Proyecto3A_Server_Backend

Generado por Doxygen 1.12.0

Capítulo 1

Sensor Management API

Este proyecto es una API para gestionar usuarios y sensores utilizando Node.js y PostgreSQL. Permite la creación, consulta, inserción y eliminación de datos de sensores, así como la administración de usuarios.

1.1. Tabla de Contenidos

1.2. Características

- Gestión de usuarios: crear, listar y eliminar usuarios.
- Gestión de sensores: agregar, listar y eliminar mediciones de sensores.
- API RESTful para interactuar con los datos.
- Base de datos PostgreSQL para almacenamiento persistente.
- Contenerización con Docker.

1.3. Tecnologías Usadas

- Node.js
- Express
- PostgreSQL
- Docker
- Docker Compose

1.4. Estructura del Proyecto

```
```
/backend
    - db.js # Configuración de la conexión a la base de datos.
    - server.js # Código principal de la API.
    - Dockerfile # Archivo para construir la imagen Docker de la aplicación.
    - docker-compose.yml # Archivo de configuración para Docker Compose.
    - package.json # Dependencias y scripts de la aplicación.
    - README.md # Documentación del proyecto.
```
```

1.5. Requisitos Previos

Antes de comenzar, asegúrate de tener instalados los siguientes programas:

- Node.js
- Docker
- Docker Compose

1.6. Configuración del Entorno

1. Clona el repositorio:
   ```
   git clone https://github.com/tu_usuario/sensor-management-api.git
   cd sensor-management-api
   ```
2. Construye y ejecuta los contenedores:
   ```
   docker-compose up --build
   ```

Esto levantará los contenedores para la base de datos y la aplicación.

1.7. Uso

Una vez que los contenedores estén en funcionamiento, la API estará disponible en http://localhost:13000.

1.8. API Endpoints

- GET /setup: Crea las tablas de usuarios y sensores en la base de datos.
- GET /latest: Obtiene las últimas mediciones de temperatura y Ozono.
- GET /: Devuelve todos los usuarios y sensores.
- POST /: Inserta un nuevo sensor.
  - Body: { "type": "temperature", "value": 25.5, "timestamp": "2024-09-22T12:00:00Z", "userId": 1 }
- POST /users: Crea un nuevo usuario.
  - Body: { "username": "nuevo_usuario" }
- DELETE /users/:id/measurements: Elimina todas las mediciones de un usuario específico.
- DELETE /reset: Reinicia las tablas de la base de datos.
- DELETE /erase: Elimina todas las tablas.

1.9. Contribuciones

Las contribuciones son bienvenidas. Si deseas contribuir a este proyecto, por favor abre un issue o envía un pull request.

1.10. Licencia

Este proyecto está bajo la Licencia MIT. Consulta el archivo LICENSE para más información.

Capítulo 2

Índice de archivos

2.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

Capítulo 3

Documentación de archivos

3.1. Referencia del archivo backend/db.js

Configuración de la conexión a la base de datos PostgreSQL.

Variables

- const { Pool } = require('pg')

  Importa el módulo pg para manejar conexiones a PostgreSQL.
- const pool

  Instancia de conexión a la base de datos PostgreSQL.
- module exports = pool

  Exporta el pool de conexiones para que otros módulos lo utilicen.

3.1.1. Descripción detallada

Configuración de la conexión a la base de datos PostgreSQL.

Este archivo configura y exporta un pool de conexiones a PostgreSQL utilizando el módulo pg para gestionar las interacciones con la base de datos.

3.1.2. Documentación de variables

3.1.2.1. const

```
const { Pool } = require('pg')
```

Importa el módulo pg para manejar conexiones a PostgreSQL.

3.1.2.2. exports

```
module exports = pool
```

Exporta el pool de conexiones para que otros módulos lo utilicen.

3.1.2.3. pool

```
const pool
```

Valor inicial:

```
= new Pool({
    host: 'db',
    port: 5432,
    user: 'user123',
    password: 'password123',
    database: 'db123'
})
```

Instancia de conexión a la base de datos PostgreSQL.

El pool de conexiones permite reutilizar conexiones a la base de datos para mejorar el rendimiento. La configuración incluye el host, puerto, usuario, contraseña y nombre de la base de datos.

Parámetros

{string}	host - Dirección del host de la base de datos.
{number}	port - Puerto en el que escucha la base de datos.
{string}	user - Nombre de usuario para la conexión.
{string}	password - Contraseña para la conexión.
{string}	database - Nombre de la base de datos.

3.2. Referencia del archivo backend/emitirAlertas.js

Gestión de alertas basadas en los datos de sensores.

Variables

- const pool = require('./db')

  Conexión al pool de la base de datos PostgreSQL.
- const startAlertas

  Función principal para iniciar la supervisión y generación de alertas.
- module exports = { startAlertas }

  Exporta la función para iniciar la supervisión de alertas.

3.2.1. Descripción detallada

Gestión de alertas basadas en los datos de sensores.

Este archivo contiene la lógica para supervisar las mediciones de sensores, verificar si están dentro de los límites normales y generar alertas para los usuarios afectados.

3.2.2. Documentación de variables

3.2.2.1. exports

```
module exports = { startAlertas }
```

Exporta la función para iniciar la supervisión de alertas.

3.2.2.2. pool

```
const pool = require('./db')
```

Conexión al pool de la base de datos PostgreSQL.

3.2.2.3. startAlertas

```
const startAlertas
```

Función principal para iniciar la supervisión y generación de alertas.

Inicializa los módulos principales.

Supervisa periódicamente las mediciones de sensores y genera alertas si los valores están fuera de los límites definidos. Las alertas se registran en la base de datos y se gestionan por usuario y tipo de sensor.

3.3. Referencia del archivo backend/official_data.js

Integración con la API de calidad del aire para obtener y registrar mediciones oficiales.

Variables

- const axios = require('axios')

  Cliente HTTP para realizar solicitudes a la API.
- const { medicion } = require('./servicios/mediciones')

  Servicio para registrar mediciones en la base de datos.
- const API_URL

  URL de la API de calidad del aire.
- const fetchAirQuality

  Realiza una solicitud a la API y registra los datos obtenidos en el servidor.
- module exports

3.3.1. Descripción detallada

Integración con la API de calidad del aire para obtener y registrar mediciones oficiales.

