para que sea más manejable, segura y de fácil acceso para los usuarios. Esta plataforma permite el hosting de páginas web, bases de datos, servidores e incluso aplicaciones de machine learning aprovechando la amplia infraestructura que Google ha desarrollado alrededor del mundo a través de los años. Las principales ventajas de esta plataforma son:

- Capacidades de base de datos.
- Diferentes servicios y funcionalidades.
- Es gratuito.
- Documentación concisa.
- Rápida y fácil integración.

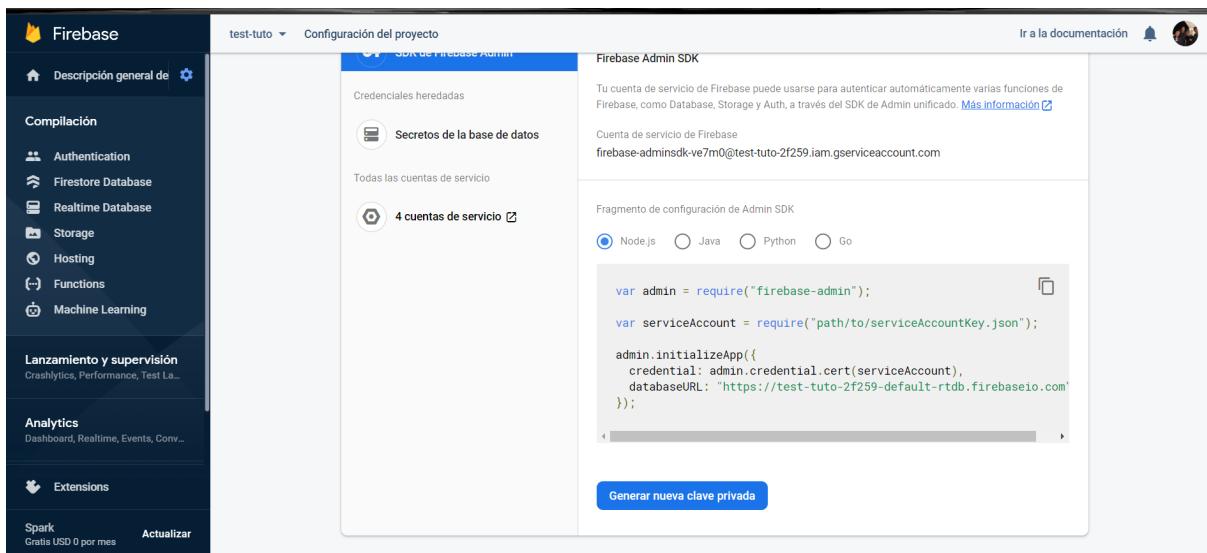
Esta plataforma cuenta con dos formas de implementar una base de datos, la primera siendo firestore la cual es una opción reciente que mejora ciertos aspectos de la base de datos en tiempo real pero que aún sigue siendo mejorada por los ingenieros de google. Por otro lado, la base de datos en tiempo real es la opción más popular gracias a la extensa documentación y material de apoyo que existe acerca de ella gracias a su longevidad. La principal ventaja de esta última es que filtrar y ordenar un pedido usando un sólo query a diferencia de su contraparte. Lo que es especialmente útil porque se busca que la aplicación de monitoreo desplegada en la web recopila la información de la última lectura cada cierto tiempo para mantenerse actualizada con las últimas actualizaciones del contenedor.

## **Firestore Real Time Database**

Para hacer uso de este servicio lo que se hizo fue en primer lugar crear un nuevo proyecto dentro de firebase y posteriormente se eligió crear base de datos en tiempo real:



Posteriormente, para asegurar la seguridad de acceso a esta información se generó una llave de autenticación única en forma de un archivo JSON el cual contiene las credenciales necesarias para acceder a la base de datos y realizar distintas operaciones con ella. Para esto se accedió a configuración y en el apartado de cuentas de servicio se generó una nueva clave privada que sería usada para acceder a la base de datos:



## Node JS

Node.js es un entorno de ejecución de código de JavaScript fuera del navegador que junto a su administrador de librerías npm nos permite crear aplicaciones complejas y escalables de forma sencilla. Este entorno está diseñado para crear aplicaciones orientadas a eventos asíncronos, es decir, que no bloqueen la ejecución del código como un servidor que recibe múltiples peticiones de forma concurrente. Node abstrae el concepto de hilos haciendo más sencillo crear aplicaciones con operaciones complejas en las que automáticamente Node delegará ciertas operaciones a otros hilos para seguir cumpliendo con su principal principio que es el de crear código que no se bloquee.

Aunque se pueden realizar servidores usando la librería standard de node “http”, para facilitar el acceso a la vista de la aplicación se usará el framework de desarrollo de servidores conocido como Express, el cual es el entorno de desarrollo más popular para crear aplicaciones complejas de forma rápida y sencilla. La ventaja de Express es que permite estructurar de una mejor manera las rutas o endpoints a las que se puede acceder a través de la creación de middlewares que reciben dos objetos, uno de petición y uno de respuesta, los cuales pueden ser modificados de distintas formas dependiendo la ruta a la que accedan. La ventaja de esto es que permite mantener un mayor control de las operaciones que interceptan ambos objetos antes de que se envíe una respuesta al cliente. A pesar de que es minimalista, se han implementado librerías que permiten implementar cookies, sesiones, inicio de sesión de usuario, parámetros URL, datos POST, cabeceras de seguridad, etc.

Para realizar este servidor lo primero es descargar la versión más reciente de Node.js y en el repositorio elegido crear una aplicación usando el comando “npm init”, el cual permitirá crear una aplicación de node, posteriormente se debe instalar express mediante el comando “npm i express”. En adición a estos paquetes se hará uso de morgan, una librería que permite visualizar en tiempo real las peticiones siempre y cuando se encuentre en una versión de prueba y dotenv para definir las configuraciones que rigen el funcionamiento de la aplicación.

Una vez hecho esto se definieron las siguientes configuraciones y rutas:

```
const express = require('express');
const morgan = require('morgan');

const app = express();

// ROUTERS
const containersRouter =
require(`__dirname/routes/containersRoutes`);

// MIDDLEWARES
if (process.env.NODE_ENV === 'development') {
    app.use(morgan('dev'));
}

app.use(express.json());
// Cuando se accede a un archivo al usar este middleware definimos que el
// archivo a retornar se encuentra en el directorio public desde el
// directorio
// actual
```

```

app.use(express.static(`${__dirname}/public`));

// USING ROUTERS
// Usamos el middleware containersRouters cuando se accede a la
// extensi n /api/contenedores/
app.use('/api/contenedores/', containersRouter);

module.exports = app;

```

Posteriormente, se cre  una aplicaci n de express desde un archivo index.js, el cual ser  ejecutado cada vez que se reinicie el servidor para probarlo:

```

const dotenv = require('dotenv');

// va a hacer que lea del archivo y las guarde en las variables
// de entorno
dotenv.config({ path: './config.env' });

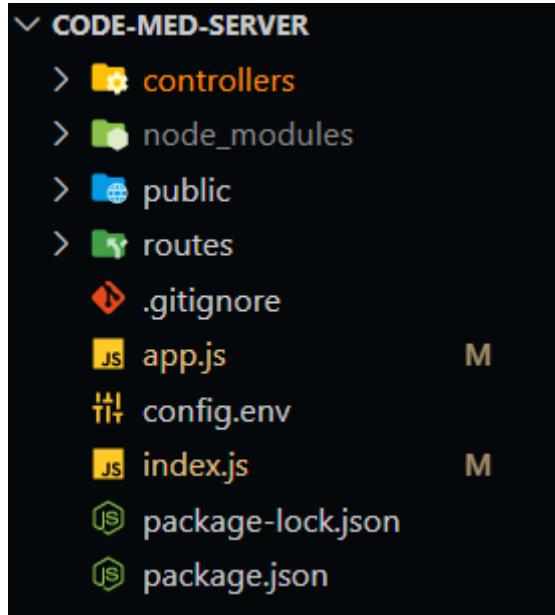
// requerir el c digo va a ir despu s porque primero queremos
// tener bien definidas las variables
const app = require(`${__dirname}/app.js`);

// detr s de escenas heroku le va a dar un puerto
const port = process.env.PORT || 3000;
// iniciamos el servidor - puerto en el que va a salir
app.listen(port, () => {
    //callback en cuanto se inicia
    console.log(`Server running on ${port}...`);
});

```

La estructura de este servidor se basa en la arquitectura Modelo-vista-controlador, en la que se clasifican los componentes de la aplicaci n en tres grupos:

- **Modelo:** componentes que gestionan el acceso a la informaci n.
- **Controlador:** componentes que responden a eventos.
- **Vista:** presenta el modelo en una forma en la que se pueda interactuar con la informaci n.



- Dentro de la carpeta public se guardarán las vistas que serán desplegadas cuando se pide acceso a algún archivo.
- Dentro de la carpeta controllers se almacenan las funciones que responden a las peticiones del servidor.
- Dentro de la carpeta routes se definen las rutas de acceso a la información, es decir, los modelos que regulan el acceso a la información.

## Endpoints

Un Endpoint o punto final de comunicación es una interfaz expuesta por un canal de comunicación, en el caso del servidor este se refiere a todas las rutas de acceso que se tienen en este y que pueden ser usadas por usuarios externos mediante una API. Dado que todas las operaciones se relacionan con la manipulación de la información de los contenedores de órganos se definió una única ruta llamada contenedores. Estas operaciones son exclusivamente para crear o actualizar datos, de acuerdo al tipo de petición se puede leer la información en ella, actualizar sus valores, y finalmente, si es que es necesario también borrarlos.