Este archivo realiza solicitudes periódicas a una API externa para recopilar datos de ozono y temperatura, y registra las mediciones en el servidor.

3.3.2. Documentación de variables

3.3.2.1. API_URL

```
const API_URL
```

Valor inicial:
```
= 'https:


const official_data = () => {
    setInterval(fetchAirQuality, 30000);
}
```

URL de la API de calidad del aire.

Esta API proporciona datos de ozono (O3) y temperatura, junto con otras métricas ambientales.

3.3.2.2. axios

```
const axios = require('axios')
```

Cliente HTTP para realizar solicitudes a la API.

3.3.2.3. const

```
const { medicion } = require('./servicios/mediciones')
```

Servicio para registrar mediciones en la base de datos.

3.3.2.4. exports

```
module exports
```

Valor inicial:
```
= {
    official_data,
    fetchAirQuality
}
```

3.3.2.5. fetchAirQuality

```
const fetchAirQuality
```

Realiza una solicitud a la API y registra los datos obtenidos en el servidor.

Obtiene y registra la calidad del aire.

Este método consulta los datos de ozono y temperatura de la API y los envía al servidor como mediciones oficiales.

3.4. Referencia del archivo backend/server.js

API para gestionar usuarios y sensores, incluyendo funcionalidades de creación, consulta, inserción y eliminación de datos.

Funciones

- app use (cors())

  Habilita CORS para todas las rutas.
- app use (express.json())

  Middleware para parsear cuerpos de solicitudes en formato JSON.
- startAlertas ()

  Inicializa los módulos principales.
- fetchAirQuality ()

  Obtiene y registra la calidad del aire.
- official_data ()

  Configura el monitoreo de datos oficiales.
- resetTables ()

  Reinicia las tablas de la base de datos.
- app use ('/', rutabbdd)

  Configuración de rutas principales.
- app use ('/sensores', rutaSensores)

  Rutas para operaciones con sensores.
- app use ('/', rutaMediciones)

  Rutas para gestionar mediciones.
- app use ('/alertas', rutaAlertas)

  Rutas para gestionar alertas.
- app use ('/', rutaUsuarios)

  Rutas para gestionar usuarios.
- if (require.main===module)

  Inicia el servidor en el puerto especificado.

Variables

- const express = require('express')

  Framework para construir aplicaciones web y APIs.
- const cors = require('cors')

  Middleware para habilitar el intercambio de recursos de origen cruzado.
- const { startAlertas } = require('./emitirAlertas')

  Función para iniciar el sistema de alertas basado en mediciones.
- const rutabbdd = require('./rutas/bbdd')

  Rutas relacionadas con las operaciones generales de la base de datos.
- const rutaSensores = require('./rutas/sensores')

  Rutas para gestionar los sensores.
- const rutaMediciones = require('./rutas/mediciones')

  Rutas para registrar y obtener mediciones.
- const rutaAlertas = require('./rutas/alertas')

  Rutas para gestionar alertas relacionadas con sensores y usuarios.
- const rutaUsuarios = require('./rutas/usuarios')

Rutas para gestionar usuarios.

- const port = 3000

  Puerto en el que el servidor escucha las solicitudes.

- const app = express()

  Instancia principal de la aplicación Express.

- module exports = app

  Exporta la aplicación Express para pruebas o reutilización.

3.4.1. Descripción detallada

API para gestionar usuarios y sensores, incluyendo funcionalidades de creación, consulta, inserción y eliminación de datos.

Este servidor Express gestiona las tablas de usuarios y sensores en una base de datos PostgreSQL. Proporciona rutas para interactuar con las entidades de usuarios, sensores, mediciones, alertas y resetear las tablas.

3.4.2. Documentación de funciones

3.4.2.1. fetchAirQuality()

```
fetchAirQuality ()
```

Obtiene y registra la calidad del aire.

Obtiene y registra la calidad del aire.

Este método consulta los datos de ozono y temperatura de la API y los envía al servidor como mediciones oficiales.

3.4.2.2. if()

```
if (
            require. main = == module)
```

Inicia el servidor en el puerto especificado.

Si este módulo es el principal, lanza el servidor en el puerto definido por port. Muestra un mensaje en la consola indicando que el servidor está en funcionamiento.

3.4.2.3. official_data()

```
official_data ()
```

Configura el monitoreo de datos oficiales.

3.4.2.4. resetTables()

```
resetTables ()
```

Reinicia las tablas de la base de datos.

Reinicia las tablas de la base de datos.

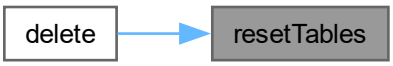
Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al reiniciar las tablas.

Gráfico de llamadas a esta función:



3.4.2.5. startAlertas()

```
startAlertas ()
```

Inicializa los módulos principales.

- startAlertas: Inicia el monitoreo de mediciones para emitir alertas.
- fetchAirQuality y official_data: Obtienen y registran datos de calidad del aire.
- resetTables: Resetea las tablas de la base de datos. Inicia el sistema de alertas.

Inicializa los módulos principales.

Supervisa periódicamente las mediciones de sensores y genera alertas si los valores están fuera de los límites definidos. Las alertas se registran en la base de datos y se gestionan por usuario y tipo de sensor.

3.4.2.6. use() [1/7]

```
app use (
            '/' ,
            rutabbdd )
```

Configuración de rutas principales.

Estas rutas conectan los controladores correspondientes para realizar las operaciones CRUD en las entidades de la base de datos. Rutas para operaciones generales de la base de datos.

3.4.2.7. use() [2/7]

```
app use (
            '/' ,
            rutaMediciones )
```

Rutas para gestionar mediciones.

3.4.2.8. use() [3/7]

```
app use (
            '/' ,
            rutaUsuarios )
```

Rutas para gestionar usuarios.

3.4.2.9. use() [4/7]

```
app use (
            '/alertas' ,
            rutaAlertas )
```

Rutas para gestionar alertas.