Dado que todas las operaciones se realizan a través de peticiones distintas rutas, para el alcance de este proyecto se definieron las siguientes dentro del archivo de containersRoutes:

```
const express = require('express');

// Se recuperan las funciones del archivo de controladores
const {
    getAllContainers,
```

```

postContainer,
getContainerInfo,
postRead,
getAllReads,
getLastRead,
} = require(`_${__dirname}/../controllers/containersController`);

// Se crea el router que engloba todos los middlewares
const router = express.Router();

// Se definen los middlewares que actuarán sobre cada ruta dentro del router
router.route('/').get(getAllContainers).post(postContainer);
router.route('/:id').get(getContainerInfo).post(postRead);
router.route('/:id/allReads').get(getAllReads);
router.route('/:id/lastRead').get(getLastRead);

// Se exporta el router
module.exports = router;

```

Implementación de las funcionalidades de cada endpoint:

```

const admin = require('firebase-admin');

const credentials =
require(`_${__dirname}/med-code-iot-firebase-adminsdk-okkub-1a9094ffc2.json`);

admin.initializeApp({
  credential: admin.credential.cert(credentials),
  databaseURL:
'https://med-code-iot-b579d-default-rtdb.firebaseio.com/',
});

const db = admin.database();

const containersRef = db.ref('contenedores');

// Se usa para obtener todos los contenedores
exports.getAllContainers = (req, res) => {
  // En la referencia pedimos todos los contenedores
  containersRef.once('value', (snap) => {
    // devolvemos una respuesta exitosa con los datos
  });
}

```

```

        res.status(200).json({
            status: 'Success',
            message: 'All containers were successfully retrieved',
            data: snap.val(),
        });
    });
};

// Se usa la primera vez que se enciende un contenedor
exports.postContainer = (req, res) => {
    // Creamos el contenedor con los datos del request
    const newContainer = {
        ...req.body,
        inicio: Date.now(),
    };

    // añadimos el contenedor con push para que tenga un id único
    const generatedKey = containersRef.push(newContainer).key;

    // ya creado devolvemos una respuesta exitosa
    res.status(200).json({
        status: 'success',
        message: 'The container was added successfully',
        id: generatedKey,
    });
};

// Se usa para obtener la información de un contenedor a partir de su
// información
exports.getContainerInfo = async (req, res) => {
    // obtenemos la referencia al hijo usando el parámetro de la url
    const oneContainer = containersRef.child(req.params.id);

    oneContainer.once('value', (snap) => {
        const information = snap.val();
        if (information) {
            // devolvemos una respuesta exitosa con los datos
            res.status(200).json({
                status: 'success',
                message: 'All containers retrieved successfully',
                data: snap.val(),
            });
        } else {
    
```

```

        res.status(404).json({
            // devolvemos una respuesta fallida
            status: 'failed',
            message: 'The container you were looking for was not
found',
        });
    }
}
);

// Se usa para subir una fecha
exports.postRead = (req, res) => {
    // añadimos nueva lectura al contenedor
    containersRef
        .child(req.params.id)
        .child('lecturas')
        .push({
            ...req.body,
            fecha: Date.now(),
        });
    res.status(200).json({
        status: 'success',
        message: 'New date successfully uploaded',
    });
};

// Se usa para obtener todas las lecturas del sensor
exports.getAllReads = (req, res) => {
    // obtenemos la referencia al arreglo de lecturas
    const oneContainerReads =
    containersRef.child(req.params.id).child('lecturas');
    oneContainerReads.once('value', (snap) => {
        const reads = snap.val();
        if (reads) {
            // devolvemos una respuesta exitosa con los datos
            res.status(200).json({
                status: 'success',
                message: 'All reads retrieved successfully',
                data: reads,
            });
        } else {
            // devolvemos una respuesta fallida
            res.status(404).json({

```

```

        status: 'failed',
        message: 'The reads could not be retrieved',
    ) );
}
} );
};

// Se usa para obtener la última fecha añadida
exports.getLastRead = (req, res) => {
    containersRef
        .child(req.params.id)
        .child('lecturas')
        .limitToLast(1)
        .once('value', (snap) => {
            res.status(200).json({
                status: 'success',
                message: 'The last read was successfully retrieved',
                data: snap.val(),
            });
        });
};
}
;

```

ENDPOINTS A PARTIR DE <a href="https://code-med-iot.herokuapp.com/">https://code-med-iot.herokuapp.com/</a>	DESCRIPCIÓN
/	Al acceder a la raíz el servidor servirá la página web y los archivos necesarios gracias al uso del middleware static provisto por Express js
/api/contenedores/	<ul style="list-style-type: none"> <li><b>GET:</b> usando el método get en este endpoint se obtiene la información de todos los contenedores registrados en firebase.</li> <li><b>POST:</b> usando el método post en este endpoint se crea un nuevo contenedor usando el cuerpo del mensaje enviado y se retorna la id generada por firebase.</li> </ul>
/api/contenedores/:id	<ul style="list-style-type: none"> <li><b>GET:</b> usando el método get en este endpoint se puede acceder a la información del contenedor cuya id coincide con el argumento de la url de la petición.</li> <li><b>POST:</b> usando el método post en este endpoint se agrega una nueva lectura usando el cuerpo del</li> </ul>