3.4.2.10. use() [5/7]

```
app use (
            '/sensores' ,
            rutaSensores )
```

Rutas para operaciones con sensores.

3.4.2.11. use() [6/7]

```
app use (
            cors() )
```

Habilita CORS para todas las rutas.

3.4.2.12. use() [7/7]

```
app use (
            express. json())
```

Middleware para parsear cuerpos de solicitudes en formato JSON.

3.4.3. Documentación de variables

3.4.3.1. app

```
const app = express()
```

Instancia principal de la aplicación Express.

3.4.3.2. const

```
const { startAlertas } = require('./emitirAlertas')
```

Función para iniciar el sistema de alertas basado en mediciones.

Función para reiniciar las tablas de la base de datos.

Funciones para obtener y registrar datos oficiales.

3.4.3.3. cors

```
const cors = require('cors')
```

Middleware para habilitar el intercambio de recursos de origen cruzado.

3.4.3.4. exports

```
module exports = app
```

Exporta la aplicación Express para pruebas o reutilización.

3.4.3.5. express

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

Framework para construir aplicaciones web y APIs.

3.4.3.6. port

```
const port = 3000
```

Puerto en el que el servidor escucha las solicitudes.

3.4.3.7. rutaAlertas

```
const rutaAlertas = require('./rutas/alertas')
```

Rutas para gestionar alertas relacionadas con sensores y usuarios.

3.4.3.8. rutabbdd

```
const rutabbdd = require('./rutas/bbdd')
```

Rutas relacionadas con las operaciones generales de la base de datos.

3.4.3.9. rutaMediciones

```
const rutaMediciones = require('./rutas/mediciones')
```

Rutas para registrar y obtener mediciones.

3.4.3.10. rutaSensores

```
const rutaSensores = require('./rutas/sensores')
```

Rutas para gestionar los sensores.

3.4.3.11. rutaUsuarios

```
const rutaUsuarios = require('./rutas/usuarios')
```

Rutas para gestionar usuarios.

3.5. Referencia del archivo backend/rutas/alertas.js

Funciones

- router delete ('/', async(req, res)=> { try { const result=await borrarAlertas();res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Elimina todas las alertas del sistema.
- router delete ('/:email', async(req, res)=> { const { email }=req.params;try { const result=await borrarAlertas↩ Usuarios(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error↩ :err.message });} })

  Elimina todas las alertas asociadas a un usuario específico.
- router delete ('/:email/:alertaId', async(req, res)=> { const { email, alertaId }=req.params;try { const result=await borrarAlertasEspecificasUsuario(email, alertaId);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Elimina una alerta específica asociada a un usuario.
- router get ('/:email', async(req, res)=> { const { email }=req.params;try { const result=await getAlertas↩ Usuario(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩ message });} })

  Obtiene todas las alertas asociadas a un usuario específico.

Variables

- const express = require('express')
- const { borrarAlertas, borrarAlertasUsuarios, borrarAlertasEspecificasUsuario, getAlertasUsuario } = require('../servicios/alertas')
- const router = express.Router()
- module exports = router

  Exporta el router para ser utilizado en la aplicación principal.

3.5.1. Documentación de funciones

3.5.1.1. delete() [1/3]

```
router delete (
            '/' ,
            async(req, res) ,
            { try { const result=await borrarAlertas();res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Elimina todas las alertas del sistema.

@route DELETE /

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.5.1.2. delete() [2/3]

```
router delete (
            '/:email' ,
            async(req, res) ,
            { const { email }=req.params;try { const result=await borrarAlertasUsuarios(email);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Elimina todas las alertas asociadas a un usuario específico.

@route DELETE /:email

Parámetros

{string}	email - Correo electrónico del usuario cuyas alertas serán eliminadas.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.5.1.3. delete() [3/3]

```
router delete (
            '/:email/:alertaId' ,
            async(req, res) ,
            { const { email, alertaId }=req.params;try { const result=await borrarAlertas↩
EspecificasUsuario(email, alertaId);res.status(200).send(result);} catch(err) { console.↩
error(err);res.status(500).send({ error:err.message });} } )
```

Elimina una alerta específica asociada a un usuario.

@route DELETE /:email/:alertaId

Parámetros

{string}	email - Correo electrónico del usuario.
{string}	alertaId - Identificador de la alerta a eliminar.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.5.1.4. get()

```
router get (
            '/:email' ,
            async(req, res) ,
            { const { email }=req.params;try { const result=await getAlertasUsuario(email);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene todas las alertas asociadas a un usuario específico.

@route GET /:email

Parámetros

{string}	email - Correo electrónico del usuario cuyas alertas serán consultadas.

Devuelve

{Object} Respuesta con las alertas del usuario.

Excepciones

{Error}	Error interno del servidor.

3.5.2. Documentación de variables

3.5.2.1. const

```
const { borrarAlertas, borrarAlertasUsuarios, borrarAlertasEspecificasUsuario, getAlertas↩
Usuario } = require('../servicios/alertas')
```

3.5.2.2. exports

```
module exports = router
```

Exporta el router para ser utilizado en la aplicación principal.

3.5.2.3. express

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

3.5.2.4. router

```
const router = express.Router()
```

3.6. Referencia del archivo backend/servicios/alertas.js

Funciones

- router post ('/mediciones', async(req, res)=> { const { sensorId, valor, timestamp, tipo, location }=req.↩ body;try { const result=await medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea una nueva medición en la base de datos.
- router get ('/latestByEmail/:email', async(req, res)=> { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩ message });} })

  Obtiene la última medición de un usuario por correo electrónico.
- router get ('/mediciones/:fecha', async(req, res)=> { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Obtiene las mediciones realizadas en una fecha específica.
- router post ('/', async(req, res)=> { const { uuid, email }=req.body;try { const result=await createSensor(uuid, email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea un nuevo sensor en la base de datos.
- router post ('/usuarios', async(req, res)=> { const { username, email, password }=req.body;try { const result=await usuarios(username, email, password);res.status(200).send(result);} catch(err) { console.↩ error(err);res.status(500).send({ error:err.message });} })

  Gestión de usuarios y autenticación.
- router get ('/usuarios/:email/sensores', async(req, res)=> { const { email }=req.params;try { const result=await sensoresDeUsuarios(email);res.status(200).send(result);} catch(err) { console.error(err);res.↩ status(500).send({ error:err.message });} })

  Obtiene sensores asociados a un usuario específico.

Variables

- const express = require('express')
- const { medicion, latest, getMediciones } = require('../servicios/mediciones')
- const router = express.Router()

3.6.1. Documentación de funciones

3.6.1.1. get() [1/3]

```
router get (
             '/latestByEmail/:email' ,
             async(req, res) ,
             { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);
catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene la última medición de un usuario por correo electrónico.