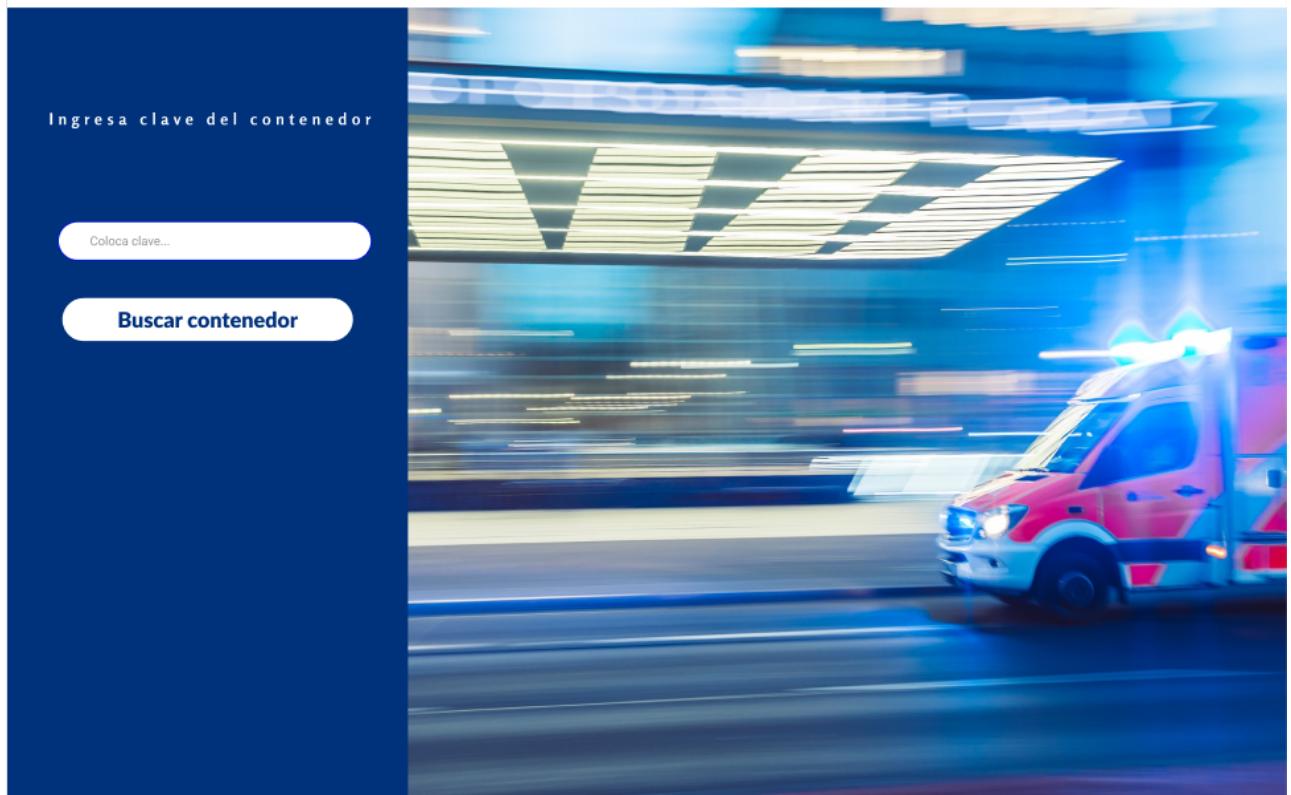
	mensaje para el contenedor cuya id coincide con el argumento de la url de la petición.
/api/contenedores/:id/allReads	<ul style="list-style-type: none"> <li>• <b>GET:</b> para este endpoint al usar el método get se obtienen todas las lecturas del contenedor que tiene el id incluido en el argumento de la url.</li> </ul>
/api/contenedores/:id/lastRead	<ul style="list-style-type: none"> <li>• <b>GET:</b> para este endpoint al usar el método get se obtiene la última lectura del contenedor que tiene el id incluido en el argumento de la url.</li> </ul>

### Frontend - visualización y monitoreo

El proceso de visualización y monitoreo inició con el desarrollo de la interfaz gráfica o visualización fue el realizar modelo de desarrollo en base a prototipos, dichos prototipos del frontend funcionan como versiones de control para así poder implementar la codificación de forma precisa y tener una idea de cómo se visualizará, los prototipos realizados fueron a través de la plataforma de figma, donde primero se realizó la página de inicio, o la interfaz de para autenticación de clave del contenedor, debido a que como se implementó nuestro sistema la autenticación, el usuario colocará la clave para así tener los datos en tiempo real del sensor de temperatura del contenedor que está conectado al MCU al igual que a la base de datos dinámica, El diseño de la primera interfaz consta de un fondo azul y una imagen relacionada al uso más práctico de nuestro producto el cual es una ambulancia donde los órganos serán trasladados, las proporciones de ambos componentes de la interfaz son: Fondo azul 30% del ancho y la imagen será 70% del ancho de la interfaz, de la misma forma tomamos la decisión de que cada interfaz o “ventana” tuviera un encabezado de color blanco en el cual se mostrará el logo de CodeMed basamos nuestros diseños en base a lo visto en la curso donde respetamos la composición de los elementos de la interfaz con un grid de 12 columnas donde las proporciones pueden ser visualizadas de mejor forma, la paleta de colores de esta interfaz de registro es de blanco y azul, esto se hizo en base a dos cosas, la primera es que son los colores de la institución del Tecnológico de Monterrey, y la segunda es porque son los colores más utilizados en cualquier institución de salud, esto se debe a que la teoría de colores expresa que el blanco y azul evocan sentimientos de paz y tranquilidad en las personas, lo cual es clave en sectores de salud.