@route GET /latestByEmail/:email

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con la última medición del usuario.

Excepciones

{Error}	Error interno del servidor.

3.6.1.2. get() [2/3]

```
router get (
             '/mediciones/:fecha' ,
             async(req, res) ,
             { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene las mediciones realizadas en una fecha específica.

@route GET /mediciones/:fecha

Parámetros

{string}	fecha - Fecha para consultar las mediciones.

Devuelve

{Object} Respuesta con las mediciones realizadas en la fecha.

Excepciones

{Error}	Error interno del servidor.

3.6.1.3. get() [3/3]

```
router.get(
            '/usuarios/:email/sensores',
            async(req, res),
            { const { email }=req.params;try { const result=await sensoresDeUsuarios(email);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene sensores asociados a un usuario específico.

@route GET /usuarios/:email/sensores

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con los sensores asociados al usuario.

Excepciones

{Error}	Error interno del servidor.

3.6.1.4. post() [1/3]

```
router.post(
            '/',
            async(req, res),
            { const { uuid, email }=req.body;try { const result=await createSensor(uuid,
email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({
error:err.message });} } )
```

Crea un nuevo sensor en la base de datos.

@route POST /

Parámetros

{string}	uuid - Identificador único del sensor.
{string}	email - Correo electrónico del usuario propietario del sensor.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.6.1.5. post() [2/3]

```
router post (
            '/mediciones' ,
            async(req, res) ,
            { const { sensorId, valor, timestamp, tipo, location }=req.body;try { const result=await
medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Crea una nueva medición en la base de datos.

@route POST /mediciones

Parámetros

{string}	sensorId - Identificador del sensor.
{number}	valor - Valor medido por el sensor.
{string}	timestamp - Marca de tiempo de la medición.
{string}	tipo - Tipo de medición.
{Object}	location - Ubicación de la medición.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.6.1.6. post() [3/3]

```
router post (
            '/usuarios' ,
            async(req, res) ,
            { const { username, email, password }=req.body;try { const result=await usuarios(username,
email, password);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({
error:err.message });} } )
```

Gestión de usuarios y autenticación.

Este módulo contiene rutas para registrar usuarios, autenticar, gestionar sensores asociados, actualizar información y administrar estados de usuarios.

Registra un nuevo usuario en la base de datos.

@route POST /usuarios

Parámetros

{string}	username - Nombre del usuario.
{string}	email - Correo electrónico del usuario.
{string}	password - Contraseña del usuario.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.6.2. Documentación de variables

3.6.2.1. const

```
const { medicion, latest, getMediciones } = require('../servicios/mediciones')
```

3.6.2.2. express

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

3.6.2.3. router

```
const router = express.Router()
```

3.7. Referencia del archivo backend/rutas/bbdd.js

Funciones

- router post ('/tipos', async(req, res)=> { const { tipo }=req.body;try { const result=await createSensorType(tipo);res.↩ status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea un nuevo tipo de sensor en la base de datos.
- router get ('/setup', async(req, res)=> { try { const result=await setupTables();res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Configura las tablas de la base de datos.
- router delete ('/reset', async(req, res)=> { try { const result=await resetTables();res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Reinicia las tablas de la base de datos.
- router get ('/', async(req, res)=> { try { const result=await getTables();res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Obtiene información sobre las tablas de la base de datos.
- router delete ('/erase', async(req, res)=> { try { const result=await eraseTables();res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Elimina las tablas de la base de datos.

Variables

- const express = require('express')
- const { createSensorType, setupTables, resetTables, getTables, eraseTables } = require('../servicios/bbdd')
- const router = express.Router()
- module exports = router

  Exporta el router para ser utilizado en la aplicación principal.

3.7.1. Documentación de funciones

3.7.1.1. delete() [1/2]

```
router delete (
            '/erase' ,
            async(req, res) ,
            { try { const result=await eraseTables();res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Elimina las tablas de la base de datos.

@route DELETE /erase

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.7.1.2. delete() [2/2]

```
router delete (
            '/reset' ,
            async(req, res) ,
            { try { const result=await resetTables();res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Reinicia las tablas de la base de datos.

@route DELETE /reset

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.7.1.3. get() [1/2]

```
router get (
            '/' ,
            async(req, res) ,
            { try { const result=await getTables();res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene información sobre las tablas de la base de datos.

@route GET /

Devuelve

{Object} Respuesta con los datos de las tablas.

Excepciones

{Error}	Error interno del servidor.

3.7.1.4. get() [2/2]

```
router get (
            '/setup' ,
            async(req, res) ,
            { try { const result=await setupTables();res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Configura las tablas de la base de datos.

@route GET /setup

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.7.1.5. post()

```
router post (
            '/tipos' ,
            async(req, res) ,
            { const { tipo }=req.body;try { const result=await createSensorType(tipo);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Crea un nuevo tipo de sensor en la base de datos.

@route POST /tipos

Parámetros

{string}	tipo - Nombre del tipo de sensor a crear.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.7.2. Documentación de variables

3.7.2.1. const

```
const { createSensorType, setupTables, resetTables, getTables, eraseTables } = require('../servicios/bbdd')
```

3.7.2.2. exports

```
module exports = router
```

Exporta el router para ser utilizado en la aplicación principal.

3.7.2.3. express

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

3.7.2.4. router

```
const router = express.Router()
```

3.8. Referencia del archivo backend/servicios/bbdd.js

Variables

- const pool = require('../db')
- const createSensorType

  Crea un nuevo tipo de sensor en la base de datos.
- const setupTables

  Crea las tablas necesarias en la base de datos.
- const resetTables

  Reinicia las tablas de la base de datos y las llena con datos predeterminados.
- const getTables

  Obtiene los datos de todas las tablas de la base de datos.
- const eraseTables

  Elimina todas las tablas de la base de datos.
- module exports = { createSensorType, setupTables, resetTables, getTables, eraseTables }

3.8.1. Documentación de variables

3.8.1.1. createSensorType

```
const createSensorType
```

Valor inicial:

```
= async (tipo) => {
    try {
        await pool.query('INSERT INTO tipos (tipo) VALUES ($1)', [tipo]);
        return { message: "Sensor type created successfully" };
    } catch (err) {
        throw new Error("Error creating sensor type");
    }
}
```

Crea un nuevo tipo de sensor en la base de datos.

Parámetros

{string}	tipo - Nombre del tipo de sensor.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al crear el tipo de sensor.

3.8.1.2. eraseTables

```
const eraseTables
```

Valor inicial:

```
= async () => {
    try {
        await pool.query(`
            DROP TABLE IF EXISTS mediciones CASCADE;
            DROP TABLE IF EXISTS usuario_sensores CASCADE;
            DROP TABLE IF EXISTS alertas_usuarios CASCADE;
            DROP TABLE IF EXISTS sensores CASCADE;
            DROP TABLE IF EXISTS actividad CASCADE;
            DROP TABLE IF EXISTS usuarios CASCADE;
            DROP TABLE IF EXISTS alertas CASCADE;
            DROP TABLE IF EXISTS tipos CASCADE;
        `);
        return { message: "Successfully erased all tables" };
    } catch (err) {
        console.log(err);
        throw new Error("Error al devolver las tablas: " + err);
    }
}
```

Elimina todas las tablas de la base de datos.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al borrar las tablas.

3.8.1.3. exports

```
module exports = { createSensorType, setupTables, resetTables, getTables, eraseTables }
```

3.8.1.4. getTables

```
const getTables
```

Valor inicial:

```
= async () => {
    try {
        const tiposQuery = await pool.query('SELECT * FROM tipos');
        const medicionesQuery = await pool.query('SELECT * FROM mediciones');
        const sensoresQuery = await pool.query('SELECT * FROM sensores');
        const usuariosQuery = await pool.query('SELECT * FROM usuarios');
        const actividadQuery = await pool.query('SELECT * FROM actividad');
        const usrSensQuery = await pool.query('SELECT * FROM usuario_sensores');
        const usrAlertsQuery = await pool.query('SELECT * FROM alertas_usuarios');
        const alertasQuery = await pool.query('SELECT * FROM alertas');

        return {
            tipos: tiposQuery.rows,
            mediciones: medicionesQuery.rows,
            sensores: sensoresQuery.rows,
            usuarios: usuariosQuery.rows,
            actividad: actividadQuery.rows,
            usuario_sensores: usrSensQuery.rows,
            alertas_usuarios: usrAlertsQuery.rows,
            alertas: alertasQuery.rows
        };
    } catch (err) {
        console.log(err);
        throw new Error("Error al devolver las tablas: " + err);
    }
}
```

Obtiene los datos de todas las tablas de la base de datos.

Devuelve

{Object} Datos de las tablas.

Excepciones

{Error}	Error al obtener los datos de las tablas.

3.8.1.5. pool

```
const pool = require('../db')
```

3.8.1.6. resetTables

```
const resetTables
```

Valor inicial:

```
= async () => {
    try {
        await pool.query(`
            DROP TABLE IF EXISTS mediciones CASCADE;
            DROP TABLE IF EXISTS usuario_sensores CASCADE;
            DROP TABLE IF EXISTS alertas_usuarios CASCADE;
            DROP TABLE IF EXISTS sensores CASCADE;
            DROP TABLE IF EXISTS actividad CASCADE;
            DROP TABLE IF EXISTS usuarios CASCADE;
            DROP TABLE IF EXISTS alertas CASCADE;
            DROP TABLE IF EXISTS tipos CASCADE;
        `);

        await setupTables();

        await pool.query(`INSERT INTO tipos (tipo) VALUES ('temperature'), ('ozono');`);
        await pool.query(`INSERT INTO sensores (uuid) VALUES ('sensor-uuid-1'), ('sensor-uuid-2');`);

        return { message: "Successfully reset tables with default data" };
    } catch (err) {
        console.log(err);
        throw new Error("Error reseteando las tablas: " + err);
    }
}
```

Reinicia las tablas de la base de datos y las llena con datos predeterminados.

Reinicia las tablas de la base de datos.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al reiniciar las tablas.

3.8.1.7. setupTables

```
const setupTables
```

Crea las tablas necesarias en la base de datos.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al crear las tablas.

3.9. Referencia del archivo backend/rutas/mediciones.js

Funciones

- router post ('/mediciones', async(req, res)=> { const { sensorId, valor, timestamp, tipo, location }=req.↩ body;try { const result=await medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea una nueva medición en la base de datos.
- router get ('/latestByEmail/:email', async(req, res)=> { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩ message });} })

  Obtiene la última medición de un usuario por correo electrónico.
- router get ('/mediciones/:fecha', async(req, res)=> { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Obtiene las mediciones realizadas en una fecha específica.

Variables

- const express = require('express')
- const { medicion, latest, getMediciones } = require('../servicios/mediciones')
- const router = express.Router()
- module exports = router

  Exporta el router para ser utilizado en la aplicación principal.

3.9.1. Documentación de funciones

3.9.1.1. get() [1/2]

```
router get (
            '/latestByEmail/:email' ,
            async(req, res) ,
            { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);
catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene la última medición de un usuario por correo electrónico.

@route GET /latestByEmail/:email

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con la última medición del usuario.

Excepciones

{Error}	Error interno del servidor.

3.9.1.2. get() [2/2]

```
router.get('/mediciones/:fecha', async(req, res), { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene las mediciones realizadas en una fecha específica.