El diseño de interfaz de registro de clave, implementó aparte de lo ya mencionado un campo de texto, una entrada de texto y un botón, siendo este el primer prototipo:

# Code Med

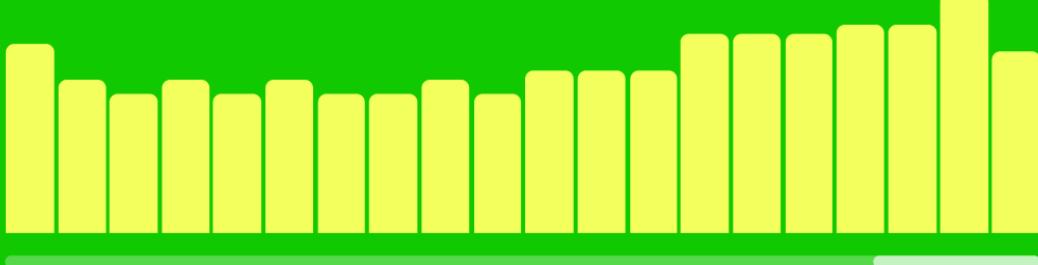


Lo segundo a desarrollar fue la interfaz que mostrará las temperaturas, esta interfaz hecha en figma presenta las temperaturas de forma dinámica, es decir, a parte de tener un contador el cual irá cambiando cuenta con un sistema de barras, las cuales se irán modificando dependiendo de las diferentes temperaturas que se registren en el contenedor, si la temperatura sale de los parámetros deseados entonces el background cambiará por completo, pasará de un color verde a rojo, esto señalando que hay urgencia para cambiar este estado.

El resultado del primer diseño de la interfaz de visualización de temperaturas es el siguiente:

# Code Med

Temperatura actual: 2°C



Temperatura optima: 5°C

Temperatura promedio: 5°C

Contenido de la hielera: 

Tiempo de viaje restante: 2:23:50



Hospital General de  
Huichapan Hidalgo

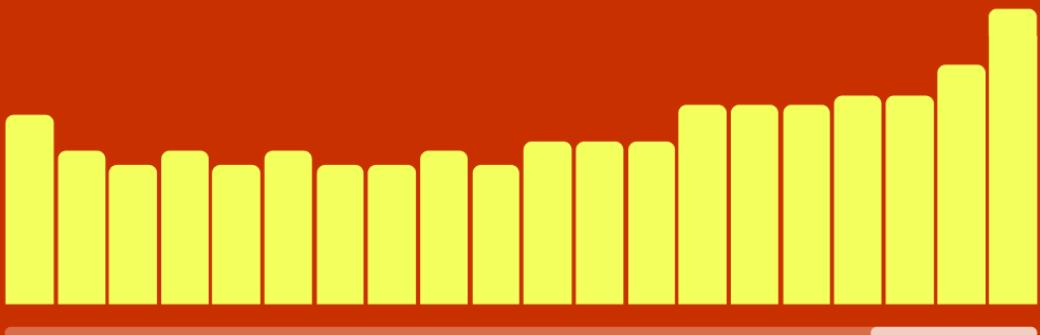


Hospital Star Médica

Los resultados son agregados en forma de barras en cuanto han sido registrados, después tenemos la representación del recorrido con dos hospitales los cuales son representados con dos íconos y sus nombres, y en medio decidimos agregar el ícono de un carro, La pantalla de cuando las temperaturas salen de los límites requeridos son los siguientes:

# Code Med

Temperatura actual: 7C



Temperatura óptima: 5C

Temperatura promedio: 5C

Contenido de la heladera: 7

Tiempo de viaje restante: 2:23:50



Hospital General de  
Huichapan Hidalgo



Hospital Star Médica

Básicamente los colores del background cambian por un tono rojo el cual indica que los registros de temperatura han salido de los límites establecidos por la temperatura óptima que el usuario, o en este caso indica el personal médico. La ventaja de utilizar figma para el prototipado y visualización de interfaz es la producción del css de los componentes, el css generado de estos componentes será después utilizado en los archivos scss los cuales se compilarán en un archivo main scss el cual será referenciado en el documento index HTML.