@route GET /mediciones/:fecha

Parámetros

{string}	fecha - Fecha para consultar las mediciones.

Devuelve

{Object} Respuesta con las mediciones realizadas en la fecha.

Excepciones

{Error}	Error interno del servidor.

3.9.1.3. post()

```
router.post('/mediciones', async(req, res), { const { sensorId, valor, timestamp, tipo, location }=req.body;try { const result=await medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Crea una nueva medición en la base de datos.

@route POST /mediciones

Parámetros

{string}	sensorId - Identificador del sensor.
{number}	valor - Valor medido por el sensor.
{string}	timestamp - Marca de tiempo de la medición.
{string}	tipo - Tipo de medición.
{Object}	location - Ubicación de la medición.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.9.2. Documentación de variables

3.9.2.1. const

```
const { medicion, latest, getMediciones } = require('../servicios/mediciones')
```

3.9.2.2. exports

```
module exports = router
```

Exporta el router para ser utilizado en la aplicación principal.

3.9.2.3. express

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

3.9.2.4. router

```
const router = express.Router()
```

3.10. Referencia del archivo backend/servicios/mediciones.js

Variables

- const pool = require('../db')
- const medicion

  Agrega una nueva medición a la base de datos.
- const latest

  Obtiene las últimas mediciones de temperatura y ozono para un usuario.
- const getMediciones

  Obtiene mediciones realizadas en una fecha específica.
- module exports = { medicion, latest, getMediciones }

3.10.1. Documentación de variables

3.10.1.1. exports

```
module exports = { medicion, latest, getMediciones }
```

3.10.1.2. getMediciones

```
const getMediciones
```

Valor inicial:

```
= async (fecha) => {
    try {
        const query = 'SELECT * FROM mediciones WHERE CAST(timestamp AS DATE) = CAST($1 AS DATE)';
        const medicionesQuery = await pool.query(query, [fecha]);

        return medicionesQuery.rows;
    } catch (err) {
        console.log(err);
        throw new Error("Error al devolver las tablas: " + err);
    }
}
```

Obtiene mediciones realizadas en una fecha específica.

Parámetros

{string}	fecha - Fecha en formato YYYY-MM-DD.

Devuelve

{Array} Lista de mediciones realizadas en la fecha proporcionada.

Excepciones

{Error}	Error al obtener las mediciones.

3.10.1.3. latest

```
const latest
```

Obtiene las últimas mediciones de temperatura y ozono para un usuario.

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Últimas mediciones de temperatura y ozono.

Excepciones

{Error}	Error si no se encuentran mediciones o al ejecutar la consulta.

3.10.1.4. medicion

```
const medicion
```

Valor inicial:

```
= async (sensorId, valor, timestamp, tipo, location) => {
    if (!sensorId || !valor || !timestamp || !tipo || !location) {
        throw new Error("Faltan parámetros necesarios");
    }

    try {
        const sensorResult = await pool.query('SELECT * FROM sensores WHERE uuid = $1', [sensorId]);
        if (sensorResult.rows.length === 0) {
            throw new Error("Sensor no encontrado");
        }

        await pool.query(
            'INSERT INTO mediciones (sensor_id, valor, timestamp, tipo, location) VALUES ($1, $2, $3, $4,
      $5)',
            [sensorId, valor, timestamp, tipo, location]
        );

        return { message: "Medición agregada exitosamente" };
    } catch (err) {
        console.error('Error al insertar la medición:', err);
        throw new Error("Error al insertar la medición: " + err);
    }
}
```

Agrega una nueva medición a la base de datos.

Parámetros

{string}	sensorId - Identificador único del sensor.
{number}	valor - Valor medido por el sensor.
{string}	timestamp - Fecha y hora de la medición.
{number}	tipo - Tipo de medición (1: temperatura, 2: ozono).
{string}	location - Ubicación de la medición en formato POINT (latitud, longitud).

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error si faltan parámetros o al insertar la medición.

3.10.1.5. pool

```
const pool = require('../db')
```

3.11. Referencia del archivo backend/rutas/sensores.js

Funciones

- router post ('/mediciones', async(req, res)=> { const { sensorId, valor, timestamp, tipo, location }=req.↩ body;try { const result=await medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea una nueva medición en la base de datos.
- router get ('/latestByEmail/:email', async(req, res)=> { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩ message });} })

  Obtiene la última medición de un usuario por correo electrónico.
- router get ('/mediciones/:fecha', async(req, res)=> { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Obtiene las mediciones realizadas en una fecha específica.
- router post ('/', async(req, res)=> { const { uuid, email }=req.body;try { const result=await createSensor(uuid, email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea un nuevo sensor en la base de datos.

Variables

- const express = require('express')
- const { medicion, latest, getMediciones } = require('../servicios/mediciones')
- const router = express.Router()
- module exports = router

  Exporta el router para ser utilizado en la aplicación principal.

3.11.1. Documentación de funciones

3.11.1.1. get() [1/2]

```
router get (
            '/latestByEmail/:email' ,
            async(req, res) ,
            { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);
catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene la última medición de un usuario por correo electrónico.

@route GET /latestByEmail/:email

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con la última medición del usuario.

Excepciones

{Error}	Error interno del servidor.

3.11.1.2. get() [2/2]

```
router get (
            '/mediciones/:fecha' ,
            async(req, res) ,
            { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene las mediciones realizadas en una fecha específica.

@route GET /mediciones/:fecha

Parámetros

{string}	fecha - Fecha para consultar las mediciones.

Devuelve

{Object} Respuesta con las mediciones realizadas en la fecha.

Excepciones

{Error}	Error interno del servidor.

3.11.1.3. post() [1/2]

```
router post (
            '/' ,
            async(req, res) ,
            { const { uuid, email }=req.body;try { const result=await createSensor(uuid,
email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({
error:err.message });} } )
```

Crea un nuevo sensor en la base de datos.

@route POST /

Parámetros

{string}	uuid - Identificador único del sensor.
{string}	email - Correo electrónico del usuario propietario del sensor.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.11.1.4. post() [2/2]

```
router post (
            '/mediciones' ,
            async(req, res) ,
            { const { sensorId, valor, timestamp, tipo, location }=req.body;try { const result=await
medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Crea una nueva medición en la base de datos.

@route POST /mediciones

Parámetros

{string}	sensorId - Identificador del sensor.
{number}	valor - Valor medido por el sensor.
{string}	timestamp - Marca de tiempo de la medición.
{string}	tipo - Tipo de medición.
{Object}	location - Ubicación de la medición.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.11.2. Documentación de variables

3.11.2.1. const

```
const { medicion, latest, getMediciones } = require('../servicios/mediciones')
```

3.11.2.2. exports

```
module exports = router
```

Exporta el router para ser utilizado en la aplicación principal.

3.11.2.3. express

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

3.11.2.4. router

```
const router = express.Router()
```

3.12. Referencia del archivo backend/servicios/sensores.js

Variables

- const pool = require('../db')
- const createSensor

  Crea un nuevo sensor y lo asocia a un usuario en la base de datos.
- module exports = { createSensor }

3.12.1. Documentación de variables

3.12.1.1. createSensor

```
const createSensor
```

Valor inicial:

```
= async (uuid, email) => {
    try {

        await pool.query('INSERT INTO sensores (uuid) VALUES ($1)', [uuid]);


        await pool.query('INSERT INTO usuario_sensores (usuario_email, sensor_uuid) VALUES ($1, $2)',
      [email, uuid]);

        return { message: "Sensor created successfully" };
    } catch (err) {
        console.log(err);
        throw new Error("Error creating sensor: " + err);
    }
}
```

Crea un nuevo sensor y lo asocia a un usuario en la base de datos.

Parámetros

{string}	uuid - Identificador único del sensor.
{string}	email - Correo electrónico del usuario al que se asociará el sensor.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al crear el sensor o asociarlo al usuario.

3.12.1.2. exports

```
module exports = { createSensor }
```

3.12.1.3. pool

```
const pool = require('../db')
```

3.13. Referencia del archivo backend/rutas/usuarios.js

Funciones

- router post ('/mediciones', async(req, res)=> { const { sensorId, valor, timestamp, tipo, location }=req.↩ body;try { const result=await medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Crea una nueva medición en la base de datos.
- router get ('/latestByEmail/:email', async(req, res)=> { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩ message });} })

  Obtiene la última medición de un usuario por correo electrónico.
- router get ('/mediciones/:fecha', async(req, res)=> { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.message });} })

  Obtiene las mediciones realizadas en una fecha específica.
- router post ('/usuarios', async(req, res)=> { const { username, email, password }=req.body;try { const result=await usuarios(username, email, password);res.status(200).send(result);} catch(err) { console.↩ error(err);res.status(500).send({ error:err.message });} })

  Gestión de usuarios y autenticación.
- router get ('/usuarios/:email/sensores', async(req, res)=> { const { email }=req.params;try { const result=await sensoresDeUsuarios(email);res.status(200).send(result);} catch(err) { console.error(err);res.↩ status(500).send({ error:err.message });} })

  Obtiene sensores asociados a un usuario específico.
- router get ('/usuarios/login/:email/:password', async(req, res)=> { const { email, password }=req.params;try { const result=await usuariosAutent(email, password);res.status(200).send(result);} catch(err) { console.↩ error(err);res.status(500).send({ error:err.message });} })

  Autentica un usuario mediante correo electrónico y contraseña.

Variables

- const express = require('express')
- const { medicion, latest, getMediciones, usuarios, sensoresDeUsuarios, usuariosAutent, actualizarUsuarios, verificarToken, verificarActualizacion, actualizarContrasena, usuarioAdmin, infoUsers } = require('../servicios/mediciones')
- const router = express.Router()

3.13.1. Documentación de funciones

3.13.1.1. get() [1/4]

```
router get (
            '/latestByEmail/:email' ,
            async(req, res) ,
            { const { email }=req.params;try { const result=await latest(email);res.status(200).send(result);
catch(err) { console.error(err);res.status(500).send({ error:err.message });} } )
```

Obtiene la última medición de un usuario por correo electrónico.

@route GET /latestByEmail/:email

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con la última medición del usuario.

Excepciones

{Error}	Error interno del servidor.

3.13.1.2. get() [2/4]

```
router get (
            '/mediciones/:fecha' ,
            async(req, res) ,
            { const { fecha }=req.params;try { const result=await getMediciones(fecha);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene las mediciones realizadas en una fecha específica.

@route GET /mediciones/:fecha

Parámetros

{string}	fecha - Fecha para consultar las mediciones.

Devuelve

{Object} Respuesta con las mediciones realizadas en la fecha.

Excepciones

{Error}	Error interno del servidor.

3.13.1.3. get() [3/4]

```
router.get(
            '/usuarios/:email/sensores',
            async(req, res),
            { const { email }=req.params;try { const result=await sensoresDeUsuarios(email);res.↩
status(200).send(result);} catch(err) { console.error(err);res.status(500).send({ error:err.↩
message });} } )
```

Obtiene sensores asociados a un usuario específico.

@route GET /usuarios/:email/sensores

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Object} Respuesta con los sensores asociados al usuario.

Excepciones

{Error}	Error interno del servidor.

3.13.1.4. get() [4/4]

```
router.get(
            '/usuarios/login/:email/:password',
            async(req, res),
            { const { email, password }=req.params;try { const result=await usuariosAutent(email,
password);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({
error:err.message });} } )
```

Autentica un usuario mediante correo electrónico y contraseña.

@route GET /usuarios/login/:email/:password

Parámetros

{string}	email - Correo electrónico del usuario.
{string}	password - Contraseña del usuario.

Devuelve

{Object} Respuesta con los datos de autenticación.

Excepciones

{Error}	Error interno del servidor.

3.13.1.5. post() [1/2]