Cuando los prototipos son realizados, se intenta replicar el diseño de dichos prototipos en el archivo html el cual en este caso funciona como los cimientos de nuestra página web. Básicamente en html se importaron o utilizaron referencias para tener las fuentes y estilos de texto que necesitábamos para replicar la visualización de la página en figma, se trabajó el archivo html utilizando diferentes contenedores, al igual que la misma creación de clases para poder después cambiar su estética por medio del main scsss, cosas que tienen los contenedores son el texto de la temperatura óptima, o el texto de la temperatura actual, la gráfica donde el registro de las temperaturas puede ser visualizado de mejor manera por el usuario o personal médico está dentro de un contenedor, y cada barra es un contenedor de la clase lectura, la cual en el scss se definió sus dimensiones y color entre otros atributos. dicha clase gráfica esta dentro de otro contenedor de clase “scrollable” el cual le permite al registro

de temperaturas tener una barra de scroll para que el usuario visualice todas las temperaturas que se han registrado. La parte del código html de la página de lectura de temperaturas y su visualización en el navegador se muestra a continuación:

```
<div class="reads" id="charts">
    <h1 id='actual'>Temperatura actual: 2C</h1>
    <div id="scrollable" class="scrollable">
        <div id="grafica" class="grafica"></div>
    </div>
    <h3 id='optima'>Temperatura óptima: 2C</h3>
    <h3 id='promedio'>Temperatura promedio: 2C</h3>
    <h3 id='contenido'>Contenido de la heladera: corazón</h3>
    <h3 id='tiempo'>Tiempo última actualización: 2:23:50</h3>
    <div class="destinos">
        <div class="hospital_nombre">
            
            <p id='from' class="destino">Hospital de huichapan</p>
        </div>
        <div class="carretera">
            <div class="camino"></div>
            
        </div>
        <div class="hospital_nombre">
            
            <p id='to' class="destino">Hospital de huichapan</p>
        </div>
    </div>
</div>
```

# Code Med



Del mismo modo se implementó una parte que no estaba contemplada en el prototipo inicial, la cual es la parte de la página que habla sobre nosotros

```
<div class="about">
    <h1>THE TEAM</h1>
    <div class="teamContainer">
        <div class="teamMember">
            <h2>Alan Fernando Razo Peña</h2>
            
            <h2 class="role">Sensado de datos</h2>
            <p class="descripcion">Para la elaboración de un proyecto de tal an
        </div>
        <div class="teamMember">
            <h2>Andrés Acevedo Caracheo</h2>
            
            <h2 class="role">Conexión con internet</h2>
            <p class="descripcion">Programando el chip Node MCU y usando el módulo
        </div>
        <div class="teamMember">
            <h2>Pablo César Jiménez Villeda</h2>
            
            <h2 class="role">Backend</h2>
            <p class="descripcion">La mejor forma de implementar esta parte del sta
        </div>
        <div class="teamMember">
            <h2>Ariann Arriaga Alcántara</h2>
            
            <h2 class="role">Frontend</h2>
            <p class="descripcion"> El proceso de visualización y monitoreo inició
        </div>
    </div>
</div>
```

# THE TEAM

Alan Fernando Razo Peña



Sensado de datos

Para la elaboración de un proyecto de tal amplitud se asignan diferentes roles entre los integrantes del equipo de acuerdo a las especializaciones e intereses de cada uno logrando así que cada uno explorará a fondo una de las partes del stack del internet de las cosas.

Andrés Acevedo Caracheo



Conexión con internet

Programando el chip Node MCU y usando el módulo de http así como el de JSON, se pudo establecer comunicación entre el chip que crearía contenedores y actualizaría las lecturas de temperatura de los mismos a través de los métodos POST y GET a través del protocolo HTTP al servidor implementado en Node.

Pablo César Jiménez Villeda



Backend

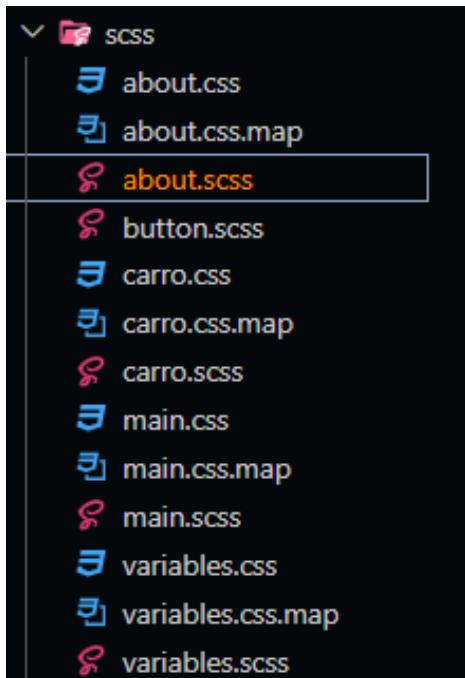
La mejor forma de implementar esta parte del stack sería creando un servidor usando la herramienta node.js y el framework express para servir al cliente la página web para el monitoreo de datos y poder responder a los pedidos de información del node MCU a través del protocolo http.

Ariann Arriaga Alcántara



Frontend

El proceso de visualización y monitoreo inició con el desarrollo de la interfaz gráfica en figma, esto con el objetivo de tener claro cómo hacer uso de el lenguaje de codificación html y scss y js para cumplir con los requerimientos funcionales de la aplicación web.



```
public > scss > main.scss > ...
1  @import './variables';
2
3  @import './button';
4
5  @import './carro';
6  @import './about.scss';
7
8  * {
9    margin: 0;
10   padding: 0;
11   font-family: 'Rosario';
12   box-sizing: border-box;
13 }
14
15 body {
16   height: 100vh;
17 }
18
19 .header {
20   height: 20%;
21   width: 100%;
22   background-color: #fff;
23   display: flex;
24   flex-direction: row;
25   align-items: center;
26   justify-content: center;
27
28   img {
29     width: 35%;
30     cursor: pointer;
31   }
32 }
33
```

```
.scrollable {
    width: 1050px;
    height: 180px;
    overflow-x: scroll;
}

.grafica {
    height: 100%;
    display: flex;
    flex-direction: row;
    align-items: flex-end;
    width: min-content;
    justify-content: flex-end;
    overflow: hidden;
}

.lecturaContenedor {
    display: flex;
    flex-direction: column;
    align-items: flex-end;
    justify-content: flex-end;
    height: 100%;
    width: min-content;
}

p {
    color: transparent;
    transition: all ease-out 0.5s;
}

&:hover {
    p {
        display: block;
        color: ■#ffffff;
        width: 100%;
```

```
427 .scrollable .grafica {
428     height: 100%;
429     display: -webkit-box;
430     display: -ms-flexbox;
431     display: flex;
432     -webkit-box-orient: horizontal;
433     -webkit-box-direction: normal;
434     ||| -ms-flex-direction: row;
435     ||| flex-direction: row;
436     -webkit-box-align: end;
437     ||| -ms-flex-align: end;
438     ||| align-items: flex-end;
439     width: -webkit-min-content;
440     width: -moz-min-content;
441     width: min-content;
442     -webkit-box-pack: end;
443     ||| -ms-flex-pack: end;
444     ||| justify-content: flex-end;
445     overflow: hidden;
446 }
447
448 .scrollable .grafica .lecturaContenedor {
449     display: -webkit-box;
450     display: -ms-flexbox;
451     display: flex;
452     -webkit-box-orient: vertical;
453     -webkit-box-direction: normal;
454     ||| -ms-flex-direction: column;
```

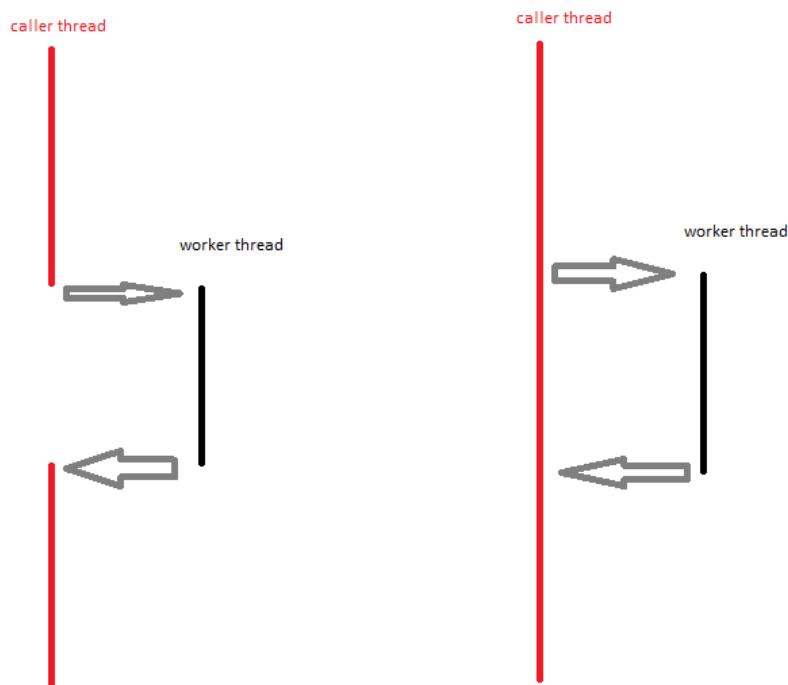
```

public > JS functions.js > [@] updateUI
1 // SELECCION DE ELEMENTOS
2 const form = document.getElementById('homepage');
3 const charts = document.getElementById('charts');
4
5 const logo = document.getElementById('Logo');
6 const input = document.getElementById('input');
7 const findButton = document.getElementById('find-button');
8
9 const graph = document.getElementById('grafica');
10 const actual = document.getElementById('actual');
11 const optimaH3 = document.getElementById('optima');
12 const promedioH3 = document.getElementById('promedio');
13 const contenido = document.getElementById('contenido');
14 const tiempo = document.getElementById('tiempo');
15
16 const to = document.getElementById('to');
17 const from = document.getElementById('from');
18