```
router post (
            '/mediciones' ,
            async(req, res) ,
            { const { sensorId, valor, timestamp, tipo, location }=req.body;try { const result=await
medicion(sensorId, valor, timestamp, tipo, location);res.status(200).send(result);} catch(err)
{ console.error(err);res.status(500).send({ error:err.message });} } )
```

Crea una nueva medición en la base de datos.

@route POST /mediciones

Parámetros

{string}	sensorId - Identificador del sensor.
{number}	valor - Valor medido por el sensor.
{string}	timestamp - Marca de tiempo de la medición.
{string}	tipo - Tipo de medición.
{Object}	location - Ubicación de la medición.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.13.1.6. post() [2/2]

```
router post (
            '/usuarios' ,
            async(req, res) ,
            { const { username, email, password }=req.body;try { const result=await usuarios(username,
email, password);res.status(200).send(result);} catch(err) { console.error(err);res.status(500).send({
error:err.message });} } )
```

Gestión de usuarios y autenticación.

Este módulo contiene rutas para registrar usuarios, autenticar, gestionar sensores asociados, actualizar información y administrar estados de usuarios.

Registra un nuevo usuario en la base de datos.

@route POST /usuarios

Parámetros

{string}	username - Nombre del usuario.
{string}	email - Correo electrónico del usuario.
{string}	password - Contraseña del usuario.

Devuelve

{Object} Respuesta con el resultado de la operación.

Excepciones

{Error}	Error interno del servidor.

3.13.2. Documentación de variables

3.13.2.1. const

```
const { medicion, latest, getMediciones, usuarios, sensoresDeUsuarios, usuariosAutent, actualizarUsuarios,
verificarToken, verificarActualizacion, actualizarContrasena, usuarioAdmin, infoUsers } =
require('../servicios/mediciones')
```

3.13.2.2. express

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

3.13.2.3. router

```
const router = express.Router()
```

3.14. Referencia del archivo backend/servicios/usuarios.js

Variables

- const pool = require('../db')
- const { hashPassword, verifyPassword } = require("../utilidades/security")
- const nodemailer = require('nodemailer')
- const usuarios

  Crea un nuevo usuario en la base de datos y envía un correo de verificación.
- const sensoresDeUsuarios

  Obtiene los sensores asociados a un usuario.
- const usuariosAutent

  Autentica a un usuario.
- const actualizarUsuarios

  Actualiza la información de un usuario y envía un correo de confirmación.
- module exports = { usuarios, sensoresDeUsuarios, usuariosAutent, actualizarUsuarios }

3.14.1. Documentación de variables

3.14.1.1. actualizarUsuarios

```
const actualizarUsuarios
```

Actualiza la información de un usuario y envía un correo de confirmación.

Parámetros

{string}	username - Nuevo nombre de usuario.
{string}	email - Correo electrónico del usuario.
{string}	password - Nueva contraseña del usuario.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al actualizar la información del usuario o enviar el correo de confirmación.

3.14.1.2. const

```
const { hashPassword, verifyPassword } = require("../utilidades/security")
```

3.14.1.3. exports

```
module exports = { usuarios, sensoresDeUsuarios, usuariosAutent, actualizarUsuarios }
```

3.14.1.4. nodemailer

```
const nodemailer = require('nodemailer')
```

3.14.1.5. pool

```
const pool = require('../db')
```

3.14.1.6. sensoresDeUsuarios

```
const sensoresDeUsuarios
```

Valor inicial:

```
= async (email) => {
    try {
        const result = await pool.query(`
            SELECT s.uuid, s.id
            FROM sensores s
            JOIN usuario_sensores us ON s.uuid = us.sensor_uuid
            WHERE us.usuario_email = $1
        `, [email]);

        if (result.rows.length === 0) {
            throw new Error("Usuario no encontrado o sin sensores asignados");
        }

        return result.rows;
    } catch (err) {
        console.error(err);
        throw new Error(err);
    }
}
```

Obtiene los sensores asociados a un usuario.

Parámetros

{string}	email - Correo electrónico del usuario.

Devuelve

{Array} Lista de sensores asociados al usuario.

Excepciones

{Error}	Error al obtener los sensores o si el usuario no tiene sensores asignados.

3.14.1.7. usuarios

```
const usuarios
```

Crea un nuevo usuario en la base de datos y envía un correo de verificación.

Parámetros

{string}	username - Nombre del usuario.
{string}	email - Correo electrónico del usuario.
{string}	password - Contraseña del usuario.

Devuelve

{Object} Mensaje de éxito.

Excepciones

{Error}	Error al crear el usuario o enviar el correo de verificación.

3.14.1.8. usuariosAutent

```
const usuariosAutent
```

Valor inicial:

```
= async (email, password) => {
    try {
        const userResult = await pool.query('SELECT * FROM usuarios WHERE email = $1', [email]);
        if (userResult.rows.length === 0) {
            throw new Error("Email o contraseña incorrectos");
        }
        const user = userResult.rows[0];
        if (!user.is_verified) {
            throw new Error("Please verify your account to log in");
        }

        const isPasswordValid = await verifyPassword(password, user.password);
        if (!isPasswordValid) {
            throw new Error("Email o contraseña incorrectos");
        }

        return user;
    } catch (err) {
        console.error(err);
        throw new Error(err);
    }
}
```

Autentica a un usuario.

Parámetros

{string}	email - Correo electrónico del usuario.
{string}	password - Contraseña del usuario.

Devuelve

{Object} Datos del usuario autenticado.

Excepciones

{Error}	Error si las credenciales son incorrectas o la cuenta no está verificada.

3.15. Referencia del archivo backend/utilidades/security.js

Variables

- const bcrypt = require('bcryptjs')
- const hashPassword
- const verifyPassword
- module exports

3.15.1. Documentación de variables

3.15.1.1. bcrypt

```
const bcrypt = require('bcryptjs')
```

3.15.1.2. exports

```
module exports
```

Valor inicial:

```
= {
    hashPassword,
    verifyPassword
}
```

3.15.1.3. hashPassword

```
const hashPassword
```

Valor inicial:

```
= async (password) => {
    const saltRounds = 10;
    return await bcrypt.hash(password, saltRounds);
}
```

3.15.1.4. verifyPassword

```
const verifyPassword
```

Valor inicial:

```
= async (password, hashedPassword) => {
    return await bcrypt.compare(password, hashedPassword);
}
```

3.16. Referencia del archivo README.md