```

Async/ await

Synchronous



```
const getContainer = async (event) => {
    try {
        event.preventDefault();
        id = input.value;
        let endpointDirection = apiEndpoint + id;

        const resContainer = await fetch(endpointDirection);
        const dataContainer = await resContainer.json();

        updateUI(dataContainer.data);
        getLastRead();

        goToCharts();
    } catch (e) {
        alert('El contenedor ingresado no existe, intente de nuevo');
    }
};
```

```
const getLastRead = async () => {
    let endpointDirection = apiEndpoint + id + '/LastRead';

    const resRead = await fetch(endpointDirection);
    const readData = await resRead.json();

    updateLastRead(Object.entries(readData.data));

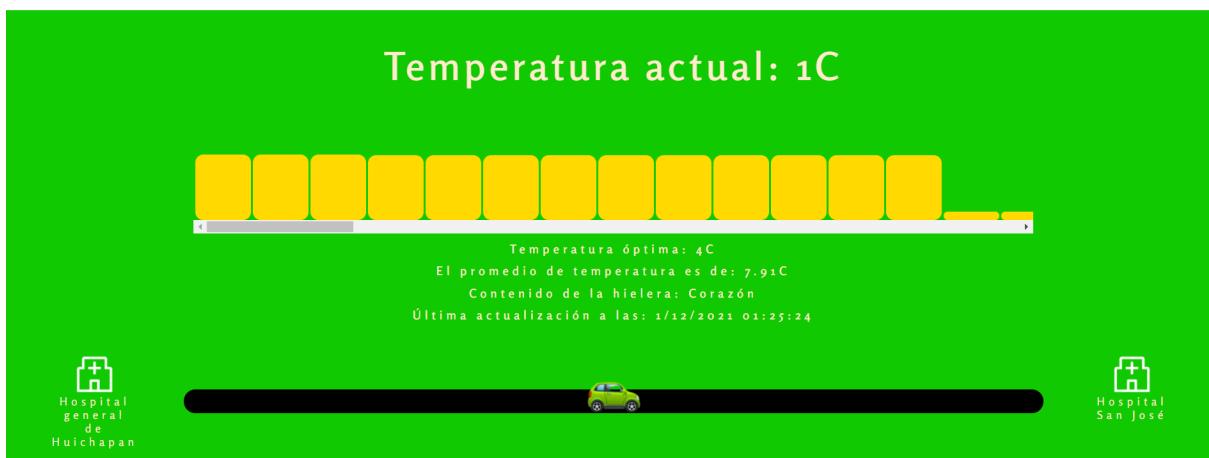
    await timeout(3000);
    getLastRead();
};
```

El resultado final de la página web (visualización) es el siguiente:

# Code Med



# Code Med



## Conclusión y aprendizajes

Durante la realización de este proyecto de internet de las cosas pudimos tener una experiencia inmersiva de distintas áreas de la rama de computación y tecnologías de la información como fue prácticamente la utilización de software y hardware. Esto fue combinado para crear un prototipo funcional conectado a internet que transmita información y esta se pueda administrar en una base de datos para su almacenamiento y crear una página web donde se desplieguen los datos con información útil.

Para ello tuvimos distintos módulos durante el bloque en los cuales aprendíamos las bases de los circuitos, tanto de forma teórica como práctica a la vez que fuimos conociendo la base de datos MySQL e interfaz gráfica del usuario con figma, html y css.

Al tener distintos profesores dedicados a áreas específicas para la implementación del reto de IoT no solo contamos con diversos elementos de aprendizaje para su ejecución sino que colaboramos en equipo para preparar un proyecto exitoso el cual demostraría nuestras habilidades adquiridas y además sirviera como experiencia en todos los aspectos del internet de las cosas.

## Resultados

**Repositorio del servidor y página web:**

<https://github.com/CesarJimenezVilleda02/code-med-node-server>

**Repositorio de implementación de arduino:**

<https://github.com/Andres7702/code-med-arduino-code>

**Video:**

<https://youtu.be/dSGfgV6YX3M>

## Bitácoras

- Pablo César Jiménez Villeda - Backend

<https://drive.google.com/file/d/1A1acKLINEeDTu7wwTYcjMjwrc7zDqoHa/view?usp=sharing>

- Ariann Fernando Arriaga Alcántara- Frontend

<https://docs.google.com/document/d/14ETxzaqQGN4KCIL9xIsMMrJGlibqOIRcDHCr gARIYcl/edit?usp=sharing>

- Alan Fernando Razo Peña - Sensado de datos

<https://docs.google.com/document/d/1pWIupkiEcgMAaoQXA8bEv60Q9vdE5VsHByW pPcf8U2w/edit?usp=sharing>

- Andrés Acevedo Caracheo - Conexión a Internet

[https://docs.google.com/document/d/1J\\_irX-Z32AXoxgGbyvA7a7cvEBkwImTeaEEzM mIvFMo/edit?usp=sharing](https://docs.google.com/document/d/1J_irX-Z32AXoxgGbyvA7a7cvEBkwImTeaEEzM mIvFMo/edit?usp=sharing)

## Referencias

- ¿Qué es el internet de las cosas (IoT)? (n.d.). Red Hat. Retrieved November 5, 2021, from <https://www.redhat.com/es/topics/internet-of-things/what-is-iot>
- Así viajan los órganos durante un trasplante. (2017, December 26). ConSalud. Retrieved November 5, 2021, from [https://www.consalud.es/pacientes/asi-viajan-los-organos-durante-un-trasplante\\_1992\\_7\\_102.html](https://www.consalud.es/pacientes/asi-viajan-los-organos-durante-un-trasplante_1992_7_102.html)
- Firebase. (s.f.) Agrega el SDK de Firebase Admin a tu servidor. Recuperado el 19 de noviembre dle 2021 de: <https://firebase.google.com/docs/admin/setup#windows>
- Node.js.org(s.f.) Acerca de Node.js. Recuperado el 19 de noviembre del 2021 de: <https://nodejs.org/es/about/>